ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Review on Software Reliability

Damanjit Kaur[*1], Inderpal Singh[2]
[*1,2] Department Computer Science and Engineering, DAV Institute of Engineering and Technology, Jalandhar (Pb.), India
damanjitaulakh@gmail.com

Abstract

Software reliability signifies the probability that software in a pre-defined condition executes its tasks without malfunctioning for a specified duration. It may be regarded as a component of software quality. Unlike software quality, however, it concentrates on the functionality of the software and disregards such issues as ergonomics of software products, development economics, etc. unless they constitute functional attributes of the software product.

To most project and software development managers, reliability is equated to correctness, that is, they look to testing and the number of "bugs" found and fixed. While finding and fixing bugs discovered in testing is necessary to assure reliability, a better way is to develop a robust, high quality product through all of the stages of the software lifecycle. That is, the reliability of the delivered code is related to the quality of all of the processes and products of software development; the requirements documentation, the code, test plans, and testing.

Keywords: Software reliability, software quality, bugs, test plans, and testing.

Introduction

In order to express the reliability of a software product quantitatively, first, the product itself must be "measured". For this purpose, the abstraction of measurement has to be removed. This can be achieved by defining certain measures, or metrics, about software product and its development process. Once reliability metrics are defined, it is wise to question if it is possible to determine and improve the reliability of software with a system based on these metrics.

Software reliability is comprised of three activities:

1. Error prevention
2. Fault detection and removal
3. Measurements to maximize reliability, specifically measures that support the first two activities.

Defect: A product anomaly. Examples include such things as (1) omissions and imperfections found during early life cycle phases and (2) faults contained in software sufficiently mature for test or operation.

Fault: (1) An accidental condition that causes a functional unit to fail to perform its required function. (2) A manifestation of an error in software. A fault, if encountered, may cause a failure. It is synonymous with 'bug'.

Failure: (1) The termination of the ability of a functional unit to perform its required function. (2) An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered.

Error: Human action that results in software containing a fault. Examples include omission of misinterpretation of user requirements in a software specification, incorrect translation, or omission of a requirement in the design specification.

Measure: A quantitative assessment of the degree to which a software product or process possesses a given attribute.

Assessment of Software Reliability

Apart from classical hardware reliability, software reliability has rather different nature [2, 3, 4]. While the reliability of hardware continues to change even after the product is delivered, the reliability of software is improved throughout the development process until the product is delivered.

Another major difference between software reliability and hardware reliability is that software reliability is not a function of how frequent that specific software is used; whereas hardware is subject to wear out [4,5]. Also, because software is rather conceptual, documentation is considered as an integral part of software and software reliability [3].

ISSN: 2277-9655
Impact Factor: 1.852

A common constituent of hardware and software reliability techniques is testing [4]. The results of testing process are employed in software reliability growth models to translate defect and/or failure data into reliability measures [6,7]. Because of all these common points and differences mentioned, it is wise to classify studies on assessment of software reliability into two groups: Software Reliability Modeling, and Software Testing.

To increase the reliability by preventing software errors, the focus must be on comprehensive requirements and a comprehensive testing plan, ensuring all requirements are tested. Focus also must be on the maintainability of the software since there will be a "useful life" phase where sustaining engineering will be needed. Therefore, to prevent software errors, we must:

1. Start with the requirements, ensuring the product developed is the one specified, that allrequirements clearly and accurately specify the final product functionality.
2. Ensure the code can easily support sustaining engineering without infusing additionalerrors.

Metric Collection Systems

If reliability is essential, then it has to be controllable. The necessary controlprocess has to be based on observations or measurements. Because the raw materialof these measurements may be defined differently from one organization to another,a generalized method of observation or measurement is needed. Metric collectionsystems are the answers to this need.

The process of creation of a software metric collection system is defined by [8] asof six successive steps. These steps are:

1. Documentation of the software development process
2. Statement of the purpose of the metric collection system
3. Determination of the metrics required to be collected in order to reach specific purposes
4. Identification of the data to be collected
5. Definition of the procedures to obtain data from the organization andprojects
6. Coding of the designed overall system.

Reliability Growth Models

Increasingly software plays a critical part in not only scientific and business related enterprises, but in daily life where it runs devices such as cars, phones, and television sets. Although advances have been made towards the production of defect free software, any software required to operate reliably must still undergo extensive testing and debugging. This can be a costly and time consuming process, and managers require accurate information about how software reliability grows as a result of this process in order to effectively manage their Budgets and projects.

The effects of this process, by which it is hoped software is made more reliable, can be modeled through the use of Software Reliability Growth Models, hereafter referred to as SRGMs. Ideally, these models provide a means of characterizing the development process and enable software reliability practitioners to make predictions about the expected future reliability of software under development. Such techniques allow managers to accurately allocate time, money, and human resources to a project, and assess when a piece of software has reached a point where it can be released with some level of confidence in its reliability. Unfortunately, these models are often inaccurate.

All of the models examined here have two parameters. Regardless of how these models where originally formulated, we will refer to the parameters of these models as 0 and 1. When necessary, we will use a superscript to differentiate between parameters of different models. For example, $E0$ will refer to the 0 parameter of the exponential model, and $L1$ will refer to the 1 parameter of the logarithmic model. Standard practice is to determine the values of these parameters by fitting the model in question to the available data; we will examine the various means for doing so in chapter 2. Once the model has been fitted to the data, it can then be used to obtain estimates of current stability of the software and make predictions about the programs future reliability.

The Exponential Model

The most widely used software reliability growth model is the exponential model. This is a stochastic model based on a non-homogeneous Poisson process (Goel and Okumoto 1979). Originally proposed by Jelinski and Moranda in (Jelinski and Moranda 1971),many variations have since appeared. The original JM-exponential model made use of the elapsed wall clock time when a failure was encountered. A significant refinement was made by Musa, who restated the model in terms of CPU execution time allowing for more accurate predictions. Musa also described a method for moving between execution time and wall clock time, making it easier to make predictions in terms of real world calendars and deadlines (Musa 1975). Latter, Goel and Okumoto worked to generalize the model, allowing the initial number of errors in a program to be random rather than fixed; and permitting errors to be independent (Goel and Okumoto 1979). Although superior to the earlier models, it has been shown that the exponential

ISSN: 2277-9655
Impact Factor: 1.852

model is not generally the most accurate SRGM (Malaiya, Karunanithi, and Verma 1992). However, this model remains popular and widely used.

Theexponential model, in the formulation used here is also termed Musa's basic execution model [17]. It isgiven by:

$$\mu(t) = \beta_0[1 - e^{-\beta_1 t}]$$
$$\lambda(t) = \beta_0\beta_1 e^{-\beta_1 t}$$

The Logarithmic Model

The logarithmic model was originally proposed by Musa and Okumoto in (Musa and Okumoto 1984). Like theExponential model, it models the failure process as a non-homogeneous Poisson process. The most significantdifference between this model and the exponential is that the logarithmic model assumes that failure intensitywill decrease exponentially with the expected number of failures experienced, while the exponential modelassumes an equal reduction in failure intensity with each fault uncovered and corrected. In this sense it canbe viewed as a continuous formulation of the geometric model (Musa and Okumoto 1984).In (Malaiya, Karunanithi, and Verma 1992) it was shown that the logarithmic model was generally moreaccurate than many other SRGMs. It is relatively simple to use, although not as widely used as the exponentialmodel. This may be due in part to the difficulty of obtaining a concrete interpretation of the model'sparameters. The logarithmic model takes the form:

$$\mu(t) = \beta_0 \ln(1 + \beta_1 t)$$
$$\lambda(t) = \frac{\beta_0\beta_1}{1 + \beta_1 t}$$

Quality and Software Reliability

Software reliability is considered as an important metric for software quality [1, 3, 9, 5]. In [10], however, Voas indicates that highly reliable software is not necessarily a high-quality product, as there exist situations in which ultra-reliable software systems showed performance degradations, poor robustness and lack of maintenance precautions. An approach proposed to make reliability estimations and predictions parallel to quality is to organize the testing process in such a way to make the user requirements tested more strictly with increased frequency of repetition of revealing input set [2, 11]. The essence of this technique is that most of the time the user is not interested in how the problem was solved; he/she wants to see that the proposed solution is the one that meets the requirements.

The problem with the method mentioned above is that exception handling is not always considered when such testing scenarios are created [8]. Especially in the case of safety-critical software, it is difficult to determine the test cases that lead the exception handling routines to run [12]. In [14] it is claimed that aspect-oriented programming improves reliability by its nature providing direct control over exception handling.

Another way of improvement of quality and reliability of software systems is the code-inspection [15]. There are examples of checklists for improvement of quality of code-inspection process [13].

Conclusion

Metrics to measure software reliability do exist and can be used starting in the requirements phase. At each phase of the development life cycle, metrics can identify potential areas of problems that may lead to problems or errors. Finding these areas in the phase they are developed decreases the cost and prevents potential ripple effects from the changes, later in the development life cycle. Metrics used early can aid in detection and correction of requirement faults that will lead to prevention of errors later in the life cycle. We also have examined the exponential and logarithmic models. The results on the logarithmic model are more difficult to interpret.

References

[1] "IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary ofMeasures to Produce Reliable Software", 1998

[2] L. Rosenberg, T. Hammer, J. Shaw, "Software Metrics andReliability",http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/software_metrics_and_reliability.htmlLast Date Accessed: 25.06.2013

[3] "MIL-HDBK-338B, Electronic Reliability Design Handbook", US DoD

[4] J. D. Musa, "A Theory of Software Reliability and Its Applications", September1975, IEEE Transactions on Software Engineering, Volume: SE-1, No: 3, pp.312-327

[5] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, andApplicability", December 1975, IEEE Transactions on Software Engineering, Volume: SE-11 No: 12, pp. 1411-1423

[6] W. W. Everett, "Software Component Reliability Analysis", 1995, "SoftwareReliability and Testing", Los

ISSN: 2277-9655
Impact Factor: 1.852

Alamitos, California, IEEE Computer SocietyPress, pp.45-46
[7] B. Littlewoods, “Software Reliability”, 1987, Blackwell Scientific Publications
[8] “Creating a Metrics Program”, Software Productivity Center Inc., http://spc.ca/resources/metrics/Last Date Accessed: 25.06.2013
[9] M. Şahinoğlu, “Compound-Poisson Software Reliability Model”, July 1992,IEEE Transactions on Software Engineering, Volume: 18 Issue: 7, pp. 624-630
[10] J. Voas, “Assuring Software Quality Assurance”, May-June 2003, IEEESoftware, Volume: 20 Issue: 3, pp. 48-49
[11] R. L. Glass, “Defining Quality Intuitively”, May-June 1998, IEEE Software, pp.103-107
[12] J. Viega, J. Voas, “Can Aspect-Oriented Programming Lead to More Reliable Software?” November-December 2000, IEEE Software, pp. 19-21
[13] J. R. de Almeida Jr., J. B. Camargo Jr., B. A. Basseto, S. M. Paz, “BestPractices in Code Inspection for Safety-Critical Software”, May-June 2003,IEEE Software, Volume: 20 Issue: 3, pp. 56-63
[14] R. Laddad, “Aspect-Oriented Programming Will Improve Quality”, November-December 2003, IEEE Software, pp. 90, 92
[15] J. Barnard, A. Price, “Managing Code Inspection Information”, March 1994,IEEE Software, Volume: 11 Issue: 2, pp. 59 -69