Volume-8 | Issue-4 | April-2019 | PRINT ISSN No 2250-1991

ORIGINAL RESEARCH PAPER

Engineering

SOFTWARE RELIABILITY IMPROVES SOFTWARE QUALITY AND ITS MODELS: A REVIEW

KEY WORDS: failure, hardware, reliability, integration, metrics, software

Ekta Nehra Student, Dept. of CSE, OPJS University, Churu, Rajasthan,india

ABSTRACT

Reliability of a software product essentially denotes its trustworthiness or dependability. Alternatively, reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time. Hardware reliability and software reliability both are different concept; main difference in both is hardware is manufactured whereas software is developed. Software reliability has its own factors and characteristics. Reliability can be measured using reliability metrics, and failures that occur in software can be of many types. Software reliability model can be used to specify the form of a random process that describes the behavior of software failures with respect to time, and which model is best to find out which kind of failure and what future improvement can be done in software reliability.

I.INTRODUCTION

Software Reliability is defined as it is possibility of software performs failure-free operation for a definite period of time in a stated environment [1]. System reliability is affected by Software Reliability because software reliability is also an essential factor affecting it. When software is developed by software engineer it has on average 6 faults in 1000 lines [3]. Software reliability is very different from hardware reliability in that software reliability reflects the design perfection whereas hardware reliability reflects the manufacturing perfection. As the complexity of software increases it becomes major causative factor of Software Reliability problems. Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error. "It is the probability of a failure free operation of a program for a specified time in a specified environment". Serious problems can also occur in unreliable software like automatic pilot in airplane and you want an emergency break in car but it's not responding when you hit the pedal. In other words we can say that a major problem of software industry is its inability to develop bug free software. Hence, software crisis has become a fixture of everyday life. Many well published failures have had not only major economic impact but also become the cause of death of many human beings. Some failures are: Y2K problem, patriot missile, Arian-5 space rocket, The Space Shuttle etc. so by using reliability factor software engineers try to avoid such problems.

II.SOFTWARE RELIABILITY

The term reliability is often misunderstood in the software field since software does not break or wear-out in the physical sense. Once a software defect is properly fixed, it is in general fixed for all times. Failure usually occurs only when a program is exposed to an environment that it was not developed or tested for. We do not have wear out phase in software. Software may be retired only if it becomes obsolete some contributing factors are like change in environment, change in infrastructure/technology, major change in requirements, increase in complexity , extremely difficult to maintain Software, deterioration in structure of the code, slow execution speed, poor graphical user interface. The expected curve for software is given below:

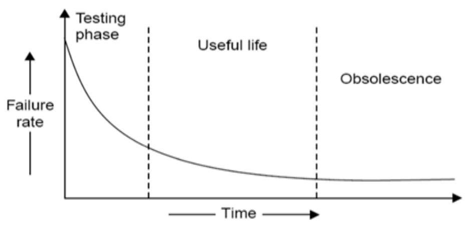

Fig.1. Change in failure rate of a software product

III.REASONS FOR SOFTWARE RELIABILITY BEING DIFFICULT TO MEASURE

The reasons why software reliability is difficult to measure can be summarized as follows:

1. The reliability improvement due to fixing a single bug depends on where the bug is located in the code.
2. The perceived reliability of a software product is highly observer dependent.
3. The reliability of a product keeps changing as errors are detected and fixed.

V.RELIABILITY METRICS

The reliability requirements for different categories of software products may be different. For this reason, it is necessary that the level of reliability required for a software product should be specified in the SRS (software requirements specification) document. In order to be able to do this, some metrics are needed to quantitatively express the reliability of a software product [5].

1. Rate of occurrence of failure (ROCOF).

ROCOF measures the frequency of occurrence of unexpected behavior (i.e. failures). ROCOF measure of a software product can be obtained by observing the behavior of a software product in operation over a specified time interval and then recording the total number of failures occurring during the interval.

2. Mean Time To Failure (MTTF).

MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures. Let the failures occur at the time instants t1, t2, …, tn. Then, MTTF can be calculated as $\sum_{i=1}^{n}\left(\frac{ti+1-ti}{(n-1)}\right)$ It is important to note that only run time is considered in the time measurements, i.e. the time for which the system is down to fix the error, the boot time, etc are not taken into account in the time measurements and the clock is stopped at these times.

3. Mean Time To Repair (MTTR).

Once failure occurs, sometime is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.

4. Mean Time Between Failure (MTBR).

MTTF and MTTR can be combined to get the MTBR metric: MTBF = MTTF + MTTR. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours. In this case, time measurements are real time and not the execution time as in MTTF.

5. Probability of Failure on Demand (POFOD).

Unlike the other metrics discussed, this metric does not explicitly involve time measurements. POFOD measures the likelihood of the system failing when a service request is made. For example, a POFOD of 0.001 would mean that 1 out of every 1000 service requests would result in a failure.

6. Availability.

Availability of a system is a measure of how likely shall the system be available for use over a given period of time. This metric not only considers the number of failures occurring during a time interval, but also takes into account the repair time (down time) of a system when a failure occurs. This metric is important for systems such as telecommunication systems, and operating systems, which are supposed to be never down and where repair and restart time, are significant and loss of service during that time is important.

.V.CHARACTERISTICS OF SOFTWARE RELIABILITY

1. **Reliability prediction:** software reliability cannot be predicted from any physical basis, since it depends completely on human factors in design.
2. **Wear out:** Software does not have wear-out phase.
3. **Redundancy:** cannot improve software reliability if identical software components are used.
4. **Time dependency and life cycle:** software reliability is not a function of operational time.
5. **Environmental factors:** do not affect software reliability; expect it might affect program inputs.
6. **Built with standard components:** Well-understood and extensively-tested standard parts will help improve maintainability and reliability.

VI. CLASSIFICATION OF SOFTWARE FAILURES

A possible classification of failures of software products into five different types is as follows:

1. **Transient**. Transient failures occur only for certain input values while invoking a function of the system.
2. **Permanent.** Permanent failures occur for all input values while invoking a function of the system.
3. **Recoverable.** When recoverable failures occur, the system recovers with or without operator intervention.
4. **Unrecoverable.** In unrecoverable failures, the system may need to be restarted.
5. **Cosmetic.** These classes of failures cause only minor irritations, and do not lead to incorrect results.

VII. SOFTWARE RELIABILITY MODELS

To model software reliability one must first consider the principal factors that affect it: fault introduction, fault removal, and the environment. Fault introduction depends primarily on the characteristics of the developed code (code created or modified for application) and development process characteristics, which include software engineering technologies and tools used and level of experience of personnel. Note that code can be developed to add features or remove faults. Fault removal depends upon time, operational profile. The models are distinguished from each other in general terms by the nature of the variation of the random process with time. a software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time. Software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability[6]. Over 200 models have been developed since the early 1970s, but how to quantify software reliability still remains largely unresolved. There is no single model that can be used in all situations. No model is complete or even representative. Some of them are explained below:

VII.1.Basic Execution Time Model

This model was developed by J.D. MUSA in 1979 and is based on execution time.it is assumed that failures may occur to a non-homogeneous poisson process. Examples of poisson processes are: expected number pf road accidents in a given period of time.in this model, the decrease in failure intensity, as a function of the number of failures observed, is constant and given as: $\lambda\mu= \lambda_0(1-\mu/V_0)$ where λ_0: initial failure intensity at start of execution.

V_0: number of failures experienced, if program is executed for infinite time period.

μ: average or expected number of failures experienced at a given point in time.

This model implies a uniform operational profile. If all input classes are selected equally often, the various faults have an equal probability of manifesting themselves. The correction of any of those faults then contributes an equal decrease in the failure intensity. The negative sign shows that there is a negative slope meaning thereby a decrementing trend in failure intensity.

VII.2.Logarithmic Poisson Execution Time Model

This model is also developed by musa. The failure intensity function is different here as compared to basic model. In this case, failure intensity function (decrement per failure) decreases exponentially whereas it is constant for basic model.

The failure intensity function is given as: $\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$

Where θ is called the failure intensity decay parameter. The relationship between failure intensity and mean failures experienced (μ) is shown:

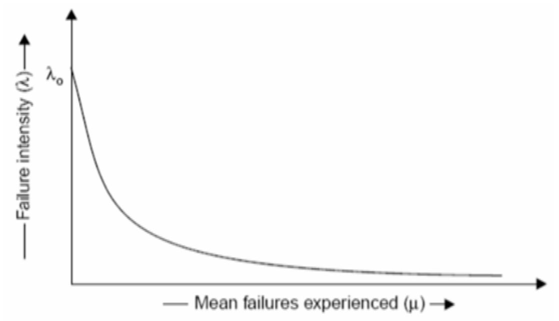

Fig.2. relationship between λand μ

VII.3.Calendar Time Component

The calendar time component relates execution time and calendar time by determining the calendar time to execution time ratio at any given point in time. The ratio is based on the constraints that are involved in applying resources to a project. In test, the rate of testing at any time is constrained by the failure identification or test team personnel, the failure correction or debugging personnel, or the computer time available. The following is common scenario, at the start of testing one identifies a large number of failures separated by short time intervals. Testing must be stopped from time to time to let the people who are fixing the faults keep up with the load. As testing progresses, the intervals between failure s become longer and longer. Finally, at even longer intervals, the capacity of the computing facilities becomes limited.

The calendar time component is based on a debugging process model. This model takes into account:

1. Resources used in operating the program for a given execution time and processing an associated quantity of failure. 2. Resources quantities available, and
3. The degree to which a resource can be utilized (due to bottlenecks) during the period in which it is limiting.

VII.4.Capability Maturity Model

It is a strategy for improving the software process, irrespective of the actual life cycle model used.

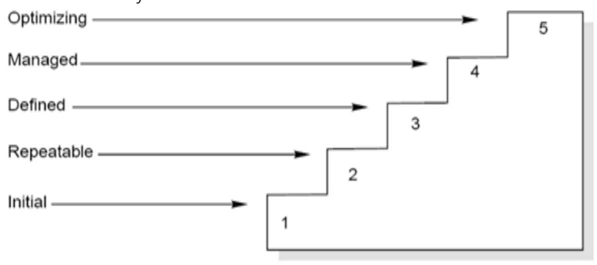

Fig.3. maturity levels of CMM

Maturity Levels:

- **Initial (Maturity Level 1)**
- **Repeatable (Maturity Level 2)**
- **Defined (Maturity Level 3)**
- **Managed (Maturity Level 4)**
- **Optimizing (Maturity Level 5)**

Key Process Areas of level 2
- Requirements management
- Software project planning
- Software project tracking and oversight
- Software subcontract management
- Software quality assurance
- Software configuration management

Key process area of level 3
- Organization process focus
- Organization process definition
- Training program
- Integrated software management
- Software product engineering
- Inter group coordination
- Peer reviews

Key process area of level 4
- Quantitative process management
- Software quality management

Key process area OF level 5
- Defect prevention
- Technology change management
- Process change management

VII.5.Reliability growth models

A reliability growth model is a mathematical model of how software reliability improves as errors are detected and repaired [7]. A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained. Thus, reliability growth modeling can be used to determine when to stop testing to attain a given reliability level. Although several different reliability growth models have been proposed, in this text we will discuss only two very simple reliability growth models.

VII.5.1.Jelinski and Moranda Model .

The simplest reliability growth model is a step function model where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. Such a model is shown in fig. 4. However, this simple model of reliability which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since it is already known that correction of different types of errors contribute differently to reliability growth.

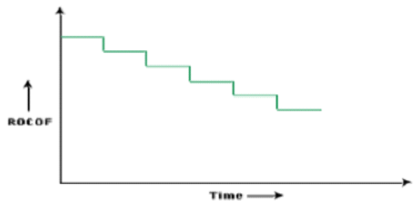

Fig.4. Step function model of reliability growth

VII.5.2.Littlewood and Verall's Model

This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases (Fig. 5). It treat's an error's contribution to reliability improvement to be an independent random variable having Gamma distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as test continues.

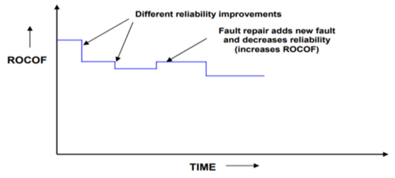

Fig.5.Random-step function model of reliability growth

VIII. CONCLUSION AND FUTURE WORK

Reliability of a software product essentially denotes its trustworthiness or dependability. Alternatively, reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time. It is obvious that a software product having a large number of defects is unreliable. It is also clear that the reliability of a system improves, if the number of defects in it is reduced.. It is clear that the quantity by which the overall reliability of a program improves due to the correction of a single error depends on how frequently the corresponding instruction is executed. Thus, reliability of a product depends not only on the number of latent errors but also on the exact location of the errors. Apart from this, reliability also depends upon how the product is used, i.e. on its execution profile. If it is selected input data to the system such that only the "correctly" implemented functions are executed, none of the errors will be exposed and the perceived reliability of the product will be high. On the other hand, if the input data is selected such that only those functions which contain errors are invoked, the perceived reliability of the system will be very low. Software reliability meaningful results can be obtained by applying suitable models to the problem, so in future improved models will be developed to overcome software reliability problems.

REFERENCES

[1.] GB/T 11457-95 Software Engineering Terms.
[2.] Yichen Wang. Test and Fault Diagnosis of NCS Software. Plant Maintenance Engineering 2005,2:36-37.
[3.] Minyan Lu. Software Reliability Engineering. National Defense Industry Press, 2011.
[4.] Z. Jelinski, P. Moranda. Software Reliability Research: Statistical Computer Performance Evaluation. N.Y. and London: Academic Press, 1972: 465-484.
[5.] Kaiyuan Cai. Software Reliability: A Personal View.Systems Engineering and Electronics, 1993, (4):47-54.
[6.] Musa, J.D., Iannino, A. and Okumoto, K. (1987). Software Reliability: Measurement, Prediction, Application, McGraw-Hill, Inc., New York, NY.
[7.] Matsumoto, K.I., Inoue, K., Kikuno, T. and Torii, K. (1988). Experimental evaluation of software reliability growth models, 11th International Symposium FTCS-18, Tokyo, Japan, pp. 148–153