International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Ultra-High Reliability for Non-Mission-Critical Applications

Renu Garg[1], Dr. Amit Gupta[2]

[1]Research Scholar, Mewar University,Chittorgarh, Rajasthan, India

[2]Maharaja Agersen Institute of Technology, Guru Gobind Singh Indraprastha University, Rohini, Delhi , India

Abstract: ***A desire to achieve Ultra high reliability of a system may demand more investment in terms of time and cost. Which means higher time and higher cost may diminish the expected returns. In other words, increasing the reliability may increase the efficiency of the software, but it does not always ensure the achievement of commercial objective of the organisation. A system is reliable if it is used according to its specific parameters. For non-mission-critical applications achieving ultra-high reliability is advantageous but not obligatory rather it may lead to diminish the returns. In today's competitive environment, delay in product release may lead to opportunity loss and hence revenue loss. If system is tested time and again to make it failure free, it may offer a competitive advantage to other company. However, in case of mission-critical applications achieving ultra-high reliability is imperative. Such applications are expected to deliver high level of security and accuracy because low reliability may lead to unbearable losses. In this scenario, investment of time and cost is acceptable & justified.***

Keywords: Ultra-high reliability, mission-critical applications, Software Quality, reliability prediction

1. Introduction

Software reliability can be defined as the probability that no failure occurs up to time t. Software Reliability is hard to achieve, because the complexity of software tends to be high. Thus, Software has become an essential part of industry, medical systems, spacecraft and military systems, and many other commercial systems. The application of software in many systems has led software reliability to be an important research area. Researchers and engineers have worked to increase the chance that the software systems will perform satisfactorily during operation. This process required the removal of faults during the testing phase. Researchers used existing technologies in order to improve the software reliability significantly by avoiding the occurrence of faults in the design and development of software programs. A failure is the departure of software behavior from the user requirements. This phenomenon must be distinguished from the fault (bug) in the software code which causes the occurrence of failure as soon as it is activated during program execution. When a failure has been experienced, the underlying fault is detected and fixed correctly, then the reliability of software will improve with time.

2. Software Reliability Prediction

After fitting a model describing the failure process we can estimate its parameters, and the quantities such as the total number of faults in the code, future failure intensity and

Metrics are used to predict a variety of measures including the initial failure rate , final failure rate, fault density per executable lines of code, fault profile, as well as the parameters of a software reliability growth model. The final outcomes of a software reliability prediction include:

- Relative measures for practical use and management.
- A prediction of the number of faults expected during each phase of the life cycle.
- A constant failure rate prediction at system release that can be combined with other failure rates.

The major difference between software reliability prediction and software reliability estimation is that predictions are performed based on historical data while estimations are based on collected data. Predictions, by their nature, will almost certainly be less accurate than estimations. However, they are useful for improving the software reliability during the development process. If the organization waits until collected data is available (normally during testing), it will generally be too late to make substantial improvements in software reliability. The predictions should be performed iteratively during each phase of the life cycle and as collected data becomes available the predictions should be refined to represent the software product at hand.

A software reliability prediction is performed early in the software life cycle, but the prediction provides an indication of what the expected reliability of the software will be either at the start of system test or the delivery date. It is largely based on the projected fault count at the point system test is initiated.

While hardware analysts will perform predictions to determine what improvements, if any, can be made in designing and selecting parts, the software analysts will perform predictions to determine what improvements, if any, can be made to the software development techniques employed and the rigor with which the process is carried out. The techniques can be on a global level, such as organization procedures, or they can be on a local level such as the complexity of each software unit. The software analyst, like the hardware analyst, must be involved in the software engineering day-to-day activities to be able to measure the software reliability parameters and to be able to understand what improvements can be made.

One important benefit from performing predictions is to correlate the software methods and techniques employed to the actual failure rate later experienced. This comparison can lead to improved software methods and techniques, particularly testing techniques.

There are certain parameters in this prediction model that have tradeoff capability. This means that there is a large difference between the maximum and minimum predicted values for that particular factor. Performing a tradeoff means that the analyst determines where some changes can be made in the software engineering process or product to experience an improved fault density prediction. A tradeoff is valuable only if the analyst has the capability to impact the software development process.

The tradeoff analysis can also be used to perform a cost analysis. For example, a prediction can be performed using a baseline set of development parameters. Then the prediction can be performed again using an aggressive set of development parameters. The difference in the fault density can be measured to determine the payoff in terms of fault density that can be achieved by optimizing the development. A cost analysis can also be performed by multiplying the difference in expected total number of defects by either a relative or fixed cost parameter.

The output of this model is a fault density in terms of faults per KSLOC. This can be used to compute the total estimated number of inherent defects by simply multiplying by the total predicted number of KSLOC. If function points are being used, they can be converted to KSLOC. Fault density can also be converted to failure rate by using one of the following:

1) Collected test data,
2) Historical data from other projects in your organization, and/or
3) The transformation table supplied with the model.

Ideally, the developing organization should determine a conversion rate between fault density and failure rate. If that data is not available then this technique supplies a conversion ratio table. The predicted fault density output from this model can also be used as an input to the Musa prediction model.

The values of many of the parameters in this model may change as development proceeds. The latest updated values should always be used when making a prediction. The predictions will tend to become more and more accurate as the metrics from each successive phase become available and as the values are updated to more closely reflect the characteristics of the final design and implementation. The details of this model are not contained in this notebook.

3. Ultra High Reliability Prediction

It is essential to consider achievability and testability when predicting reliability for software systems that must be relatively high. Demands for perfection should be avoided as they are not testable or demonstrable. For example, if the demand for the failure rate is 10-4 then there must be sufficient resources for extensive validation and verification to demonstrate this level. The current state of the art is limited in providing any help in assessing the software reliability at this level. Techniques such as Formal Methods are currently being used by software organizations developing ultra high reliability systems.

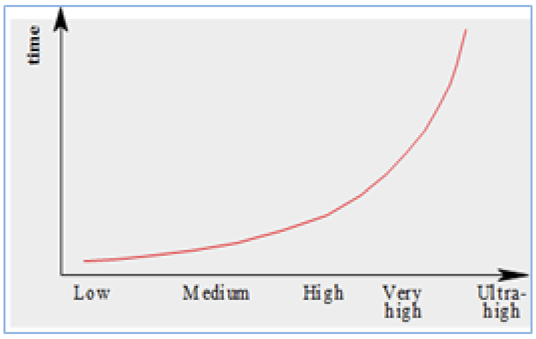
Optimum Release Time

There are methods available for predicting the optimal release time. Musa model is based on software reliability growth. Process Productivity Parameter was developed by Quantitative Software Management, Inc. It can predict optimal release time based on current productivity, effort and size of product. COCOMO model was developed by Barry Boehm. It is based on size, schedule time and effort as well as some product and development characteristics. The Musa software reliability growth model can be used to determine the optimum release time for minimizing overall cost.

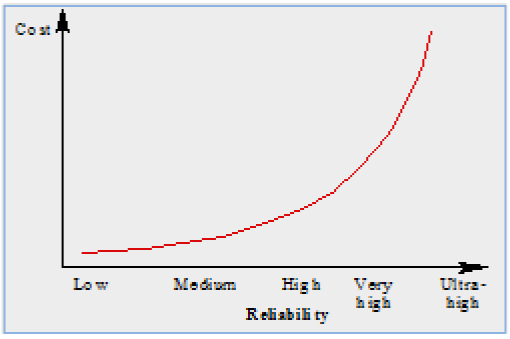
4. Limitations to Achieve High Reliability

The first limitation comes from the fact that we want to achieve higher reliability but we have limited time and cost as well. If a software is tested time and again to make it failure free it may happen that some competitive company releases its software therefore revenue loss.

An inappropriate increase of the reliability of the system may lead to a simultaneous increase of time therefore increase the cost. In other words, increasing the reliability of a system does not always mean increasing the efficiency of the software.



Time to increase Reliability



Cost to Increase Reliability

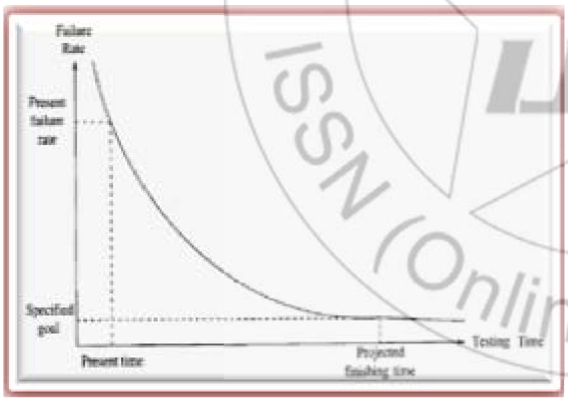
5. Conclusion

The objective of estimating delivery time is to either minimize the overall costs (or downtime, etc.) or meet a reliability objective. The ultimate goal is – To decide when to stop testing by observing the minimal risk for non-mission-critical applications.

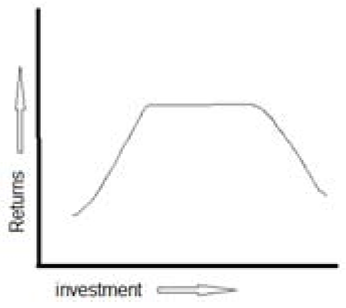
Behavior of software failure is very complex owing to the debugging process, the sequence of input and the operational environment. To recollect the success, the projects where this has been used for predicting out going quality and has matched to the prediction. A system is reliable if it is used according to its specific parameters. If the access is forced beyond limits by the employee, the system can become unreliable. As per Industry perspective, various parameters can be added like Level, Time and stage of failure. Customer satisfaction is the direct measure of software reliability.

Many software reliability growth models have been analyzed for measuring the growth of software reliability and it has been concluded that a particular model is suitable for particular industry. Initially industries recognize the model that is appropriate for them. Once this phase is completed, it is very easy to estimate the delivery time.

Simulation of various models enables evaluation of the system without actually modifying organizational structure and procedures, with simulation; it is made possible to analyze the earliest delivery date of a non-mission-critical-application



In Industry, as we surveyed some mobile companies, it has been concluded that level of customer satisfaction varies from company to company and cost of the product. If a customer purchases some costly product, his expectations will be high. Moreover, customer's expectation (satisfaction criteria) differs with branded and non-branded product.



Achieving ultra-high reliability for software may lead to increase cost (investment) and time. After a critical point returns may be diminished.

References

[1] Loan Pham , Hoang Pham . Software Reliability Models with Time-Dependent Hazard Function Based on Bayesian Approach. IEEE Transactions on Systems , Man and Cybernatics- Part A: Systems and Humans, Vol. 30, No. 1, Jan 2000
[2] Mettas, A. and Zhao, W. Modeling and Analysis of Complex Repairable Systems, Technique Report, ReliaSoft Corporation, 2004.
[3] H. Roberts. Predicting the Performance of Software Systems via the Power Law Process. Ph.D. thesis, University of South Florida, Tampa, FL, 2000.
[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA: Addison-Wesley, 2005.
[5] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability. McGraw-Hill,New York, 1987
[6] Kaminskiy, M. and Krivtsov, V. "A Monte Carlo approach to repairable system reliability analysis." Probabilistic Safety Assessment and Management, New York: Springer; p. 1063-1068, 1998.
[7] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability. McGraw-Hill,New York, 1987
[8] Kaminskiy, M. and Krivtsov, V. "A Monte Carlo approach to repairable system reliability analysis." Probabilistic Safety Assessment and Management, New York: Springer; p. 1063-1068, 1998.
[9] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability. McGraw-Hill,New York, 1987
[10] Kaminskiy, M. and Krivtsov, V. "A Monte Carlo approach to repairable system reliability analysis." Probabilistic Safety Assessment and Management, New York: Springer; p. 1063-1068, 1998.
[11] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability. McGraw-Hill,New York, 1987
[12] Kaminskiy, M. and Krivtsov, V. "A Monte Carlo approach to repairable system reliability analysis." Probabilistic Safety Assessment and Management, New York: Springer; p. 1063-1068, 1998.
[13] J. D. Musa, A. Iannino, and K. Okumoto. Software reliability. McGraw-Hill,New York, 1987
[14] Kaminskiy, M. and Krivtsov, V. "A Monte Carlo approach to repairable system reliability analysis." Probabilistic Safety Assessment and Management, New York: Springer; p. 1063-1068, 1998.

Author Profile

Renu Garg is a research scholar of Mewar university, Chittorgarh, Rajasthan, India. She is currently working with Vivekananda College, Delhi University. She has completed MCA from Gurukul Kangri University- Hardwar, M. Tech from DOEACC(C Level). She has 14 years academic experience. Her research interests are in improving software reliability.

Dr Amit Gupta is working as Associate Professor in Maharaja Agrasen Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University, Delhi. He obtained his Ph.D degree from University of Delhi. He has published extensively in Indian

Journals and abroad in the area of Reliability, optimization, Innovation in ICT and maintenance and software reliability. He had guided M.Phil/ Ph.D theses in Computer Science as well in Operational Research