Software Reliability Metrics

R. C. Tripathi[1*]

[1*]Associate Prof. Institute of Management Studies, IMS - Noida, UP; ramesh_c_tripathi@yahoo.co.in

Abstract

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor of Software Reliability problems. Software Reliability is not a function of time, although researchers have come up with models relating the two. The modeling technique for Software Reliability is reaching its prosperity, but before using the technique, we must carefully select the appropriate model that can best suit our case. Measurement in software is still in its infancy. No good quantitative methods have been developed to represent Software Reliability without excessive limitations. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability.

Keywords: software components, component based software engineering, component-based development, Interaction Constraints (ICs)

1. Definitions

According to American National Standard Institute, (ANSI) Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment. Although Software Reliability is defined as a probabilistic function, and comes with the notion of time, we must note that, different from traditional Hardware Reliability, Software Reliability is not a direct function of time. Electronic and mechanical parts may become "old" and wear out with time and usage, but software will not rust or wear-out during its life cycle. Software will not change over time unless intentionally changed or upgraded.

Software Reliability is an important to attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation (Wu, Zhong & Zhu, 2010). Software Reliability is hard to achieve, because the complexity of software tends to be high. While any system with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software. For example, large next-generation aircraft will have over one million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming international space station will have over two million lines on-board and over ten million lines of ground support software; several major life-critical defense systems will have over five million source lines of software. While the complexity of software is inversely related to software reliability, it is directly related to other important factors in software quality, especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to software.

"Using these definitions, 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

There has been extensive work in measuring reliability using mean time between failure and mean time to failure. These activities address the first and third aspects of reliability, identifying and removing faults so that the software works as expected with the specified reliability.

** Address for correspondence:*

R. C. Tripathi

Associate Prof. Institute of Management Studies, IMS - Noida, UP.

ramesh_c_tripathi@yahoo.co.in

2. Software Life Cycle for Reliability

Software reliability, however, does not show the same characteristics similar as hardware. A possible curve is shown in figure 2 if we projected software reliability on the same axes. There are two major differences between hardware and software curves. One difference is that in the last phase, software does not have an increasing failure rate as hardware does. In this phase, software is approaching obsolescence; there are no motivations for any upgrades or changes to the software. Therefore, the failure rate will not change. The second difference is that in the useful-life phase, software will experience a drastic increase in failure rate each time an upgrade is made (Alipour & Isazadeh, 2008). The failure rate levels off gradually, partly because of the defects found and fixed after the upgrades.

The upgrades in figure 2 imply feature upgrades, not upgrades for reliability. For feature upgrades, the complexity of software is likely to be increased, since the functionality of software is enhanced. Even bug fixes may be a reason for more software failures, if the bug fix induces other defects into software. For reliability upgrades, it is possible to incur a drop in software failure rate, if the goal of the upgrade is enhancing software reliability, such as a redesign or reimplementation of some modules using better engineering approaches, such as clean-room method.

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:

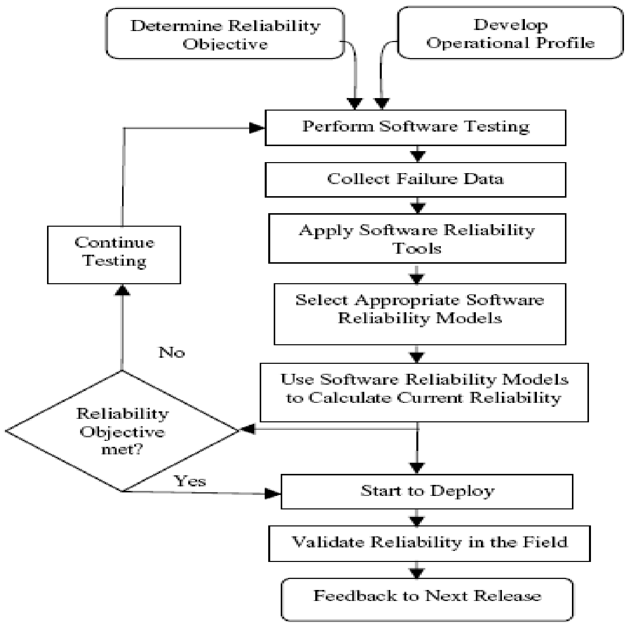
Determine Reliability Objective
Develop Operational Profile
Perform Software Testing
Collect Failure Data
Apply Software Reliability Tools
Select Appropriate Software Reliability Models
Use Software Reliability Models to Calculate Current Reliability
Continue Testing
No
Reliability Objective met?
Yes
Start to Deploy
Validate Reliability in the Field
Feedback to Next Release

Figure 1: Software reliability engineering process overview

- Start with the requirements, ensuring the product developed is the one specified, that all requirements clearly and accurately specify the final product functionality
- Ensure the code can easily support sustaining engineering without infusing additional errors
- A comprehensive test program that verifies all functionality stated in the requirements is included

3. Software Reliability Models and Measurement

As a major task of fault/failure forecasting, software reliability modeling has attracted much research attention in estimation (measuring the current state) as well as prediction (assessing the future state) of the reliability of a software system. A software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time. There are three main reliability modeling approaches: the error seeding and tagging approach, the data domain approach, and the time domain approach, which is considered to be the most popular one (Kumar & Misra, 2008). The basic principle of time domain software reliability modeling is to perform curve fitting of observed time-based failure data by a pre-specified model formula, such that the model can be parameterized with statistical techniques (such as the Least Square or Maximum Likelihood methods). The model can then provide estimation of existing reliability or prediction of future reliability by extrapolation techniques. Software reliability models usually make a number of common assumptions, as follows:

1. The operation environment where the reliability is to be measured is the same as the testing environment in which the reliability model has been parameterized.
2. Once a failure occurs, the fault which causes the failure is immediately removed.
3. The fault removal process will not introduce new faults.
4. The number of faults inherent in the software and the way these faults manifest themselves to cause failures follow, at least in a statistical sense, certain mathematical formulae. Since the number of faults (as well as the failure rate) of the software system reduces when the testing progresses, resulting in growth of reliability, these models are often called Software Reliability Growth Models (SRGMs).

It can be seen from figure 1 that there are four major components in this SRE process, namely

1. Reliability objective,
2. Operational profile,

3. Reliability modeling and measurement, and
4. Reliability validation.

A reliability objective is the specification of the reliability goal of a product from the customer viewpoint. If a reliability objective has been specified by the customer, that reliability objective should be used. Otherwise, we can select the reliability measure which is the most intuitive and easily understood, and then determine the customer's "tolerance threshold" for system failures in terms of this reliability measure (Alvaro, de Almeida, & de Lemos Meira, 2005). The operational profile is a set of disjoint alternatives of system operational scenarios and their associated probabilities of occurrence.

The construction of an operational profile encourages testers to select test cases according to the system's likely operational usage, which contributes to more accurate estimation of software reliability in the field. Reliability modeling is an essential element of the reliability estimation process. It determines whether a product meets its reliability objective and is ready for release. One or more reliability models are employed to calculate, from failure data collected during system testing, various estimates of a product's reliability as a function of test time. Several interdependent estimates can be obtained to make equivalent statements about a product's reliability. These reliability estimates can provide the following information, which is useful for product quality management: (1) The reliability of the product at the end of system testing. (2) The amount of (additional) test time required to reach the product's reliability objective. (3) The reliability growth as a result of testing (e.g., the ratio of the value of the failure intensity at the start of testing to the value at the end of testing). (4) The predicted reliability beyond the system testing, such as the product's reliability in the field. Despite the existence of a large number of models, the problem of model selection and application is manageable, as there are guidelines and statistical methods for selecting an appropriate model for each application. Furthermore, experience has shown that it is sufficient to consider only a dozen models, particularly when they are already implemented in software tools.

Using these statistical methods, "best" estimates of reliability are obtained during testing (Alvaro, de Almeida, & de Lemos Meira, 2005). These estimates are then used to project the reliability during field operation in order to determine whether the reliability objective has been met. This procedure is an iterative process, since more testing will be needed if the objective is not met.

A test compression factor is defined as the ratio of execution time required in the operational phase to execution time required in the test phase to cover the input space of the program. Since testers during testing are quickly searching through the input space for both normal and difficult execution conditions, while users during operation only execute the software with a regular pace, this factor represents the reduction of failure rate (or increase in reliability) during operation with respect to that observed during testing.

Finally, the projected field reliability has to be validated by comparing it with the observed field reliability.

This validation not only establishes benchmarks and confidence levels of the reliability estimates, but also provides feedback to the SRE process for continuous improvement and better parameter tuning (Crnkovic, Larsson, & Chaudron, 2004). When feedback is provided, SRE process enhancement comes naturally: the model validity is established, the growth of reliability is determined, and the test compression factor is refined.

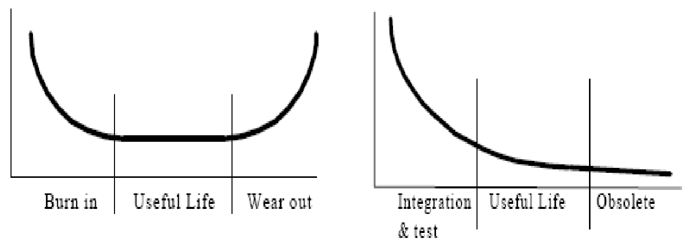

Figure 2: Failure rate

4. Software Reliability Prediction Models and Estimation

A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. As many models as there are and many more emerging, none of the models can capture a satisfying amount of the complexity of software; constraints and assumptions have to be made for the quantifying process.

Therefore, there is no single model that can be used in all situations. No model is complete or even representative. One model may work well for a set of certain software, but may be completely off track for other kinds of problems. Most software models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors (Aggarwal & Singh, 2005). The mathematical function is usually higher order exponential or logarithmic.

Software modeling techniques can be divided into two subcategories: prediction modeling and estimation modeling. Both kinds of modeling techniques are based on observing and accumulating failure data and analyzing with statistical inference. The major differences of the two models are shown in table 1.

Table 1: Difference between Software Reliability Prediction Models Estimation Models

Issues	Prediction models	Estimation models
Data reference	Uses historical data	Uses data from the current software development effort
When used in development cycle	Usually made prior to development or test phases; can be used as early as concept phase	Usually made later in life cycle(after some data have been collected); not typically used in concept or development phases
Time frame	Predict reliability at some future time	Estimate reliability at either present or some future time

Representative prediction models include Musa's Execution Time Model, Putnam's Model and Rome Laboratory models TR-92-51 and TR-92-15, etc. Using prediction models, software reliability can be predicted early in the development phase and enhancements can be initiated to improve the reliability (Rodrigues, Roshenblum, & Sebastian, 2005). Representative estimation models include exponential distribution models, Weibull distribution model, Thompson and Chelson's model, etc. Exponential models and Weibull distribution model are usually named as classical fault count/fault rate estimation models, while Thompson and Chelson's model belong to Bayesian fault rate estimation models.

The field has matured to the point that software models can be applied in practical situations and give meaningful results and, second, that there is no one model that is best in all situations. Because of the complexity of software, any model has to have extra assumptions. Only limited factors can be put into consideration. Most software reliability models ignore the software development process and focus on the results—the observed faults and/or failures. By doing so, complexity is reduced and abstraction is achieved, however, the models tend to specialize to be applied to only a portion of the situations and a certain class of the problems. We have to carefully choose the right model that suits our specific case. Furthermore, the modeling results can not be blindly believed and applied.

5. Conclusion

Software reliability is a key part in software quality. The study of software reliability can be categorized into three parts: modeling, measurement and improvement. Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exist, but no single model can capture a necessary amount of the software characteristics. Assumptions and abstractions must be made to simplify the problem. There is no single model that is universal to all the situations.

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.

References

Aggarwal, K. K., & Singh, Y. (2005). *Software Engineering,* (2nd ed.). New Age International.

Alipour, H., & Isazadeh, A. (2008). A Software reliability assessment based on a formal requirements specification. *Conference on Human System Interactions,* 311–316.

Alvaro, A., de Almeida, E. S., & de Lemos Meira, S. R. (2005). Software component certification: A survey. *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05) IEEE 2005,* 106–113.

Crnkovic, I., Larsson, S., & Chaudron, M. (2004). Component-based development process and component lifecycle. *IEEE Transaction on Software Engineering,* 44.

Kumar, S.K., Misra, R.B. (2008). An enhanced model for early software reliability prediction using software engineering metrics. *Second International Conference on Secure System Integration and Reliability Improvement, SSIRI '08,* 177–178.

Rodrigues, G. N., Roshenblum, D. S. & Sebastian, U. (2005). Sensitivity analysis for a scenario – based Reliability prediction Model. *Proceedings ICSE 2005 Workshop on Architecting Dependable Systems,* 73–77, ACM Press: USA.

Wu, H. L., Zhong, Y., & Zhu, H. D. (2010). Construct operation model based on process dababase for software reliability prediction. *The 2nd IEEE International Conference on Information Management and Engineering (ICIME),* 190–193.