386 Journal of Electrical Engineering & Technology, Vol. 2, No. 3, pp. 386~390, 2007

Quantitative Reliability Assessment for Safety Critical System Software

Dae-Won Chung*

Abstract – At recent times, an essential issue in the replacement of the old analogue I&C to computer-based digital systems in nuclear power plants becomes the quantitative software reliability assessment. Software reliability models have been successfully applied to many industrial applications, but have the unfortunate drawback of requiring data from which one can formulate a model. Software that is developed for safety critical applications is frequently unable to produce such data for at least two reasons. First, the software is frequently one-of-a-kind, and second, it rarely fails. Safety critical software is normally expected to pass every unit test producing precious little failure data. The basic premise of the rare events approach is that well-tested software does not fail under normal routine and input signals, which means that failures must be triggered by unusual input data and computer states. The failure data found under the reasonable testing cases and testing time for these conditions should be considered for the quantitative reliability assessment. We presented the quantitative reliability assessment methodology of safety critical software for rare failure cases in this paper.

Keywords: Safety critical software, Software reliability, Software verification and validation, Rare events, Quantitative assessment of software reliability

1. Introduction

Conventional quantitative reliability assessment aims at determining an estimate for the failure probability of a system. In principle, the reliability of computer-based systems can also be seen analogous to that of overall systems. This becomes good evidence for application to the safety analysis of probabilistic safety assessment (PSA), where the failure probabilities for nuclear safety functions are needed independently on the technology by which the functions are realized. Software reliability is often defined as the probability of failure free software operation of a computer program for a specified period of time in a specified environment.

The importance of quantitative assessment of software reliability for safety critical computer-based systems such as nuclear reactors, air traffic and trains is not surprising and has become quite significant in this area due to quantitative assessment of software failure being dealt with by the reasonable probabilistic safety assessment (PSA).

Software reliability, which often demands to have a failure probability in the order of 10^{-8} to 10^{-9} is an attribute of software quality, which is a multidimensional property, but it is considered to be the key attribute since it quantifies the probability of failures.

In software reliability engineering, reliability is typically illustrated by the failure intensity, which is a measure of the frequency of system failures from the view points of the users. The relationship of software failure intensity to reliability engineering is usually represented as shown in Fig. 1.

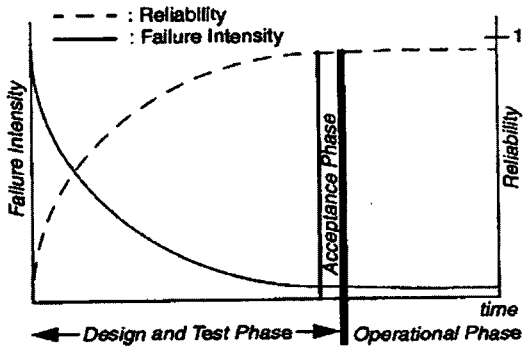

Fig. 1. Comparison of software failure rates and reliability with time

In Fig. 1, the software's failure intensity decrease and its reliability increase with time. These phenomena can occur because the software failures are the result of design faults, which arise during design and/or modification of a piece of software. Thus, when a program is first written or modified, the software defects (errors, faults, bugs) are usually introduced in the design document and/or codes.

* Department of Electrical Engineering, Honam University, 59-1 Seobong-dong, Gwangsan-gu, Gwangju-city, Korea (dwchung@honam.ac.kr)

Received 21 March, 2006 ; Accepted 23 November, 2006

During the design and test phase, software failure intensity decreases due to the discovery and removal of the defects that manifest themselves as failures, since it is assumed that the modifications to the software to remove any design defects introduces fewer design defects than are removed. As these defects are discovered and removed, the software reliability tends to grow. The final stage of the design and test phase, called the acceptance testing phase, determines if a piece of software is ready for release.

Software reliability models have been successfully applied to many commercial applications, but have the unfortunate drawback of requiring data from which one can formulate a model. Software that is developed for safety critical applications is frequently unable to produce such data, for at least two reasons. First, the software is frequently one-of-a-kind and second, it rarely fails. Safety critical software is usually expected to pass every test, producing previous little failure data.

Therefore, software that is developed for safety critical applications is difficult to analyze using conventional software reliability models.

In this paper, we addressed the quantitative assessment methodology for analyzing software reliability for the applications of nuclear safety critical systems, especially considering the rare failure cases when extensive validation testing reveals few or no failures in the feature of importance and critical points for assuring nuclear safety. Therefore, the quantitative assessment methodology of software failure is yet a large issue to be dealt with by the reasonable probability assessment analysis (PSA) of safety systems.

2. Reliability Analysis Models

The needs for demonstrating the quantitative assessment on software reliability of safety critical systems require the analysis of currently known methodologies to be able to quantify the expected failure behavior of newly developed software. In Table 1, an overview of the principal methodologies for reliability assessment is presented and individual description of each methodology is introduced hereafter.

2.1 Fault Density Model

The fundamental assumption behind fault density based prediction models is that as the number of software coding defects (faults) increases, reliability decreases without time domain. Fault (defect) density model are based on the number of coding defects per thousand lines of source codes (KSLOC), which are dependent on the characteristics of the man-power of the software development team, development tools and environment, extent of reuse, etc., from requirements to coding. Since software does not deprecate like hardware, fault density (or failure rates) may be checked and found by the effective software verification and validation activities (V&V activities) in each design phase.

Table 1. Comparison of Software Reliability Assessment

Technique	Life Cycle Phase	Typical Measure	Advantages	Limitations	Predictive Power
Fault Density	All	Faults/ KSLOC	Reference data available	Must assume encounter rate	Low
Reliability Growth	Test	Failures/execution hour	Some reference data available, objective measurement	Requires observation of multiple failures	Medium
Structured Dependability	Test & Operation	Failures/execution hour for each segment	Models software structure, objective means.	Limited reference data, requires observations	Medium /high
Rare Events	Operation	Failures/operating year	Applicable to very high integrity systems	No reference data, requires observations	Potentially high

These empirical defect data available during V&V activities will support a very accurate assessment of the fault density model without considering time domains of dynamic operation. Then, failure density function and cumulative failure distribution function will be equation (1) and (2), respectively;

$$f(t) = \frac{dQ(t)}{dt} = -\frac{dR(t)}{dt} \tag{1}$$

$$Q(t) = \int_0 f(t)dt = \frac{N_F(t)}{N_S(t) + N_F(t)} \tag{2}$$

Therefore, this model may be only applied to design documents to check and to review for source code development in the software design phases of requirements, design and code implementation.

2.2 Software Reliability Growth Models

Software reliability growth models are used to measure the trend of failure rates and/or change intervals between failures and to extrapolate them to future operation. Examples of such failures include;

① Functional failures where the actual behavior of the functions deviates from the specified functional requirements.
② Timing failures where the program may carry out correct results but where it fails to meet its timing requirements.
③ Safety failures where an accident resulting in harm or injury is deemed to be a failure of the system.

In most cases, they evaluate the reduction in failure frequency during successive development testing intervals to estimate the software reliability at the conclusion of the test or sometimes into operational deployment. Reliability growth models have been an active research theme since the early 1970s. Since then, there have been many kinds of models introduced and developed.

Most software reliability models fall into one of two broad categories. One is those which are based on modeling the times between successive failures of the software, and the other is those in which a counting process approach is used to model the number of observed failures in the software by time. Fig. 2 shows an example of such a model. The software is executed over a certain time interval, represented as t'_n, until a failure occurs.

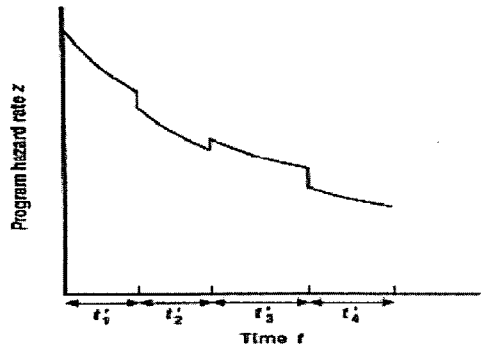

Fig. 2. Typical shape of a hazard rate for Musa-Reliability Growth Model

The time between failures is defined as a hazard rate. It is expected that overall, the hazard rate will decrease over time, but that there are discontinuities as each failure occurs. However, as the program runs for more time, there is increasing confidence in the reliability of the program. Applications of these models have all been demonstrated using real data from software failure rates of 10^{-1} to 10^{-3} per hour or in the case that at least a failure occurs every 10 to 1000 run-hours.

In spite of the fact that each of the reliability growth models has its own benefits, since no failure is allowed and a very low failure rate is often recorded for critical safety software, reliability growth models are no longer suitable for this application. If we consider that the failure rates are expected to be 10^7 to 10^9 hours (hundreds of years) of testing to demonstrate a failure rate of 10^{-7} to 10^{-9} per hour, assuming one copy of software would be tested, one and/or no failure would be observed.

Another limitation of reliability growth models is their lack of ability to model software structure. Reliability growth models treat the software as a black box and form a single expression for its reliability. Most critical systems have redundant channels for error detecting and handling, such as redundancy management and back-up tasks, in the case of nuclear safety systems having 3 redundant channels. In this case, the reliability of the whole software system cannot be simply quantified by the number of failures observed at the component levels. For example, a transient task failure may be covered by the fault tolerance provisions and may not affect critical functions. This kind of situation has been verified by several research works [2, 3], which showed that 80 to 95 percent of software failures in real-time fault-tolerant systems are recoverable by redundant processes. In such a case, reliability growth models do not provide meaningful assessment and the structured dependability models should be applied instead.

2.3 Structured Dependability Models

An alternate approach for reliability analysis is to use structured measurement, similar to the established hardware practice, such as the application of software tasks, operating system kernels or executives, and hardware components as revealed in Fig. 3. The operating times, failure rates, correlated failure probability, recovery times, and recovery probabilities of any of these components/elements can be measured during reliability tests. Reliability and availability are then estimated by models of the system structure and failure data, using measurement-based parameters for each component. Statistical estimation of reliability and availability parameters and reliability modeling based on these parameters has been a research topic in computer engineering.

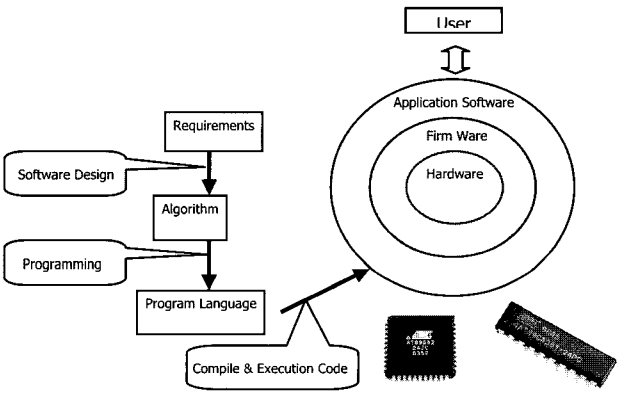

Fig. 3. Software structure of digital computer system

These analyses are based on operating experience and failure data. There are often three models that are useful in this case such as reliability block diagrams, the k-out-of-n models, and the Markov chain models. Both reliability block diagram and k-out-of-n models are combinational models and typically assume failure independence among

modeled components.

Markov chain models are stochastic models that can incorporate interactions among components and failure dependence in the model.

However, this model is very difficult to use in the case of black box testing. The current practice of measurement-based evaluation for individual software is still limited to failure rates of 10^{-2} to 10^{-5} per hour and an availability of 0.999 to 0.99999.

3. Assessment by Rare Events Technique

As previously discussed, none of the techniques described above can be furnished a credible direct assessment for failure rates lower than 10^{-6} per hour. Under favorable circumstances, the structured dependability approach may support the conclusion that such requirements are met by two or more independent versions running under a highly reliable selection or voting scheme, and this is indeed the way adopted by many exacting applications. It is an expensive solution, because in addition to the multiple software implementations it requires the development and very extensive testing of selection mechanisms. Further, in multi-channel redundant safety systems having identical software in each channel, the residual faults in the fault tolerant and communication functions of the software may be presented as a source of common cause failures (CCF) of the redundant programmable system. Therefore, there is ample motivation to investigate other assessment techniques.

The basic premise of the rare events approach is that well-tested software would tend not to fail under routine input conditions, which means that failure must be triggered by unusual input or computer states. This assumption is validated by a number of investigations that are summarized elsewhere.

In the Parnas et al. model [6], in case where no failure has been encountered during the acceptance testing phase, the probability of failure for given software artifacts are estimated as $R=(1-F/C)^N$ where, R, F, C, N is the reliability of software, number of failures, number of selected test cases and number of testing executions, respectively.

Late permits assessment of the failure probability by the likelihood of encountering the rare conditions that triggered the failure rather than by test time. As an example, consider a program that failed twice during the last 1,000 hours of testing. The first failure occurred on restart after a simulated power interruption, while at the same time one of the input signals faulted to zero (sensor fault). The second failure occurred when one out of three inputs faulted to high and another one to low. Is the failure rate of this program $2x10^{-3}$ per hour as computed from the test time? Most observers would disagree with such an assessment and will find it more reasonable to take into account the occurrence rate of the triggering events in the environment in which this program will operate. Assume that power interrupts normally occur only once a year, and sensor failures to zero are expected to occur only once every two years. The combined probability of a joint event, assuming the individual triggers to be independent, is therefore well over 10^{-7} per hour. The second test case that triggered a failure (one sensor high and one low) has an even lower probability. After the software has been modified so that it will not fail again due to these triggers, its failure probability will be much lower than that computed from the testing time.

A quantitative assessment will consider the total number of test cases that had been used and the probability of the natural occurrence of the simulated conditions. The test cases should be included in both the normal input cases of which output requires normal system behaviors and abnormal input cases of which output may be required as fail-safety behaviors. The sample space for failure probability will be {C times N} and outcomes of failure may be assumed as one (1) in the case that no failure is experienced during testing. The failures may include functional failures, timing failures and/or safety failures, and common cause failures. To illustrate the basics of the quantitative assessment, assume that during the 1,000 hours of testing time, there were chosen 10,000 test cases that simulated conditions that are expected to arise more frequently than once per 10,000,000 hours and 1,000 test cases simulating conditions that are expected to occur less frequently. Since the only failures observed were due to the second category, and since there was a ten-fold greater opportunity for failures under the first category, it can be reasoned that the failure rate in the natural environment is expected to be more than 10^{-7} per hour. The mathematical formulation of this approach is simply based on the probability of drawing black and white balls from an urn. Therefore, it is recommended and proposed to extend testing time [hours] and test cases in order to increase software reliability, which means that the quantitative reliability assessment is greatly dependent on both testing time and test cases.

4. Conclusions

As already stated above, a safety system must have a safety case, which provides a body of evidence that the system is adequately safe for a given application and

operational period of time. The safety case should include an assessment of the functionality, performance and reliability of the system. Definitely assuming that for a safety critical system software failure is permitted no longer during testing and normal operation. In this case, the conventional reliability analysis method cannot be applied. It is regarded as a rare event, the reliability assessment based on fault density and reliability growth models supports comparative evaluations but are usually not sufficiently validated to be a credible basis for stating that a software product has attained a required reliability, particularly when the required reliability is high. Structured dependability models can furnish estimates that are more precise and that also identify the elements where reliability improvement will provide the greatest benefit. They are well suited for designing and maintaining highly dependable computer systems intended for nuclear safety systems.

Except under unusually favorable circumstances, none of these methods can currently assess whether a software product meets requirements for failure rates of less than 10^{-6} per hour. The rare events approach has the potential for being useful for applications that demand the highest dependability, but it is the least validated of the methodologies discussed here Due to the constantly increasing use of software based systems in critical applications. Therefore, we can conclude that it is recommended to extend testing run time [hours] and test cases including normal and abnormal operational scenarios in order to increase software reliability, which means that the quantitative reliability assessment of safety critical software is greatly dependent on both testing time and test cases.

References

[1] M.H. Tnag, "Software Reliability Assessment-Myth and Reality", NSA Proram and Project Management, Washington Dc, 1996.

[2] R.K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability", *Technical Report*, CRHC-93-15, Center of Reliable and High Performance Computing, University of Illinois, July 1993.

[3] D. Tang and R.K Iyer, "Analysis and Modeling of Correlated Failures in Multi-computer Systems, IEEE Trans. Computers Vol. 41, No. 5, pp. 567-577, May 1992.

[4] P. J. Boland, "Challenges in Software Reliability and Testing", IEEE Trans. Computers, Vol. 46, No. 3, pp 427-432, June 1996.

[5] L.M. Kaufman, J.B. Dugan, "Using Statistics of Extremes for Software Reliability Analysis of Safety Critical Systems", IEEE Trans, Computers, Vol. 50, No. 2, pp. 355-363, Feb. 1998.

[6] Parnas, D.L, "Evaluation of Safety Critical Software," Computation of the ACM, Vol. 33, No. 6, pp. 636-648, June 1990.

[7] H. Pentti, H. Atte, "Quantitative reliability assessment in the safety case of computer-based automation system", STUK Nuclear Safety Authority, STUK-YTO-TR202, May 2004.

Dae-Won Chung

He was born in 1957 and received his B.S. degree from Busan University, Korea in 1983, and his M.S. and Ph.D. degrees in Electrical Engineering from Chungnam University, Korea in 1996 and 2000, respectively. From 1982 to 1996, he was a Senior Researcher with the Korea Atomic Energy Research Institute. Since 1997, he has been a Professor at the Department of Electrical Engineering, Honam University. His research interests are in the area of software reliabilities, mechatronics servo systems, power system control, advanced and intelligent control techniques and microprocessor systems, etc.