Research on Software Reliability of Nuclear Power Plant Instrument Control System

Ning Qiao and Jin-bing Liu(✉)

I&C Department, Nuclear and Radiation Safety Center, Beijing, China
iujingbinjob@163.com

Abstract. The reliable operation of the digital I&C system of nuclear power plants is of great significance to nuclear safety, but there is no industry-recognized method for evaluating and improving software reliability. This paper proposes a method which is depend on software reliability growth model, to calculate software reliability. At first this paper analyzes the cause and mechanism of software failure,then demonstrates how to descript software reliability mathematically. The second part of this paper explains two way to reduce software failures: fault-tolerant design and error-avoidance design. Furthermore, the third part proposes a software reliability assessment and improvement method based on the reliability growth model, which is used for the digital I&C system of nuclear power plants.

Keywords: Reliability · Software · Nuclear plant

1 Introduction

While the digital I&C technology has been developing, the traditional plant simulation technology of nuclear power plants has been gradually replaced, making the reliable operation of the digital I&C system the key to ensuring the safety of nuclear power plants and improving its reliability is of great significance. The digital I&C system is a complex system including hardware, software, and firmware. When evaluating its reliability, it is necessary to consider the influence of hardware failure and software error. Due to the inherent characteristics of aging, the trend of hardware failure rate can be predicted, and its reliability can be improved through circuit design, component selection, structural design, noise suppression, and redundancy design. However, software reliability assessment has limitations because of the system's complexity, the uncertainty of interactions with the external, and the credibility of test data. Presently, there are no global standards or methods to assess nuclear safety level software's reliability effectively. This paper first analyzes the mechanism of software failure and then introduces the software's error avoidance design and fault tolerance design method in the second part. In recent years, evaluating software reliability using the reliability growth model has received more and more attention. The third part of this paper proposes a method for reliability assessment and improvement of the core-level software based on the reliability growth model.

P. Gu et al. (Eds.): SICPNPP 2023, LNEE 1055, pp. 332–342, 2023.
https://doi.org/10.1007/978-981-99-3455-3_33

2 The Mechanism of Software Failure

2.1 Definition of Software Failure

Figure 1 defines software failure. Software failure means that the program's operation deviates from the requirements, such as crashes, wrong output results, and failure to respond within the specified time. It is the reflection of the software fault at the user end. The cause of the failure is that the software execution encounters software defects. Software defects refer to the existence of codes that can cause software failures, that is, software errors. Software defects are static and remain in the program as long as the program is not modified, such as incorrect functional requirements, missing performance requirements, etc. Software errors are mistakes made by developers during software development, including startup errors, input errors, input range errors, algorithm errors, and boundary errors. Software failure is the gradual manifestation of software defects in continuous testing and use, which is inevitable but can be improved by learning to reduce the number of errors in the program as much as possible, especially the number of significant errors.

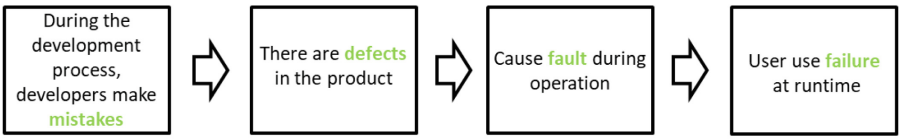

Fig. 1. Definition of software failure

There are three direct ways to reduce software failures based on the different life cycles of the software. The first is error avoidance design, which aims to reduce software defects by preventing or avoiding the occurrence or introduction of errors. The second is fault-tolerant design, which means that system functions can still be completed in the presence of errors. The third is failure detection and troubleshooting by reducing the number and severity of failures in software testing. This report mainly introduces the methods of the reliability growth model. In the following section, the paper will describe in detail how the three methods improve software reliability.

2.2 Mathematical Description of Software Reliability

Software reliability is defined as the ability of software not to cause system failure under specified conditions and within a specified period. It can be described intuitively with mathematics, and the program is regarded as the mapping from the input space to the output space. The software running error can be seen as some inputs do not get the expected output mapping. Assuming that there are I points in the input space, whether the program runs correctly has the following relationship expressed as Eq. 1,

$$Y(i) = \begin{cases} 1, & \textit{the program runs correctly} \\ 0, & \textit{else} \end{cases} \tag{1}$$

Let P(i) be the probability that the input point i occurs, the probability that output will cause the program to run correctly can be written as

$$\sum_{i=1}^{\mathrm{I}} \mathrm{P}(i)\mathrm{Y}(i) \tag{2}$$

Assuming that the expected time of each execution of the program is T, then the number of times the program runs in [0, t] time is t/T times, and the probability of non-failure in a given environment and a given time can be expressed as follows,

$$\mathrm{R(t)} = \left[\sum_{i=1}^{I} \mathrm{P}(i)\mathrm{Y}(i)\right]^{\frac{\mathrm{t}}{\mathrm{T}}} \tag{3}$$

Since I is a very large number, and P(i) is difficult to determine, Eq. (3) can only essentially reflect the definition of software reliability.

Let n represent the number of times the program runs in a particular application, which is a large number, and $\mathrm{C_n}$ represents the number of correct runs, then the software reliability during (0, t) based on running performance can be expressed as,

$$R(t) = \left[\lim_{n\to\infty} \frac{c_n}{n}\right]^{\frac{t}{T}} \tag{4}$$

Let λ(t) represent the risk function, which is the probability that the program runs correctly until the program fails at time t per unit time. Suppose T represents the time elapsed between the program's start and failure, then T is a continuous random variable. Thus,

$$\lambda(t) = \lim_{\Delta t\to 0} \frac{Pr\{(t+\Delta t) \geq T > t | T \rangle t\}}{\Delta t} \tag{5}$$

So,

$$\begin{aligned} &R(t+\Delta t) = Pr\{T > (t+\Delta t)\} \\ &= \Pr(T > t) \cap \Pr(\text{non - failure in}[t, t+\Delta t]) \\ &= R(t)(1-\lambda(t)\Delta t) \end{aligned} \tag{6}$$

Then there is,

$$\frac{dR(t)}{R(t)} = -\lambda(t)dt \tag{7}$$

And

$$R(t) = e^{-\int_0^t \lambda(x)dx} \tag{8}$$

According to the definition of reliability, R(0) = 1, R(∞) = 0. As the testing progresses, the faults are continuously corrected, and λ(t) can be optimistically viewed as a decreasing function, and the reliability function R(t) is an increasing function.

3 Software Design Method

3.1 Error Avoidance Design

To eliminate software failure, one of the common solutions in the industry is to avoid or reduce defects as much as possible during the software development process, which is the starting point of software error avoidance design. Software error avoidance design is implemented through rigorous software development methods, specifications, and process, which closely integrates the software reliability and conventional design. It runs through the conventional design process, including plan formulation, requirement analysis, and the final implementation. There are significant differences in the reliability of software and hardware, which are reflected in the failure causes, failure rates, and abandonment reasons, so the design quality has special significance for software reliability. Compared with hardware, software reliability depends more on design. It is crucial to develop the reliability design of software to ensure its reliability.

NUREG/CR6101-1993 Reliability and Safety of Nuclear Power Plant Protection System Software puts forward the qualitative evaluation terms of software reliability from the perspective of the whole life cycle of software. These terms are derived from key industrial software design and production activities, which are summaries of existing and similar engineering experiences, and have been adopted by the US Electric Power Industry Special Report. NRC believes that when the reliability requirements listed in NUREG/CR6101 are implemented, it is of great significance to the reliability of the safety-level software in nuclear power plants, which can effectively avoid or reduce the introduction of software defects and improve the inherent reliability of software. Taking the "requirements stage of the safety-level software in nuclear power plants" as an example, as the starting point of software reliability design, the software requirements specification (SRS) records all the requirements of the software, which is very important to ensure the reliability attributes of the final product. In order to ensure the reliability requirements of safety software in nuclear power plants, both the requirements of regulations and standards and the experience from the design and construction of nuclear power plants need to be considered. In addition to the conventional functional description, attention should also be paid to the following aspects when making the SRS:

- whether the changes to hardware or software are documented in the SRS, and whether the functional requirements have been independently verified;
- whether each requirement is expressed unambiguously, and whether the functional requirements as a whole can be coordinated with each other;
- whether each functional requirement can be checked, or passed inspection and testing;
- whether the communication channel is clearly expressed in subroutine calls and remote procedure calls;
- whether the transmission method and medium of each message are defined;
- whether the static operation requirements are adequately described, and whether there are complete and precise timing requirements;
- whether each software requirement is testable, and whether reliability and safety requirements are independently defined;
- whether each hazard identified in the system risk analysis can be traced to one or more software requirements that would prevent, suppress or mitigate those hazards;

- whether the level of detected errors will be rated if the software is required to diagnose hardware or software errors.

3.2 Fault-Tolerant Design

As software systems are getting increasingly large and complex, there will be inevitable software defects, no matter how well-designed. The software error avoidance design discussed in Sect. 3.1 embodies the idea of prioritizing prevention. Error avoidance design applies to all types of software and should run through the entire software development process. However, the software cannot be free of defects through error avoidance design and can only achieve a certain degree of reliability. For systems with high-reliability requirements or their failure consequences may be catastrophic, using only error avoidance design cannot meet the requirements. Therefore, it is necessary to introduce the software fault-tolerant design, which allows the system to detect errors and avoid failures, or to detect and recover itself from errors. As "the other line of defense" of software error avoidance design, software fault-tolerant design plays the role of defense in depth.

Take a nuclear power plant's digital I&C protection system as an example, the system adopts the fault-tolerant design, and the Watchdog timer monitors the running status of each program. When the program failure occurs due to external disturbances or other reasons, the working step of the process exceeds its Watchdog timing value, and the Watchdog can capture this state. Then the interrupt service subroutine recognizes the process and executes a series of identification and repair steps. Specifically, firstly check whether the RAM protection zone set by the system has been rewritten and whether the collected data is within a reasonable range.

For example, the first byte of the temperature value at the outlet of the primary circuit must be within a specific range, and any other value indicates corrupted illegal data. Rereading the control output feedback value determines whether the control output is correct, and the operation processing result is compared with a set of standard values the system sets to determine whether it needs corrections. Different repair strategies can be made depending on the type and nature of the fault diagnosed. If irreparable faults occur, such as repeatedly destructions of the RAM working area, the main control room alarm will be triggered. The backup disk will be switched, and manual intervention measures such as error correction and system reset will be implemented.

4 Software Reliability Growth Model

4.1 Overview

The reliability growth model is a mathematical model based on the error or failure time data found in the software testing stage. It is a theory that uses failure data to estimate and predict the failure intensity. Essentially, it describes the relationship between software defects and software failure and reflects the increasing trend of software reliability with defect elimination. It is realized by establishing the mathematical equation of the relationship between software failure and operating profile. It can be used as an indicator to evaluate whether the reliability of the current software meets the deliverable target, predict the reliability level in actual operation in the future, and estimate the time

required for testing before delivery. It is independent of the programming language and the specific software development method used. According to the uniform specification, the same software reliability model should give the same estimates regardless of which programming language is used.

So far, the research on the software reliability model has been more than 40 years, and nearly 100 kinds of models are published nationally and internationally. There are currently no complete and systematic classification methods, and some common classification methods are as follows.

< 1 > By analyzing the role of the software reliability growth model, the preconditions for its establishment, the factors affecting the accuracy of the model, the applicable reliability assessment scope, and the advantages and limitations of the model, *IEEEStd1633-2008* divides the current reliability growth models into three classes: (1) Exponential Non-homogeneous Poisson Process (NHPP) model, (2) Non-exponential NHPP model, (3) Bayesian model.

< 2 > The random process classification method: The random process classification method is a common classification method, as shown in Fig. 2. Stochastic process models are often called macroscopic models, and others are called microscopic models. Generally speaking, the macroscopic model's benefits are much greater than those of the microscopic model. The macroscopic model measures the software performance and behavior more comprehensively and extensively than the microscopic model, and the advantage of the microscopic model is its local relative precision.

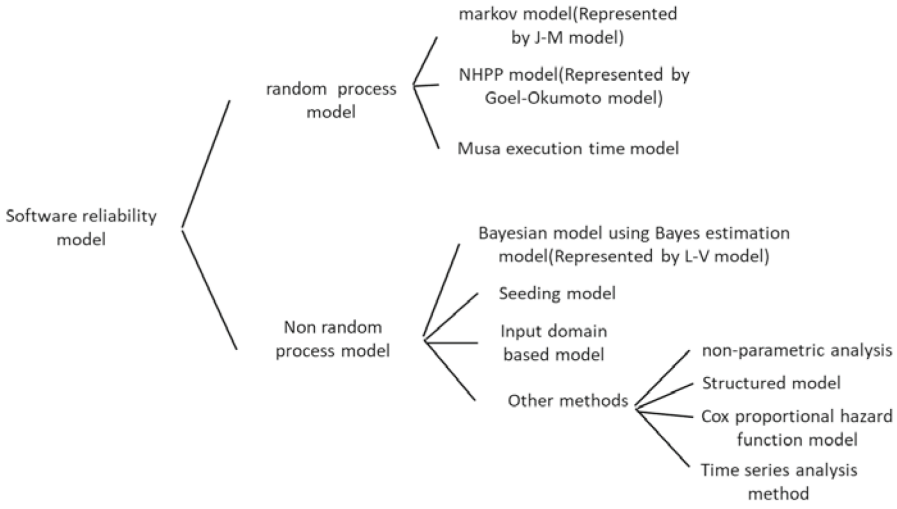

Fig. 2. Reliability models classified by random processes

< 3 > Classification by fault characteristics:

(1) Time interval between faults model: study the time interval between fault and assume it follows a specific distribution.

(2) Faults counting model: study the number of software faults in a specific time interval, and assume it follows a consistent random process. The NHPP reliability growth model belongs to this category.
(3) Faults seeding model: For a program with an unknown number of inherent faults, artificially implant a specific number of faults and then test it to estimate the program's reliability and other software performance measures based on the number of faults.
(4) Model based on the input domain: According to the usage of the program, find out the probability distribution of the input domain, randomly select test cases on the input domain, and observe the program faults to infer the program's reliability.

< 4 > Classification by data domain and time domain model (Fig. 3):

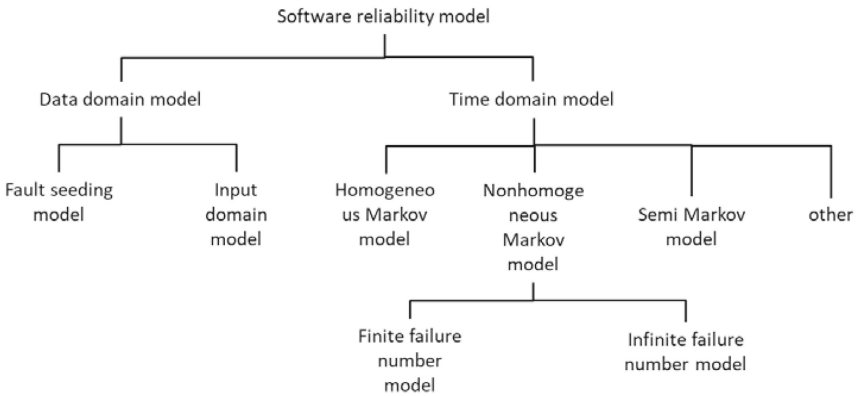

Fig. 3. Classification method of data domain and time domain model

4.2 Modeling and Application of Reliability Growth Model

The primary process of establishing a reliability growth model is shown in Fig. 4.

In most cases, obtaining a suitable model through such a process is complicated. If the fitting accuracy does not meet the requirements, repeat the process of selecting an appropriate reliability model. If the final reliability does not meet the requirements, repeat the entire process until the requirement is met.

For the various software reliability growth models, NHPP is very important and the most attractive model for managing and improving software reliability. It can quantitatively describe the reliability-related attributes in software testing, such as the number of software faults, changes in software reliability, etc. It has been verified as the most suitable and straightforward model for evaluating software reliability in engineering practice.

Take the establishment of the reliability growth model of a nuclear power plant protection system software as an example. The model is based on data recorded during the development process and focuses on the severity level of errors, dividing errors

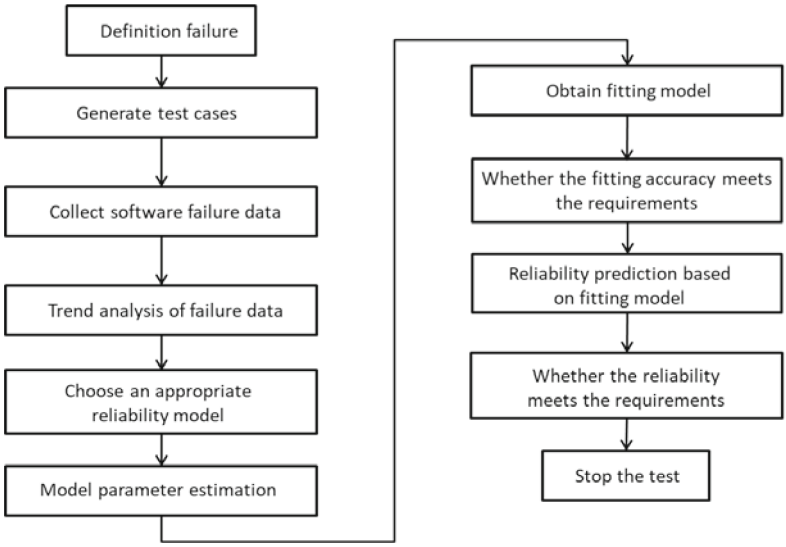

Fig. 4. Reliability model modeling process

into critical and non-critical errors. Let the critical error ratio function ρ_1 (t) and the non-critical error ratio function ρ_2 (t), as shown in Fig. 5. The error severity level ratio function can be expressed as Eq. 9 using the logistic function (a sigmoid function).

$$\begin{cases} \rho_1 t = \frac{1}{1+\beta e^{-\alpha t}} \alpha, \beta > 0 \\ \rho_2 t = \frac{\beta e^{-\alpha t}}{1+\beta e^{-\alpha t}} \alpha, \beta > 0 \end{cases} \tag{9}$$

where α and β are the parameters of the logistic function.

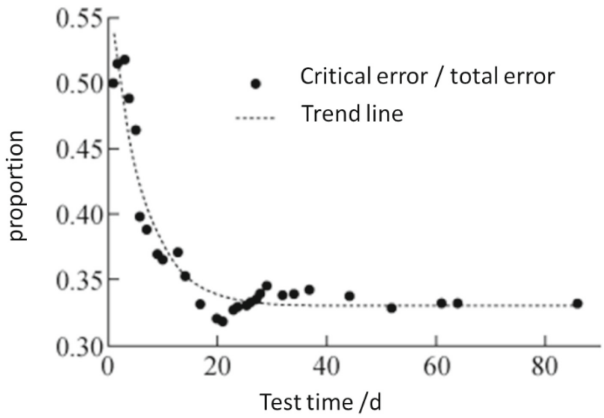

Fig. 5. Severity level of error ratio function

Make the following assumptions,

(1) Assuming that the error detection process is a Poisson process, and the cumulative number of detected errors at time t is N(t), then N(t) follows a Poisson distribution with the mean function m(t). The expected number of errors detected per unit time interval is proportional to the number of undetected errors. Therefore, the mean value function is a bounded non-decreasing time function;
(2) The errors found in software are divided into two categories according to their contribution to the failure, namely critical errors and non-critical errors;
(3) The number of errors detected in each time interval is independent of each other;
(4) Errors are corrected immediately after they are discovered, and no new errors are introduced;
(5) Errors of different severity, in the time interval of (t, Δt + t), the proportion of the expected number of detections and the number of undetected errors is different.

Then there is,

$$\begin{gathered} m_c(t+\Delta t) - m_c = b(N - m(t))\rho(t)\Delta t \\ c = 1, 2 \end{gathered} \tag{10}$$

where b is the expected detection rate given by the error detection model, and

$$\begin{cases} m(t) = \sum_{c=1}^{2} m_c(t) \\ \sum_{c=1}^{2} \rho(t) = 1 \end{cases} \tag{11}$$

when ρ(t) is fixed, the expectation function of the cumulative number of errors can be written as,

$$m_c(t) = \rho_c(t)m(t) \tag{12}$$

Based on the above assumptions, according to Eq. 10, $\Delta t \to 0$, solve the differential equation,

$$m_c(t) = N\int_0^t b\rho_c(\tau)e^{\int_0 \tau bdx}d\tau \tag{13}$$

where τ is the time integration variable, and the expected function of the cumulative number of errors with different severity is finally solved as,

$$\begin{cases} m_1(t) = Nb\int_0^t \frac{e^{b\tau}}{1+\beta e^{-\alpha\tau}}d\tau \\ m_2(t) = Nb\int_0^t \frac{e^{-\alpha\tau-b\tau}}{1+\beta e^{-\alpha\tau}}d\tau \end{cases} \tag{14}$$

According to the inhomogeneous Poisson process, for errors with different severity, the reliability is defined as

$$R_c(x|t) = e^{(-(m_c(t+x)-m_c(t)))} \tag{15}$$

The above reliability model includes four parameters N, b, α, and β. There are three commonly used parameter estimation methods,

(1) Least Squares Estimation (LSE): It has fast convergence and small deviation, and is suitable for small or medium-sized samples.

(2) Maximum Likelihood Estimation (MLE): The most widely used estimation method, suitable for large samples.
(3) Bayesian Estimation: A method of integrating prior knowledge into the estimation process, its analysis and calculation are more complicated than the LSE and MLE.

Different parameter estimation methods give different results for the same set of software failure data and may even vary greatly. Therefore in practical application, for a good software reliability model, it is better to use different parameter estimation methods and give the results after comparative analysis.

Least Squares Estimation:

$$\min S^2 = \sum_{i=1}^{k_1} \left(m_1(t_i,\theta) - n_i^1\right)^2 + \sum_{i=}^{k_2} \left(m_2(t_i,\theta) - n_i^2\right)^2 \tag{16}$$

where n_i^c is the cumulative number of errors with severity c at the end of time t_i in the testing process.

Maximum Likelihood Estimation:

$$L(\theta) = \prod_{i=1}^{k} (e^{-m_1 t_i - m_1 t_{i-1} + m_2 t_i - m_2 t_{i-1}} \frac{(m_1 t_i - m_1 t_{i-1})^{n_i^1}}{n_i^1!} \frac{(m_2 t_i - m_2 t_{i-1})^{n_i^2}}{n_i^2!}) \tag{17}$$

where $m_1 t_i + m_2 t_i$ is the overall expected cumulative number of errors.

The collected data were compared with the reliability models obtained from the LSE and MLE parameter estimation, and the results are shown in Fig. 6.

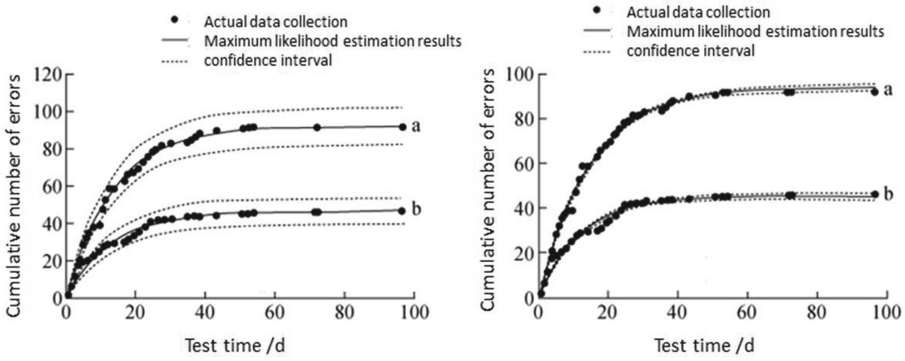

Fig. 6. Comparisons between the collected data and the reliability models

Errors of the reliability model can be calculated by comparing the average variance and the coefficient of determination between the predicted data and the collected data to evaluate the quality of the model. Another basis for evaluating the model is its predictive performance, which is not detailed here.

Select the corresponding reliability model for different software according to the modeling process given above. However, the current software reliability model may have the following limitations,

① There is a delay in the correction of faults. Fault detection and correction are two different stages in the software development process. It is ideal to assume that faults are eliminated immediately after detection, but it does not fully reflect the software testing situation and reduces the practical significance of the model.

② Software error detection and troubleshooting is a complex process, which may introduce new errors while troubleshooting due to inexperienced developers.

5 Summary

Currently, more and more software-based systems and equipment have been applied on a large scale in nuclear power plant I&C systems. Therefore, software reliability has become a critical safety issue that must be solved in designing and manufacturing digital nuclear power plants. Under the above background, this paper introduces three implementation schemes of error avoidance design, fault-tolerant design, and reliability growth model to improve software reliability in order to provide a reference for the implementation of safety-level software reliability improvement in the digital I&C system.

References

1. Li, S., Dohi, T., Hiroyuki, O.: Burr-type NHPP-based software reliability models and their applications with two type of fault count data. J. Syst. Softw. **191,** 111367 (2012). ISSN:0164-1212
2. Zhang, C.E., et al.: Review of research on software reliability growth model. J. Softw. **28**(09), 2402–2430 (2017). https://doi.org/10.13328/j.cnki.jos.005306
3. GJB/Z161-2012 military software reliability evaluation guide. National Military Standard, China (2012)
4. IEEEStd1633-2008 IEEE Recommended Practice on Software Reliability[S] .U.S.:Standards Committee of the IEEE Reliablility Society (2008)
5. IEEE Std 982.1-2005 IEEE Standard Dictionary of Measures of the Software Aspects of Dependability U.S.:Software Engineering Standards Committee (2005)
6. NUREG/CR-6901-2006 Current State of Reliability Modeling Methodologies for Digital Systems and Their Acceptance Criteria for Nuclear Power Plant Assessments. U.S.: Nuclear Regulatory Commission (2006)
7. Xian, Z., Hanyu, L.: Summary of research on quantitative reliability evaluation method of safety level DCS software. Nucl. Stand. Measure Qual. (03), 64–71 (2021)