Research on system software reliability based on service component architecture

Li WenQinq1,2*, Xu Chenguan1,2, Rao Yao1,2, Zhu Liangliang1,2, Yi Yongxian3
1 NARI Group Co., Ltd. / State Grid Electric Power Research Institute Co., Ltd, Nanjing, Jiangsu, 210000, China
2 Wuhan energy efficiency evaluation Co., Ltd., State Grid Electric Power Research Institute, Wuhan, Hubei, 430074, China
3 Marketing service Centre of State Grid Jiangsu Electric Power Co., Ltd, Nanjing, Jiangsu, 210019, China
*Corresponding author’s e-mail: liwenqing@sgepri.sgcc.com.cn

Abstract. Software reliability is an important index to evaluate system quality. Aiming at the web configuration system of service component architecture, this paper proposes a service process-oriented software reliability model, which analyses the service reliability from the whole service life cycle, and finally gives the comprehensive reliability calculation method of software. Using the method proposed in this paper, the service composition can be optimized in the system design phase, and the reliability of the whole software system can be improved.

1. Introduction
Service component architecture (SCA) is a component model architecture, which introduces the concept of service-oriented into the design of component model. The characteristics of service component are defined by service contract, which is responsible for the implementation of service contract. The system based on SCA architecture dynamically combines different service components through business requirement driven process choreography, manages and invokes services through service bus, and then realizes specific business requirements. SCA architecture realizes software reuse and encapsulation at a higher level. The software system based on SCA architecture usually belongs to the configuration open platform, and its reliability research has important theoretical and application value.

At present, there are many researches on software reliability, which can be divided into research based on reliability growth model and other research based on various mathematical models. Based on the research of reliability growth model: the definition of software reliability growth model is given in the paper [1], and three main factors affecting software reliability growth model are analyzed, including the number of faults, fault detection rate and test workload. The paper [2] focuses on the analysis and summary of the key element "fault detection rate" in the modeling process, and proposes a software reliability framework model related to fault detection rate in imperfect debugging environment. For the research of reliability growth model, many researchers choose to start with non-homogeneous Poisson process (NHPP). In the paper [3], a software reliability model is proposed by combining NHPP model with irregular change of fault detection rate. In the paper [4], a software reliability growth model based on Weibull distribution is proposed. Considering that the fault content (total number) function obeys Weibull distribution, the fitting and prediction performance of the proposed model is verified by relevant experiments. In the paper [5], the model of NHPP was used to quantify and debug the relevant environmental uncertainties, and the new framework showed the characteristics closer to the actual observation data. In the paper [6], aiming at the key problem of complex defects in software system, NHPP model is used to describe the multi-layer defect correlation effect, which can better describe the defect data set. There are many researches based on other mathematical models: The paper [7] proposes a software testing method based on logistic growth neural network, uses logistic growth curve to construct neural network model to complete fault detection, and combines exponential distribution...
correction time to complete fault correction process. In reference [8], a new cascade neural network model is proposed, which takes the output of traditional software reliability model as the input of BP neural network and transmits information step by step. In reference [9], a comprehensive evaluation method of power information system reliability based on optimal combination weight rank sum ratio is proposed. The NHPP reliability model of imperfect debugging based on problem level is given by analysing the testing characteristics of testers for minor problems and non minor problems, the debugging difficulty of developers for minor problems and non minor problems, and whether new problems will be introduced in debugging.

However, most of these researches only focus on the code quality of the software itself. In SCA architecture, the reliability of services is not only related to the implementation of service components, but also related to the registration, discovery, invocation and other stages of services. Therefore, on the basis of studying the software reliability of components, this paper analyses the reliability of the whole life cycle of service components, and proposes a service component reliability analysis model based on SCA architecture.

2. Reliability calculation of service component based on service process
Service component is the construction unit of SCA. A complete business solution is usually assembled and integrated by multiple service components. The process of service component integration usually includes four stages: service group price registration, service discovery and binding, service invocation and service execution.

2.1. Reliability analysis of service registration
Service registration is usually regarded as the encapsulation of a service component function. Web Services Description Language (WSDL) is often used to define its service interface and binding information, and web services are defined as a collection of service access points or ports. If there are errors in service registration, it will lead to service matching errors in the whole service process. The error of service registration mainly comes from the code quality problem when developers use WSDL to describe service. The probability of its occurrence is related to the quality of developers, which is an empirical value. Assuming that n services are registered with developers, and the number of services with errors is N0, the registration reliability of the service is as follows:

\[ r_{reg} = 1 - \frac{N_0}{n} \]  

(1)

2.2. Reliability analysis of service discovery phase
The service user submits the service request, and UDDI searches the service that meets the requirements in the service list according to the service request. The main reason for the system failure in this stage is the service request timeout (the failure caused by the service description error has been discussed in 2.1). In order to simplify the model, assuming that the network environment has no impact on information transmission, the service request timeout is usually caused by too many service requests to be processed by UDDI, which leads to the queuing delay exceeding the threshold. If the maximum number of UDDI service requests per unit time is, the reliability of the service discovery phase is as follows:

\[ r_{disc} = P(k \leq \tau) = \sum_{k=0}^{r} \frac{\lambda^k}{k!}e^{-\lambda} \]  

(2)

2.3. Reliability analysis of service invocation phase
The system based on SCA architecture can integrate multiple service components to provide a complete business solution. Service components can be called by other instances through service interfaces, and can also call services provided by other service components through reference interfaces. However, the calling relationship between service components will reduce the reliability of the system. If a service
component with few errors and high reliability calls other services, when the called service fails (the
component that produces the service fails), it may also affect the service component, and then cause the
service provided by the service component to fail.

If the system business contains n service components \( C = \{c_1, c_2, c_3, \ldots, c_n\} \), the importance weight
of service components is calculated as follows:

\[
\omega_i = \frac{\sum_{k=1}^{n} \phi_k q_{ik}}{\sum_{k=1}^{n} \sum_{l=1}^{n} \phi_k q_{lk}} \quad , \quad q_{ij} = 0,1
\]  

If \( q_{ij} = 0 \) the service component \( c_k \) does not call the service component \( c_i \), otherwise, \( c_i \) calls the
service component \( c_j \). Obviously, \( \omega_i \in [0,1] \) the higher the value of \( \omega_i \), the higher the relevance with
\( c_i \) and \( c_j \). The frequency of \( c_i \) call \( c_j \) is higher.

Therefore, service call reliability can be calculated by the following formula:

\[
r_{inv}(c_i) = \sum_{c_k \in \phi(c_i)} \omega_k \cdot \left(1 - r_{imp}(c_k)\right) 
\]  

where \( \phi(c_i) \) is the component set that called \( c_i \) the component; \( r_{imp}(c_k) \) is the reliability of the service
provided by the service component \( c_k \). The value of \( r_{imp}(c_k) \) depends on the code quality of the
service component \( c_k \). This article will discuss it in chapter 2.4.

2.4. Reliability analysis of service execution phase

The reliability of the service execution phase depends on the reliability of the service provider, that is,
the service component itself. Its value is related to the professional quality of the developers. The more
errors brought in during the development of the service component, the lower the reliability of the
service component. In order to improve the reliability of service components, testers usually run or test
components through different testing methods to check whether they meet the specified requirements
and find out errors.

Different from the traditional software reliability growth model based on non-homogeneous Poisson
process model (NHPP), software testing is not perfect in practice

- The total number of system errors has an initial value, which is a constant;
- The process of error detection follows NHPP, one error is detected at a time;
- There is a conditional relationship in error detection. Some errors cannot be detected before
  other specific errors are excluded;
- The total number of system errors will increase with the lapse of debugging time. The growth
  of system reliability is a process of debugging by testers. After testers detect errors, new errors
  may be introduced in the debugging work;
- The error detection rate decreases with the lapse of debugging time, and the number of errors
detected by testers per unit time will show a downward trend with the lapse of time. The more
the testing work goes to the back, the more difficult it is to find errors.

Therefore, based on the traditional software reliability growth model based on NHPP, this paper
proposes a service component reliability model based on imperfect debugging. If there are \( N_0 \) errors for
a service component in the initial state:

- \( \lambda(t) \) is the total number of errors detected by the tester up to the time \( t \);
- \( \mu(t) \) is the total number of errors introduced by the tester up to the time \( t \);
- \( \sigma(t) \in (0,1) \) is the error detection rate at the moment \( t \), which is the probability that the tester
  finds an error at the moment \( t \);
- \( \gamma(t) \in (0,1) \) is the debugging rate at the moment \( t \), which is the probability that the tester corrects
  an error at the moment \( t \);
\( \tau \in (0,1) \) is the error introduction rate, which is the probability that the tester introduce a new error with correcting an error.

In order to facilitate calculation and modeling, it is assumed that the error detection rate \( \sigma(t) \) is proportional to the number of remaining errors, and the number of detected errors \( \lambda(t) \) follows Poisson distribution by the time of arrival:

\[
\frac{d\lambda(t)}{dt} = \sigma(t) \cdot (N(t) - \psi(t)) \tag{5}
\]

\( N(t) \) is the total number of errors accumulated by the system up to the time, and \( \psi(t) \) represents the total number of errors eliminated by the tester up to the time \( t \).

The error detection rate \( \sigma(t) \) is calculated as follows:

\[
\sigma(t) = C(t) \cdot (\gamma(t) - \tau) \tag{6}
\]

\( \gamma(t) \) is error exclusion rate, \( \tau \) is error introduction rate and \( C(t) \) is test coverage rate.

The test coverage rate \( C(t) \) represents the proportion of the code that has been tested by the tester in the overall code of the component as of the time. The calculation formula is as follows:

\[
C(t) = 1 - k', \quad k \in (0,1) \tag{7}
\]

\( k \) is the difficulty coefficient of testing work, which is related to the number of testers, professional quality, total code and complexity. The larger the value of \( k \), the greater the difficulty coefficient of testing work.

In Formula 1, the formula for calculating the total number of system errors \( N(t) \) up to the time \( t \) is as follows:

\[
N(t) = N_0 + \tau \cdot \epsilon(t) \tag{8}
\]

\( \tau \) is the error introduction rate, \( \epsilon(t) \) is the total number of errors eliminated by the tester up to the time, \( N_0 \) is the initial number of errors of the component.

Because in the actual software testing work, the more errors the testers find, the more errors they exclude. Therefore, this paper assumes that the number of errors eliminated by testers is directly proportional to the number of errors detected per unit time.

\[
\frac{d\epsilon(t)}{dt} = \gamma(t) \frac{d\lambda(t)}{dt} \tag{9}
\]

Combined with formula 5 and 9, the total number of faults detected \( \lambda(t) \) can be solved.

The cumulative total number of service component errors \( N(t) \) is an independent monotonic increasing function, which also obeys Poisson distribution.

\[
P\{N(t) = n\} = \frac{(\lambda(t))^n}{n!} \exp(-\lambda(t)) \quad , n = 0,1,2... \tag{10}
\]

If \( \delta \) represents the threshold value of allowable failure for a service component up to the time \( t \), then for a service component, the reliability of the service provided by the service component \( c_i \) can be expressed as:

\[
r_{\text{up}}(c_i) = \left\{ \sum P\{N(t) = n\}, \sum n \leq \delta \right\} = \sum_n \frac{(\lambda(t))^n}{n!} \exp(-\lambda(t)) \quad , \sum n \leq \delta \tag{11}
\]

2.5. Service process oriented reliability calculation of service components

To sum up, the reliability of a service can be divided into four stages from the perspective of service process. The four stages related to the software itself include service registration, service invocation and service execution. Therefore, assuming that the service network environment satisfies the normal
transmission of service information, the reliability of service can be defined as the probability that these four stages are executed correctly.

\[ R(c_t) = r_{reg} \cdot r_{disc} \cdot r_{inv} \cdot r_{imp} \]  

(12)

3. The case study

In this paper, the client side energy consumption control platform is selected as an example to verify the method described in this paper. The customer side energy consumption control platform is a major research and development project of State Grid. The platform adopts SCA architecture to form a web-based configuration of the customer side energy consumption control platform system, which has high reusability and strong scalability. Taking a typical application scenario "energy consumption control of commercial buildings" as an example, the system mainly includes the following 14 services:

| No. | Service name         | No. | Service name     |
|-----|----------------------|-----|------------------|
| SC1 | VentilationInfo      | SC8 | EquipmentControl |
| SC2 | TemperatureInfo      | SC9 | Statistics       |
| SC3 | HumidityInfo         | SC10| OptimizationDecision |
| SC4 | FootfallInfo         | SC11| Graphical        |
| SC5 | AirqualityInfo       | SC12| ReportForm       |
| SC6 | EnergyconsInfo       | SC13| UserManagement   |
| SC7 | FluidlevelInfo       | SC14| SystemSettings   |

According to the service component call relationship, we can get the service component dependency of commercial building energy consumption control system, as shown in Figure 1.

![Figure 1 service component dependency diagram](image)

Assuming that the probability of each service being called is equal, the weight distribution of service component importance is shown in Figure 2.

As can be seen from Figure 2, service components SC11 and SC12 are much less important than other components, while service components sc13 and SC14 are the most important. Combining with figure 1, we can find that the importance of service components is mainly related to the service dependency relationship and the frequency of being called. If error detection is carried out for components with high importance to improve their reliability, it will also bring great improvement to the reliability of the whole system. At the same time, in the system development and later update, we can adjust the dependency of service components, reduce the multiple dependency of service components, and improve the reliability of the system.
4. Conclusion
For the software system based on SCA architecture, this paper gives the definition of service component reliability, analyses the service reliability from the whole service life cycle, and finally gives the comprehensive reliability calculation method of components. Using the method proposed in this paper, the service composition can be optimized in the system design stage, and the key components of the system can be focused on debugging, so as to improve the reliability of the whole software system.

Acknowledgments
This paper is supported by the following fund projects: the science and technology project of SGCC headquarters "research and demonstration application of key technologies of customer side energy control system based on smart energy service platform" (5400-202018213a-0-0-00).

References
[1] Zhang CE, Meng Fanchao, Kao Yonggui, LV Weigong, Liu Hongwei, Wan Kun, Jiang Jianan, Cui Gang, Liu Zihe. Review of software reliability growth model. (2017) Journal of Software, 28 (9): 2402-2430
[2] Zhang CE, Liu Hongwei, Bai Rui, Wang Jingyu, Wang Jinrong, LV Weigong, Meng Fanchao. Review of fault detection rate in reliability model. (2020) Journal of Software, 31 (09): 182-205
[3] Wang Jinrong, Wu Zhibo, Shu Yanjun, Zhang Zhan. Software reliability model with irregular change of fault detection rate . (2015) Journal of Software, 21 (10): 2465-2484
[4] Wang Jinrong, Zhang CE, MI Xiaoping, et al. Software reliability growth model of introducing failure in Weibull distribution. (2019) Journal of Software, 30 (6): 1759-1777
[5] Xu Jiajun, Yao Shuzhen. Quantitative research on uncertainty of software reliability growth model. (2017) Journal of Software, 28 (7): 1746-1758
[6] Yi Zelong, Wen Yumei, Lin Yanmin, et al. The influence of multilayer defect correlation effect on software reliability growth process. (2018) Computer science, 45 (2): 241-248
[7] Wei Linjing, Ning Lulu, Lian Zhichao, et al. Software testing method based on logistic growth neural network model. (2017) Journal of Harbin Engineering University, 38 (4): 646-651
[8] Li Mei. Research on software reliability prediction based on cascaded neural network combination model [J]. Automation and instrumentation, 2017, 000 (007): 14-15
[9] Xia Lu, Luo Gongyin, Liao Rongtao, et al. Comprehensive evaluation of power information system reliability based on optimal combination weight rank sum ratio. (2018) Computer applications and software, 8: 148-153