A fault alarm monitor method for safety-critical software

Yi Yu¹,a, Xiaoyong Wang²,³,b*, Cong Shi⁴,c, Pengzi Chu¹,³,d, Ye Xu²,e

¹ Maglev Transportation Engineering R&D Center, Tongji University, Shanghai, 201804, China
² CASCO Signal Co., Ltd., Shanghai, 200071, China
³ The Key Laboratory of Road and Traffic Engineering, Ministry of Education, Tongji University, Shanghai 201804, China
⁴ Telecom and Signal Branch, Shanghai Metro Maintenance Support Co., Ltd., Shanghai, 200235, China

¹email: 12164@tongji.edu.cn, bemail: wangxiaoyong@tongji.edu.cn, cemail: shicong@metro.sh.cn, demail: cpz_myhk@163.com, eemail: xuye@casco.com.cn

*Corresponding author: wangxiaoyong@tongji.edu.cn

Abstract. A design by contract (DBC) technology-based software fault alarm monitor method is discussed in this paper. Firstly, this method assigns a static module code to each software module, and assigns a static fault code to each fault alarm message in the software module. When the software is running, the dynamic module code is allocated before the software module is executed. When the software module is executed, the current module code is record, the fault alarm message is checked and collected. When a fault information is detected, all internal states of the system are recorded to the external memory, and the system is oriented to safety. At the end of the normal execution of the software module, the recorded current module code is erased. Finally, the fault analysis diagnostic tool is used to analyse the output fault message. Compared with the previous methods, the method can quickly locate the fault location, analyse the cause of the alarm, and improve the safety and maintainability of the software.

1. Introduction
The role of software-based systems is becoming more and more important in various fields of society, people’s requirements for the systems are also increasing, and the quality of a software is the key. Although there are auxiliary methods such as code review and formal verification, software testing is still the most important method to assure the quality of software [1]. The testing method and test data for software are an important part of the software testing. In order to improve the efficiency of testing, it is very important to choose a suitable method.

Since the testing and maintenance of a software are the key factors to ensure its performance, related research has attracted the attention of technicians and enterprises. This article discusses a fault alarm monitor method for a class of safety-critical software used in railway, nuclear energy, aviation, and other industries.

2. Literature Review
This section first introduces some research on conventional software testing, and then elaborates on
the concept of safety-critical software and related testing research.

In terms of conventional software testing, based on the mechanism of software failure and the cause of software faults, Shan et al. [2] proposed an integrated software fault diagnosing framework, which is composed of fault detection, fault localization, fault removal, and software release. Yuan et al. [3] proposed a method of software operation monitoring and fault recognition by using the Hook message processing mechanism of Windows system and the Structured Exception Handling mechanism of the system. The method can monitor the user's operation and identify the dynamic fault, and can generate corresponding test cases to achieve fault recurrence. For the low monitoring rate and high false alarm rate exist in current method, Cui et al. [4] suggested a method to monitor the structure fault of multi-environment programming software based on data link. Taking industrial process control as the background, Ma [5] analyzed and discussed the necessity and basic methods of software abnormal monitoring and fault recovery processing design in the development of computer real-time multi-task control machine system.

For software involving life-critical functions, the failure of the software may lead to extremely disastrous consequences, so the safety of the software is extremely concerned, and this type of software is called safety-critical software [6-10]. Safety-critical software may contain many complex logics, and manual analysis and testing are difficult to detect all hazards. In previous studies, Yu and Wu [6] took the Nelson model as an example to discuss the concept of software security and the corresponding evaluation criteria under the premise of risk equivalence. Based on the class diagram and sequence diagram for UML requirements model, Wang et al. [7] proposed a validation method of security features. Aiming at the automatic generation of test cases, one of the most critical technical issues for verifying safety-critical software, Zhang et al. [8] proposed a case-automatic-generation strategy for the safety-critical software using security coverage guidelines. Aiming at the scripting language required for safety-critical software testing, Yu et al. [9] proposed a general test scripting language SED_SCS_STL based on scenario-event-driven for the testing of safety-critical software. Aiming at the fact that the test script language and test strategy cannot meet the test requirements of safety-critical software-based systems, Yu et al. [10] proposed a safety test method based on the requirements of safety-critical software testing using a scenario-event-driven safety test strategy, and designed a security test script language based on this strategy. In order to develop safety-critical embedded software, Ge et al. [11] proposed a formal verification process based on the Systerel Smart Solver tool set. For safety-critical software with high reliability requirements, Wu et al. [12] suggested a Bayesian based software reliability demonstration test (SRDT) method. Based on the consideration of the factors that affect software reliability, Cai et al. [13] proposed an overall methodology for the assessment of software reliability, which overcame the subjectivity of separate quality assessments of the software development life cycle (SDLC) processes.

3. Method Design

3.1. Basic Approach
In order to ensure that the software can always run in the desired design space, this paper introduces the design by contract (DBC) technology [14]. DBC is a technology that can improve the security of the system. The contract consists of pre-conditions, post-conditions, and invariants. The pre-condition is checked at the beginning of each operation (i.e., function). In order to perform the operation, it must be true. The post-condition is the state of the function completion. The invariant must be true before the function is executed. It must also be true after the function is executed.

For a safety-critical software, the software needs to always run in an expected and determined state space, and the contract of each function must be satisfied at runtime. If the contract is violated, the system is outside the expected design space, and the software needs to be oriented to safety. It is a more complicated problem to record the location of the fault alarm when the software is running, because the calling relationships of function modules are very complicated. A submodule can be called by different levels of parent modules, and it can be called multiple times in the same module,
even be called thousands of times in a loop. The position of the module is dynamically determined according to the real-time status while the software is running. It is also complicated to analyze the causes of alarms. The alarms are caused by the internal state of the instantaneous system and external inputs. The source of abnormal data may not be in the alarm module, but may be the result of the combination of multiple modules. If such problems are analyzed manually, the cost is very high. In response to the above-mentioned problems, this paper proposes a software fault alarm monitor method, which leads to safe side when the software fails, and can quickly locate the fault location, analyze the cause of the alarm, and improve the safety and maintainability of the software.

3.2. Design of the Method

In view of how to monitor the fault alarm of a software in real time, and provide the internal state of the software at the moment of the fault for analyzing the problem, the process shown in Figure 1 is proposed, which is a flowchart of a functional module with the ability to monitor fault alarms.

![Flowchart of the functional module](image)

The proposed software fault alarm monitor method includes the following steps:

**Step A:** Assign a static module code to each software module. The static module code is a globally unique and non-repeated module mark. That is, in the software design stage, a static module code is assigned to each software module. The static module code is globally uniformly coded, which is only visible and used inside the module.

**Step B:** Assign a static fault code to each fault alarm message in software modules. The static fault code is a unique and non-repetitive error mark inside modules. In the software design stage, each alarm message is assigned a static fault code in modules. The static fault code is a unique and non-repetitive error mark inside modules. In the software design stage, each alarm message is assigned a static fault code in modules.

**Step C:** When the software is running, allocate the dynamic module code before the software modules are executed. The dynamic module code is that the parent module A temporarily allocates the dynamic module code to the called child module B. The dynamic module code obtained by the submodule in the same parent module is not repeated. When the submodule is called multiple times in the same parent module, each submodule will be assigned a unique and non-repeated dynamic module code. When the same submodule being called in different parent modules, only the dynamic module codes obtained by the submodules in the same parent module are not repeated. The parent module passes the dynamic module code to the submodule through formal parameters. At the same time, the input parameter interface of the module contains a parameter representing the dynamic module code.
If a submodule is called in the module, the submodule called each time will be assigned a different number, through which the sequence of execution of the submodule can be deduced inversely. The number is passed as a parameter to the submodule as its dynamic module code.

**Step D:** When the software module is executed, first record the current module code. The current module code is an ordered pair composed of a static module code and a dynamic module code. The current module code is the unique relative path that the current module executes in the parent module when the system is running. Among them, the unique relative path is represented by an ordered sequence of relative paths stored in the module information sequence table, that is, the relative path from the topmost module running in the system to the current executing module, which forms an ordered sequence in the order of execution.

Meanwhile, the set that stores the current module code is called the module information sequence table. When a current module code is recorded, the data is inserted at the end of the module information sequence table. When the module information sequence table is full, an overflow event of the module information sequence table is recorded. In the module, the static module code and the dynamic module code are combined into the current module code. The current module code represents the unique relative path executed by the module in the parent module. The first action performed by the functional module is to record the current module code in the module information sequence table. Then the module performs function-related logic and contract checking.

**Step E:** When the software module is running, check and collect fault alarm information. Specifically, when the software module is running, it is checked whether the pre-conditions, post-conditions, and invariants are all true. If the result is negative, an abnormal alarm will be generated, and the fault collection module will collect all abnormal alarm information. The fault collection module first records its own module code to the module information sequence table, and then records the fault code corresponding to the fault abnormal alarm information to the module information sequence table. When the module executes function-related logic, the module checks whether the contract (pre-condition, post-condition, invariant) is violated. If the contract is violated, an alarm message will be generated. After the contract is checked, all fault alarm events are transmitted to the fault collection module through the interface parameters. Among them, Figure 2 is the processing flow of the fault collection module.

Fig. 2 Flowchart of the fault collection module
Step F: After detecting the fault information, record all the internal states of the system to the external memory, and the system is oriented to safe side. All the internal states of the system are all global variables used in the operation of the system, including the module information sequence table data. The fault collection module also uniquely identifies the path information through the static module code and the dynamic module code. When the fault collection module detects a fault alarm, the system is oriented to the safe side. If there is no alarm, it returns to the upper module. When the system is oriented to safe side, it will interrupt the normal operation of the software, record the fault information to the module information sequence table, set the mark bit to indicate the existence of the fault information, record all the internal status of the system to the external memory, and finally call the dedicated stop module to interrupt the software operation, to make the system oriented to the safe side.

Step G: Erase the recorded current module code at the end of the normal execution of the software module. Specifically, when no abnormal alarm information occurs, the normal execution of the module ends, and the last action before the end is to erase the recorded current module code from the module information sequence table. This process is realized by deleting the last piece of data at the end of the module information sequence table. When the module information sequence table is empty, the event that the module information sequence table is empty is recorded.

Step H: Use the fault diagnosis tool to analyze undesired result. That is, after reading the module information sequence table data stored in the external memory, the unique absolute path of the abnormal alarm information when the fault occurs is judged, and the cause analysis is carried out according to the internal state of the system at this time. Among them, the unique absolute path of the abnormal alarm information when the fault occurs is indicated by the unique absolute path when the module is executed plus the fault code.

4. Conclusion
The method proposed in this paper used the relative path of the modules to locate the position where the alarm is generated, which can adapt to the complex dynamic calling relationships of a program, can increase the alarm event without limiting the capacity, and report the fault location more accurately. According to the method in this paper, the internal state of the system at the moment of the fault can be recorded to the external memory, which facilitates the use of special fault diagnosis tools to reproduce the fault scenario, quickly analyze the cause of the fault, and help the maintenance of the software.

Acknowledgments
This work was jointly supported by the Science Research Project of Shanghai Science and Technology Committee (Grant no. 18DZ1205803), the National Key R&D Program of China (Grant no. 2016YFB1200602-02), the Foundation of Shanghai Maglev and Rail Transit Cooperative Innovation Center (Grant no. 20132223), and the R&D Project of Shanghai Shentong Metro Group Co., Ltd. (JSKY19R012-WT-20018).

References
[1] Chen B., Wang P. (2006) The trend toward software of manned spaceflight engineering and its heuristics. Manned Spaceflight, 12(4): 1-4.
[2] Shan J., Xu K., Wang J. (2011) A kind of software faults diagnosing framework. Chinese Journal of Computers, 34(2): 371-382.
[3] Yuan B., Zhang W., Huangfu R., Yuan C. (2019) Method of monitoring and fault identification for software operation. Journal of Sichuan Ordnance, 40(11): 87-91.
[4] Cui J., Huang Z. (2019) Multi-environment mixed programming software structure fault layer monitoring simulation. Computer Simulation, 36(3): 317-320.
[5] Ma X. (1996) Real-time software failure detection in computer control system. Industrial Contral Computer, (5), 37-38.
[6] Yu Y., Wu M. (2001) Software safety assessment criterions with the nelson software reliability model. Computer Science, 28(11), 120-122.

[7] Wang F., Guo Y., Li B., et al. (2013) Validation method of security features in safety critical software requirements specification. Journal of Computer Applications, 33(7), 2041-2045.

[8] Zhang W., Xu Z., Yu G. (2009) Automated test data generation for safety critical software. Application Research of Computers, 26(1), 140-142.

[9] Yu G., Xu Z., Du J. (2010) Research on scenario-event-driven simulation test script language for safety-critical software system. Journal of Computer Applications, 30(2), 374-379.

[10] Yu G., Xiong J., Xu Z. (2010) General safety test method of safety-critical software. Computer Engineering, 36(17), 28-29.

[11] Ge N., Jenn E., Breton N., et al. (2018) Integrated formal verification of safety-critical software. International Journal on Software Tools for Technology Transfer, 20, 423–440.

[12] Wu Y., Li H., Yu Y. (2013) Design of safety-critical software reliability demonstration test based on Bayesian theory. In: Proceedings of 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE). Chengdu, China, IEEE, 2013, 996-1005.

[13] Cai Y., Wu Y., Zhou J., et al. (2020) Quantitative software reliability assessment methodology based on Bayesian belief networks and statistical testing for safety-critical software. Annals of Nuclear Energy, 2020, 145, Article ID 107593.

[14] Cheng H., Lu Z., Nie G., et al. (2004) Study on DBC-based software testing approach for CORBA exception mechanism. Mini-micro Systems, 25(11), 2036-2039.

[15] Zuo H., Chen X., Xia T., et al. (2020) A software fault alarm monitoring method. Shanghai: CN107145422B, 2020-08-28.