Test Case Generation Method Based on User Operation

Wei Zhang1,*, Bowen Yuan1, Zhe Li2 and Ruishuai Huangfu3

1Department of Information and Communication, Army Academy of Armored Forces, Beijing, China
2The 32064 army of the Chinese people’s Liberation Army, Zhoushan, China
3Data and Information Office of Hunan Military Region, Changsha, China

*Corresponding author email: zw2051@126.com

Abstract. After the software is published, its users are mainly operators. Although they can find problems with the software, they cannot identify and reproduce software failures. When the software fails, there must be a lot of information related to the failure in the running data of the software to be tested. Therefore, on the basis of obtaining the running data of the software under test, a test case generation method based on user operation is proposed. Based on the generation of software failure, the operation related to the failure is filtered out from the user operation. By taking these operations as the values of the system parameters to be tested, test cases that will inevitably cause software failure are obtained. Finally, the validity of the method is verified by an example, and the test cases obtained are specific to the faults of the software.

1. Introduction

The test case set is designed to adequately test parts of the system under test with as few test cases as possible. In order to achieve the goal of covering the whole system under test as much as possible, there will inevitably be many invalid test cases in the process of test case generation. Simplified test cases may result in incomplete testing. A large number of software testing researchers have been looking for a good way to achieve the balance between the two. Defects in software can occur in the form of failures. The external cause of the failure is the user's actions. Therefore, test cases are generated using user-related actions corresponding to software failures. Software failures can be reproduced. These cases are more specific and useful for later fault diagnosis.

The specific method of test case generation is introduced in the second section of this article. In the third section, a sample is designed to verify the method proposed in this article, and the results are analyzed. It is summarized in the fourth section.

2. Test Case Generation

2.1. Test Case Generation Background

In order to detect the difference between the results of the software operation and the expected results, after the test case design is completed, the test software needs to be operated manually or automatically by the system according to the test case settings. Test actions are determined by test cases. Conversely, user actions can also generate test cases. A system under test should have one or more inputs and one or more outputs during the running (or testing) process. Suppose a system under test SUT (software under testing) has ‘N’ input parameters, and these parameters are
C_1, C_2, C_3\ldots C_n$. The value range of each $C_i$ is $T_i$, which is a finite set of discrete points. 

$$(V_1', V_2', V_3',\ldots V_n') (V_1' \in T_1', V_2' \in T_2', V_3' \in T_3',\ldots V_n' \in T_n')$$

is called a test case of SUT, and it is an N-tuple vector. The specific content of the target test case can be determined by taking the user's operation according to the above parameters.

2.2. Method of Test Case Generation

2.2.1. Test case generation process. During the software running process, every operation of the user may cause the software to fail. The article [1] introduces how to use the existing technology to monitor and record the user's operation on the target software and the failure of the target software. Software exception records are also recorded in the user action documentation. Each record is sorted in the chronological order in which it occurs. Recorded documentation gives the time when software exceptions occurred and what operations caused them. During the recording process, the user's non-stop operation produces a large amount of data. Only operations that produce exceptions will generate test cases, resulting in targeted test cases.

If there is no failure record in the action record, there is no need to generate test cases. When an exception record appears in the operation record, it indicates that the software has an exception during operation. The method is shown in Figure 1 and consists of four steps: Determine the location of the failing system; Determine the parameters and range of values of the parameters; Determine the operations related to the failing system in the operation record; Generate test cases from the related operations.

![Figure 1. Test Case Generation Method.](image)

2.2.2. Determine the location of the failing system. The test objects designed in this paper are often complex and have many functional modules. Exceptions recorded by exceptions only contain the contents of exceptions, not the specific modules in which the exceptions occur. Therefore, before generating test cases, it should first determine the target system for the test cases.

When a software user operates on the software under test, the monitoring system records the user's operation step by step. Since the failure record is synchronized with the user action record and the user action is the external cause of the software failure, the position of the failure record in the recorded document reflects the user action associated with the failure. Accordingly, the function module of the previous operation record of fault record is recognized as the fault module (that is "fault system").

2.2.3. Determine the parameters and range of values of the parameters. Test cases are used to assign values to the parameters of the system under test according to the input specified to test whether the system under test meets the specified requirements. As described in the model in section 2.1., a fault system can be thought of as consisting of a number of deterministic parameters, each of which is a set of discrete values. After the fault system is determined according to the location of the fault record, the relevant characteristics of the system are analyzed reasonably to determine n parameters of the fault system and the discrete value range of each parameter. When some parameters range from a non-discrete point set, it can be divided by selecting a particular value or replaced by a special value to convert their range to a non-discrete point set.
2.2.4. Determine the operations related to the failing system in the operation record. There are many records in the user operation record, but only some records are related to the faulty system. Only a few of these operations will cause the current system failure. After determining the location of the fault system, the parameters of the fault system and their value ranges, testers can filter out the related operations of modifying these parameters in the user's operation records. The relevant operations that have been screened are divided. The dividing standard is as follows: starting from the operation of this fault system every time, and ending at the operation of leaving this fault system. In this way, all operations related to the failed system in the operation record are divided into several segments. Each segment represents a continuous and complete operation of the failed system by the user.

2.2.5. Generate test cases from the related operations. After dividing all the operations related to the fault system in the operation record, the user can get multiple continuous and complete operation segments of the fault system. Each operation segment can generate a test case. This paper considers that the previous operation section of the fault record location can cause system fault. The test cases generated by this operation section can cause system fault recurrence. Determine the change parameters corresponding to each operation in the operation section, and determine the parameter values according to the value range divided in section 2.2.3. according to the operation content. The combination of all parameter values corresponding to the operation segment is the content of the test case, and then a test case is generated.

3. The Simulation Results and Analysis

3.1. Software Instance Introduction

A representative software to be tested is designed, which has the basic menu function. On this basis, a dialog box menu item is added. Click "dialog box" to open a new window. The functions of "dialog box" window are:

(1) Enter two numbers in the input boxes "number1" and "number2". Click the "division" key, and the result of dividing the two numbers will appear in the "number3" box. In the design of this function, it should be considered that the divisor cannot be "0", and whether the divisor is "0" should be determined before calculating the result. No judgment is made here, and monitoring results of monitoring software are inspected;

(2) "Null pointer" key: another error is designed here. When clicking the "Null pointer" key, the software assigns a value to the null pointer;

(3) "Cancel" key: close this dialog box;

(4) "Shrink/Expand" key: close/open the display section below the dialog box.

3.2. Test Case Generation

First run the software to be monitored, then inject a message hook into the monitoring software to record the experimental data. The resulting operation record is shown in Figure 2 (a) and (b).

Figure 2. Operation record.
From the data recorded in the experiment, the target software produces a large number of operations, including file creation, file opening, and related functions in the Dialog menu generation dialog box. According to the test case generation method:

(1) Determine the location of the failing system

It can be seen from the operation records that there are two failure systems in the software, namely the division module and the null pointer module.

(2) Determine the parameters and range of values of the parameters

- **Function module of Division**
  
  Combining the user action record with the target software "division" function structure in Figure 3, it is considered that the function is input by three parameters: input box 1012 (Number1 input box), input box 1014 (Number2 input box), and "division" key. There are several different states for these three parameters, as shown in Table 1. There are countless input modes in input box 1012 and input box 1014. The experiment is carried out here using a special value split-in method. In order to test all cases in Table 1, $3 \times 3 \times 2 = 18$ test cases are needed to traverse all cases.

| Parameters of function module of Division | input box 1012 | input box 1014 | "Division" button |
|-------------------------------------------|---------------|---------------|-------------------|
|                                           | 0             | 0             | click             |
| Greater than 0                            | Greater than 0| Not click      |
| Less than 0                               | Less than 0   |

- **Function module of Null Pointer**
  
  Combining the user action record with the function structure of the target software "null pointer" in Figure 3, it is considered that the function is input by three parameters: input box 1012 (Number1 input box), input box 1014 (Number2 input box), and "null pointer" keys. There are several different states for these three parameters, as shown in Table 2.

| Parameters of function module of Null Pointer | input box 1012 | input box 1014 | "Null Pointer" button |
|-----------------------------------------------|---------------|---------------|-----------------------|
|                                              | 0             | 0             | click                 |
| Greater than 0                               | Greater than 0| Not click      |
| Less than 0                                  | Less than 0   |

(3) Determine the operations related to the failing system in the operation record

The record of operations related to the “division” function is shown in Figure 3.

![Operational records related to the “Division” function.](image)

The record of operations related to the “null pointer” function is shown in Figure 4.
Figure 4. Operational records related to the “Null Pointer” function.

(4) Generate test cases from the related operations
From the user's operation record, several operations related to the faulty system can be obtained. According to the operation section, several test cases corresponding to the fault system are obtained respectively. Tests that reproduce software failures are shown in tables 3 and 4.

Table 3. Software "division" function test case

| Test Case Label | 1.1                  |
|-----------------|----------------------|
| Test Case Name  | "division" function test |
| Test Case Describe | 1. First enter the dialog interface; 2. Enter "5" and "1" in the "Number1", "Number2" input box; 3. Click the Division key; 4. Enter "5" and "0" in the "Number1", "Number2" input box; 5. Click the Division key; |

Table 4. Software "null pointer" function test case

| Test Case Label | 1.2                  |
|-----------------|----------------------|
| Test Case Name  | "null pointer" function test |
| Test Case Describe | 1. First enter the dialog interface; 2. Enter "5" and "1" in the "Number1", "Number2" input box; 3. Click the Null Pointer key; |

3.3. Result Analysis
Two important test cases were obtained from user action records. It corresponds to the "division" function and the "null pointer" function test respectively, which has the following characteristics:

(1) There are five steps in test case 1.1. The division function operates twice. "5" is the dividend, "1" is the divisor and "5" is the divisor, and "0" is the divisor for calculation. Combined with the preset failures in the software under test, the target system will be tested with this test case: The target system fails when the divisor is "0".

(2) There are three steps in test case 1.2. The division function operates once. When typing "5" and "1" in the input box, click the null pointer key. According to the preset faults in the software to be tested, using this test case to test the target system will directly trigger system faults.

The two test cases can reproduce the preset faults in the software to be tested. Only by combining the fault records, the tester can accurately diagnose the software faults.
4. Conclusion

In this paper, a test case generation method based on user operation is proposed. By analyzing the structure of the fault system, the operation related to the fault can be extracted from the user operation data. It is divided into several operation segments. Several test cases are derived from the input parameters of the system under test. A test case can be generated for each operation segment. The test cases generated in the previous section of the fault record can reproduce the software failure. It is proved by an example that the test cases that can reproduce software failures can be obtained from user operation records by using the methods presented in this paper. This method improves the specificity and validity of test cases.

References

[1] Zhang W, Yuan B, Ma Z, et al. Research on Non Intrusive Methods for Dynamic Monitoring of Software[J]. IOP Conference Series Materials Science and Engineering, 2018, 466.

[2] Zhu, H., Hall, P.A.V., May, J.H.R. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 1997,29 (4), 366–427.

[3] Anand S, Burke E K, Chen T Y, et al. An Orchestrated Survey of Methodologies for Automated Software Test Case Generation[J]. Journal of Systems and Software, 2013,86(8),1978-2001

[4] Maha Kooli, Firas Kaddachi, Giorgio Di Natale, Alberto Bosio, Pascal Benoit, Lionel Torres. Computing Reliability: On the Differences between Software Testing and Software Fault Injection Techniques [J]. Microprocessors and Microsystems, 2017.

[5] M. Nosrati, H. Haghighi, M. Vahidi Asl. Using Likely Invariants for Test Data Generation[J]. The Journal of Systems &amp; Software, 2020,164.

[6] Xu Baowen, Nie Changhai, Shi Liang, Chen Huowang. A Software Failure Debugging Method Based on Combinatorial Design Approach for Testing [J]. Chinese Journal of Computers, 2006, (01),132-138.