A Software Debugging Method Based on Pairwise Testing[1]

Liang Shi[1,2], Changhai Nie[1,2], and Baowen Xu[1,2,3,4]

[1] Dept of Computer Science and Engineering, Southeast University,
210096 Nanjing, China
{Shiliang_ada, Changhainie, Bwxu}@seu.edu.cn
[2] Jiangsu Institute of Software Quality,
210096 Nanjing, China
[3] Computer School, National University of Defense Technology,
410073 Changsha, China
[4] Key Laboratory of Software Engineering, Wuhan University,
430072 Wuhan, China

Abstract. Pairwise testing is one of very practical and effective testing methods for various types of software systems. This paper proposes a novel debugging method based on pairwise testing. By analyzing the test cases and retesting with some complementary test cases, the method can narrow the factors that cause the errors in a very small range. So it can provide a very efficient and valuable guidance for the software testing and debugging.

1 Introduction

Software testing and debugging is an important but expensive part of the software development process. Much research has been aimed at reducing its cost. Among them, pairwise testing is a very important method that has been studied and applied widely for its scientificity and effectiveness in software testing with a quite small test suite, especially for the software whose faults are caused mainly by the test parameters or the interactions of a few test parameters of the system.

Pairwise testing has been used in software testing for a long time. At the beginning, through the application of design of experiments, a more efficient software testing way is obtained [4]. Much research has been done in pairwise testing. But they have been mainly aimed at reducing the size of the test suite by some methods of construction. For instance, in [1, 2] two heuristic search-based approaches are proposed, in [3] a new algebraic construction is proposed.

[1] This work was supported in part by the National Natural Science Foundation of China (60425206, 60373066, 60403016), National Grand Fundamental Research 973 Program of China (2002CB312000), and National Research Foundation for the Doctoral Program of Higher Education of China (20020286004).
Correspondence to: Baowen Xu, Department of Computer Science and Engineering, Southeast University, 210096 Nanjing, China. Email: bwxu@seu.edu.cn

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 1088 – 1091, 2005.

In this paper, we propose a debugging method based on the result of pairwise testing. The research about this has not been seen now. The convenient, reliable and effective debugging method can help people, who are working for testing or debugging the software, to find the errors and revise them quickly. It can improve the test efficiency and reduce test costs. Therefore, the research about the debugging method on the result of pairwise testing is quite important.

2 Model and Algorithm for Debugging of Pairwise Testing

Consider the system under test has n parameters c_i, where $c_i \in T_i$, T_i is a set with finite elements and the number of the elements is t_i, that is, $t_i = |T_i| (1 \le i \le n)$. We suppose the faults of the system are only caused by the parameters or the interactions of some parameters, and if an error is caused only by a combination of some k parameters where $1 \le k \le n$, every test case that includes the combination will also cause the error in testing. Let PT be a test set that covers all pair-wise combinations of parameter values. And there are m test cases in it, and each is like this: (v_1, v_2, …, v_n), where $v_i \in T_i$ and $1 \le i \le n$.

After testing the software with the test set PT, suppose there are l test cases in PT1 which cause errors in testing, and the other $m - l$ ones in PT2 which work well. The testing is successful for finding bugs in the software. Then we should analyze what cause the errors: a value v_i of parameter c_i; or a combination (v_i, v_j) of some two parameters c_i and c_j;…; a combination of some n-1 parameters values or a combination of all the n parameters values.

Let us consider the simplest case: the software failure is caused only by the value v_i of parameter c_i. Then the value v_i must appear in each test case of PT1 and will not be in any test case of PT2. For the general case, we can have the following conclusion:

Theorem 1. If a bug of the software under test is caused only by a combination of some k parameters where $1 \le k \le n$, the combination must appear in each test case of PT1, and must not appear in any of PT2.

Corollary 1. If the bugs of the software under test are caused by some combinations of some k parameters where $1 \le k \le n$, the combinations must only appear in PT1 and must not appear in PT2.

Corollary 2. If we can find the public composition in each test case of PT1, that is, if there are some parameter values or some parameter value combinations that appear in PT1 and don't appear PT2, then these public compositions have the maximum possibility to be the reasons that cause the errors.

Corollary 3. All the compositions that appear in PT1 and do not appear in PT2 maybe cause the errors. It is impossible for any of the compositions that appear in PT2 cause errors.

To locate the defects, we find all the compositions that are in PT1 and not in PT2, Such compositions form a set A. All the elements in the set may cause the errors. Generally the set A has more elements and need further reduction.

To reduce the set A, we design n test cases for each test case in PT1. For instance, $(v_1, v_2, \ldots, v_n)$ is one of the l test cases, and we can design the n test cases like these: $(*, v_2, \ldots, v_n)$, $(v_1, *, \ldots, v_n)$,…, $(v_1, v_2, \ldots, *)$, where "*" represents any value which is not the same as the original value in $(v_1, v_2, \ldots, v_n)$. Thus we need $n \times l$ additional test cases. By the result of these testing, we can reduce the two sets greatly and then obtain the conclusion.

Theorem 2. If a combination of some k values of test case $(v_1, v_2, \ldots, v_n)$ $(2 \le k \le n)$ causes the system error, when testing the software with the accordingly additional n test case generated as above, there must be n-k test cases which cause the same error.

Now we describe the algorithm for debugging of pairwise testing based on the above discussion. After testing the software with the test set PT, if we find some bugs in the running of some l test cases in PT1, we can analyze the result by the following four steps:

(1) Analyze the test cases in PT1and construct the set A.
(2) Construct the additional test cases for each of the l test cases as the Theorem 2.
(3) Test the software with the additional test suite and reduce the set A by throwing of all the elements that appear in PT2.
(4) Output the set A, which contains all the possible factors that may cause the errors.

3 Case Study

In this section, we take the testing for a telephone system [1, 2] as an example to show how to use the approach proposed in this paper. Table 1 shows four parameters that define a very simple test model. The test plan shown in Table 2 has 9 test cases but it covers every pair-wise combination of parameter values.

Table 1. Parameters for Phone Call

Call	Billing	Access	Status
Local	Caller	Loop	Success
Long	Collect	Isdn	Busy
Inter	Free call	Pbx	Blocked

Table 2. Pair-Wise Test Cases for Phone Call

Call	Billing	Access	Status
Local	Collect	Pbx	Busy
Long	Free call	Loop	Busy
Inter	Caller	Isdn	Busy
Local	Free call	Isdn	Blocked
Long	Caller	Pbx	Blocked
Inter	***Collect***	***Loop***	***Blocked***
Local	Caller	Loop	Success
Long	Collect	Isdn	Success
Inter	Free call	Pbx	Success

Table 3. The Additional Test Cases

Call	Billing	Access	Status
*	Collect	Loop	Blocked
Inter	*	Loop	Blocked
Inter	Collect	*	Blocked
Inter	Collect	Loop	*

Suppose a bug is found by the sixth test case in Table 2. By our algorithm, the set A is created at the first step, where A={(Inter, Collect), (Inter, Loop), (Inter, Blocked),

(Collect, Loop), (Collect, Blocked), (Loop, Blocked), (Inter, Collect, Blocked), (Inter, Collect, Loop), (Collect, Loop, Blocked), (Inter, Loop, Blocked), (Inter, Collect, Loop, Blocked)}. At the next step, four additional test cases shown in Table 3 are generated for the one causing the error. Next we test the system again with the additional test cases. We can suppose only the second test case in the Table 3 causes error in testing. In this case, using the algorithm we can reduce the set as A ={(Inter, Loop, Blocked), (Inter, Collect, Loop, Blocked)}. That is, if the error is caused only by the interactions of some parameter values, one of the combinations (Inter, Loop, Blocked) and (Inter, Collect, Loop, Blocked) must be the cause. The other cases can be dealt with and analyzed analogously.

4 Conclusions

In general, when finding some errors in pairwise testing, we must determine which factors cause the errors and then revise them, that is, we must debug the software. At present, researches in this field are very few. There are some ways about experiment result analysis in method of orthogonal experiment, such as the method of intuitive analysis, the method of variance analysis and so on, but these methods are not practical to analyzing testing result.

The problem of software debugging based on pairwise testing is very important and complex. This paper proposes a method based on the condition that the system errors only caused by the values of some parameters or the interactions of some parameter values, and if a bug is caused only by a combination of some k parameters where $1 \le k \le n$, every test case that includes the combination will also cause the bug in testing. As an assistant debugging tool of pairwise testing, we have implemented it in our ETS (embedded software testing supporting system). The experiment results show that our method can improve the efficiency and decrease the cost of software testing greatly.

References

1. Cohen, D. M., et al.: The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Trans. On Software Engineering, July 1997, 23(7)
2. Tai, K. C., Lei, Y.: A Test Generation Strategy for Pairwise Testing. IEEE Trans. On Software Engineering, Jan 2002, 28(1)
3. Noritaka Kobayashi, Tatssuhio Tsuchiya, Tohru Kikuno: A New Method for Constructing Pair-wise Covering Designs for Software Testing. Information Processing Letters 81(2002) 85-91
4. Mandl, R.: Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing. *Communications of the ACM*, Vol 28, no 10, pp. 1054-1058, Oct. 1985
5. Kuhn D R and Gallo A M, Software Fault Interactions and Implications for Software Testing. IEEE Trans. On Software Engineering, June 2004, 30(6):418-421
6. Colbourn C J, Cohen M B, and Turban R C.: A Deterministic Density Algorithm for Pairwise Interaction Coverage. **IASTED Proc. of the Intl. Conf on Software Engineering (SE 2004)**, Innsbruck, Austria, February 2004:345-352