Engineering Science and Technology, an International Journal xxx (2016) xxx–xxx

HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Full Length Article

A novel approach for deriving interactions for combinatorial testing

Sangeeta Sabharwal, Manuj Aggarwal *

Department of Computer Engineering, NSIT, New Delhi, India

ARTICLE INFO

Article history:
Received 15 December 2015
Revised 20 April 2016
Accepted 1 May 2016
Available online xxxx

Keywords:
t-way testing
Combinatorial testing
Interaction faults
Data flow techniques
DD path graph
Interaction testing

ABSTRACT

Combinatorial testing focuses on identifying faults that arise due to interaction of values of a small number of input parameters. Also known as t-way testing, it reduces the size of test set by selecting a minimal set of test cases that cover all the possible t-way tuples. An optimal value of t (degree of interaction) for t-way testing for the system would maximize fault detection count in minimal number of test cases. However, identification of an optimal value of for t-way testing for the system remains an open issue. In this paper, we present an approach to identify the interactions that exist in the source code, thereby reducing the count of interactions to be tested. DD path graph is generated from the source code and interactions are identified using data flow techniques. Two case studies are also discussed in order to demonstrate our approach. Experimental results indicate that our approach significantly reduces the count of interactions to be tested without significant loss of fault detection capability. The approach is extensible to large sized structured programs.

1. Introduction

Software testing is an expensive and time consuming activity that leads to production of reliable software systems [1,2]. Due to its importance, testing process is allocated a large share of the software development resources [3]. However, it is often observed that when the usage of large data-intensive software increases, the modules which have passed conventional testing methods start developing undetected errors [4]. The possible reasons include addition of records with an oddball combination of values that has not occurred before in the software. It is observed that these rare combinations of values which have escaped testing process and usage of software can cause interaction failures. To avoid such failures, it is desirable to test all combinations of values in an exhaustive manner. However, exhaustive testing is not feasible either due to time or resources availability. Thus a technique is required that focuses on testing combination of values.

Combinatorial (t-way) testing focuses on testing combinations of values. It is based on the observation that a large number of faults are caused by interactions of a few input parameters. Hence rather than testing all combinations in an exhaustive manner, combinations of only few parameters are tested. In order to generate test set, values for input parameters are selected such that every possible combination of values of any t parameters occurs at least once [5]. t is also known as the strength of coverage or interaction strength.

As an example, let us consider 3 input parameters, A, B and C, each can have 2 possible values, 0 and 1. Pairwise testing (where $t = 2$) would require the following 4 test cases as given in Table 1. Test cases are designed such that all possible pairs of values are getting covered.

As per studies, it is observed that maximum value of interaction strength is 4–6 for most of the systems [4]. As the value of interaction strength increases, the total number of detectable errors increases. But, an increase in interaction strength leads to an increase in the test set size, and hence increases the cost of testing. On the other hand, lower interaction strength leads to reduction in test set size which affects faults detection rate. Thus an optimal value of interaction strength can substantially reduce the testing costs without compromising fault detection capability. However, not much research is done in this area.

We have focused on two types of interaction failures as defined in the literature [2]. These are type 1 interaction failures and type 2 interaction failures. Type 1 interaction failures occur when a code segment in which a fault exists is executed. Due to interaction among variables, the faulty code is executed. For the pseudo code given in Fig. 1, a software system is observed to fail only for a set of customers residing at a particular location. Due to an interaction of two or more variables, a block of code is executed in which fault exists. Type 2 interaction failures occur when performing some computation on two or more variables leads to an incorrect result.

* Corresponding author.
E-mail addresses: ssab63@gmail.com (S. Sabharwal), mmanuj.aggarwal@gmail.com (M. Aggarwal).

Peer review under responsibility of Karabuk University.

http://dx.doi.org/10.1016/j.jestch.2016.05.008
2215-0986/

Please cite this article in press as: S. Sabharwal, M. Aggarwal, A novel approach for deriving interactions for combinatorial testing, Eng. Sci. Tech., Int. J. (2016), http://dx.doi.org/10.1016/j.jestch.2016.05.008

Table 1
Pairwise test set for a problem where each variable can have 2 possible values.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

```
Begin
if (customer belongs to set A){
    //some code here
    if(customer.residence==UK){
        //faulty code here
    }
    else{
        //block of code-executes normal
    }
}
End
```

Fig. 1. Pseudo code illustrating type 1 interaction failure.

Type 2 interaction failures are illustrated using pseudo code in Fig. 2. Here, placing an erroneous operator causes the set of variables involved in computation to produce an incorrect result. In this paper, we aim to identify interactions that may cause these two types of interaction failures to occur.

Data flow analysis techniques derive the information about the flow of data that exist in the program [6]. At each step in the program, the information about definition and usage of variables is obtained. The two usages defined are computation use (c-use) and predicate use (p-use) [2]. The usage of a variable is considered c-used if the variable occurs as a part of an assignment statement, as an argument passed to a function, in a subscript expression or in an output statement. The usage of a variable is considered p-used if the variable is used in a condition expression.

In this paper, a novel approach is presented that derives the interactions of variables that exist in the code. We have extended an existing work [7] by modifying their approach to handle the modularized programs where the basic modules are functions. The previous approach to identify the interactions among variables was meant for programs consisting of a single function only. Further, we have optimized the approach so that the resulting flow graph created using our approach contains less number of nodes as compared to the previous work. A flow graph is generated from the source code and data flow technique is applied on it. The c-use of a variable is redefined for the approach. From the flow graph and usage of variables, interactions are identified. The approach achieves a significant reduction in the count of interactions to be tested. As a result, rather than testing all the possible t-way interactions, only the identified interactions are tested which leads to reduction in testing costs. From the literature, the authors could not find any study that tries to identify the strength of coverage for a large sized system. Hence, our study may be considered as unique.

```
Begin
float x, y, z;
float result=(x*y)/z// it should be (x*y)-z
//block of code
End
```

Fig. 2. Pseudo code illustrating type 2 interaction failure.

The remainder of this paper is organized as follows. Next Section discusses the related work done in this field. In Section 3, the proposed approach to identify the interactions is explained. In Section 4, we have illustrated our approach with the help of case studies. Section 5 discusses the experimental results. In Section 6, threats to validity are given. Finally, in Section 7, some conclusions and future work are outlined.

2. Related work

Combinatorial testing (CT) has been widely studied and has become a well-accepted testing method. A large number of research articles have focused on CT. Researchers [8] have broadly classified them into eight categories. Category 1 includes all the articles that focus on generation of combinatorial test suites. Most widely Covering Arrays [9] are generated using metaheuristic methods such as Tabu Search, Simulated Annealing [10], Ant Colony Optimization, Particle Swarm Optimization, simplified swarm optimization [11] and Genetic Algorithms [12]. However, some authors have generated test suite using greedy methods [5], mathematical methods and recursive methods. A few researchers have taken into account the seeding and constraints. Category 2 includes all the articles that focus on test case prioritization. In CT, prioritization has been achieved either by reordering an existing test suite on the basis of prioritization criteria (coverage measurement, etc.) or generating prioritized test suite that includes important combinations first. Researchers [13] have designed formulas to compare the weights of the prioritization. Category 3 and 4 includes articles that evaluate how CT has improved the software quality and articles that focus on metrics. Most of the articles have considered combinatorial coverage as metric to evaluate the effectiveness of combinatorial testing. Category 5 includes the articles that study the application of CT to various types of applications. CT has been used for performance evaluation [14], feature testing of mobile phone applications [15,16], testing of network interface [17], etc. Category 6 includes articles that have taken into account the constraints [18]. A test suite must satisfy constraints and invalid test cases need to be eliminated.

In category 7, articles focusing on fault detection are discussed. Here, techniques to identify faults are discussed that have caused a failure to occur. It is required to identify the faults and remove them so as to improve the software quality [19]. Classification tree approach [20], classification and tuple relationships [21] are used for identification of failure inducing combinations. However, these approaches can identify the faults only when failures have occurred.

Last category includes all the articles that focus on modeling of System Under Test (SUT). Although building a precise model of SUT, such as Input Parameter Model [22] leads to effective CT, not much research has been done in this area. An SUT model for CT includes parameters and their values. It also includes the relationships among the parameters. These elements are identified from various documents generated during software life cycle, such as SRS, SDD, implementation, etc. Some researchers have identified parameters and their values from artifacts such as UML activity diagram [23] and UML sequence diagram [24]. Attempts have been made to reduce the size of test suite by reducing the count of parameters for SUT model [25] and identifying relationships between input and output parameters [26].

As can be concluded from the literature, CT has been extensively studied. Various research articles exist that focus on CT. As stated earlier, there exists a need to identify all the possible interactions among variables for t-way testing. However, not much research has been done to identify the strength of coverage for a large sized system. Some approaches, as mentioned in category 7

Please cite this article in press as: S. Sabharwal, M. Aggarwal, A novel approach for deriving interactions for combinatorial testing, Eng. Sci. Tech., Int. J. (2016), http://dx.doi.org/10.1016/j.jestch.2016.05.008