Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Combinatorial Testing: A Systematic Review

Ajay Chooramani*, Mrs. Sunita Garhwal
SMCA, Thapar University
Patiala, Punjab, India

Abstract— Combinatorial testing is applied for finding errors which are triggered by the interaction of parameters (configuration parameters and input parameters) of the software applications. Errors occur, when the usage of the software increases and interaction between those parameters grows rapidly. This paper protuberance with the aim of combinatorial testing, need, its application, strategies used for identifying interaction failures and tools which models combinatorial testing strategies. Finally, we discuss various direction of research in combinatorial testing and its future scope.

Keywords – Combinatorial testing, configuration testing, input parameter testing, AETG, IPO.

I. INTRODUCTION

Different types of testing aims for identifying different types of errors and faults. For example, mutation testing modifies the source code in a meager way that helps the tester to develop effective test cases. Similarly, combinatorial testing is focused on identifying errors and faults, occurs due to the interaction of different parameters of software. Combinatorial testing is performed by covering all the possible combination of parameter values.

Testing of a network gaming application running on the internet can be influenced by the number of parameters. These parameters are operating system, audio, graphics, number of players, internet access type, browser type, etc. Each parameter may have any number of possible values. The interaction of these parameters causes faults and errors in the application. Exhaustive testing is virtually impractical due to several possible combinations of parameters. Combinatorial Testing provides a better way to cover all the possible combinations with a better tradeoff between cost and time.

Combinatorial testing is based on the following concepts:

- **Interaction Rule:** Most failures occur due to a single factor or by the joint combinatorial effect (interaction) of two factors, with progressively fewer failures induced by interactions between three or more factors [1].
- **T-way Testing/Pairwise Testing**: Pair-wise testing [3] requires a given numbers of input parameters to the system, each possible combination of values for any pair of parameters covered with at least one test case.
- **Covering Array:** Covering array represents the test case selected under pairwise testing [1].

II. COMBINATORIAL TESTING

Combinatorial testing is a vital approach to detect interaction errors occurs because of interaction of several parameters. There are two approaches for combinatorial testing (shown in Fig. 1):

- Testing of configuration parameter values, or
- Testing of input parameter values.

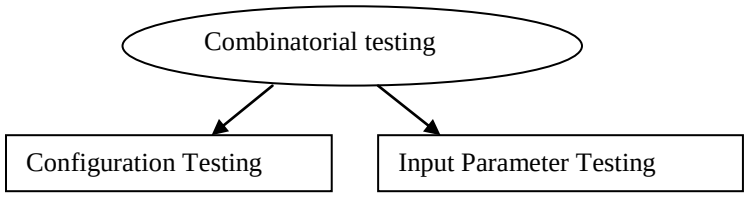

Fig. 1. Types of Combinatorial Testing

2.1 Configuration Testing

In configuration testing, a system is tested against each possible configuration of software and hardware that are supported. Consider an application that is purporting on platforms comprised of the following five components: An operating system (Windows 8, Apple OS X, Red Hat Enterprise Linux), browser (Internet Explorer, Firefox), protocol stack (IPv4, IPv6),

processor (Intel, AMD), and database (MySql, Sybase, Oracle). Total number of possible platforms is equal to 3*2*2*2*3=72. It is possible to check interaction of each component with other component using 10 test cases as shown in Table 1. Using these test cases, all possible pairs of platform components are covered.

TABLE I
4-WAY CONFIGURATION TESTING

Test	**OS**	**Browser**	**Protocol**	**CPU**	**DBMS**
1	Windows 8	IE	IPv4	Intel	MySql
2	Windows 8	Firefox	IPv6	AMD	Sybase
3	Windows 8	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySql
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHEL	IE	IPv6	AMD	MySql
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

2.2 Input Parameter Testing

In the input parameter testing [6], a system is tested against all the possible combination of values of parameters. Consider an on-line store that has four parameters of interest with following options:

- Three log-in types
- Three types of member status
- Three discount options
- Three shipping options.

Different end users may have different preferences and will probably to use different combinations of these parameters. For exhaustively testing, all combinations of the four parameters we need 81 test cases.

TABLE 2
SYSTEM'S PARAMETERS AND THEIR TYPE

Login type	**Member status**	**Discount**	**Shipping**
New member - not logged in	Guest	None	Standard (5-7 day)
New-member - logged in	Member	10% employee discount	Expedited (3-5 day)
Member - logged in	Employee	5% holiday discount	Overnight

In the online store, exhaustive testing requires 81 test cases, but pair-wise combinatorial testing uses only 9 test cases. Instead of testing each and every combination, all pairs of interactions between the parameters are tested. The resulting test suite is depicted in Table 3, which contain only 9 test cases.

TABLE 3
2-WAY INPUT PARAMETER TESTING

Test	**Login type**	**Member status**	**Discount**	**Shipping**
1	New member - not logged in	Guest	None	Standard (5-7 day)
2	New member - not logged in	Member	10% employee discount	Expedited (3-5 day)
3	New member - not logged in	Employee	$5offholiday discount	Overnight
4	New-member - logged in	Guest	$5 off holiday discount	Expedited (3-5 day)
5	New-member - logged in	Member	None	Overnight
6	New-member - logged in	Employee	10% employee discount	Standard (5-7 day)
7	Member - logged in	Guest	10% employee discount	Overnight
8	Member - logged in	Member	$5 off holiday discount	Standard (5-7 day)
9	Member - logged in	Employee	None	Expedited (3-5 day)

III. STRATEGIES FOR GENERATE COMBINATORIAL TEST SUITES

There are three types of algorithms that construct combinatorial test suites: algebraic, greedy, or heuristic search algorithms.

3.1 Algebraic method

Algebraic methods offer efficient constructions of test suits in regards to time. However, it is difficult to produce accurate results on a broad and general variety of inputs with algebraic methods [5].

3.1.1 Latin square

A Latin square [5, 8] is a table filled with *n x n* different symbols in such a way that each symbol should not repeat in any row or column again.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

Fig3. 3x3 Latin square

$$\begin{bmatrix} a & b & d & c \\ b & c & a & d \\ c & d & b & a \\ d & a & c & b \end{bmatrix}$$

Fig4. 4x4 Latin square

3.1.2 Orthogonal Array Representation

If each entry of an n x n Latin square is written as a triple (*row*, column, *symbol*), we obtain a set of triplets called the *n x* 2 orthogonal array [5] representation of the square. The orthogonal array representation of the 4x4 Latin square displayed above is { (1,1,a),(1,2,b),(1,3,d),(1,4,c), (2,1,b),(2,2,c), … , (4,3,c),(4,4,b) }.The triplet (2, 2, c) means that 2nd row and 2nd column consists of symbol c. Latin square can be defined in terms of orthogonal arrays [1, 5] as follows:

There are triplets of the form (*a, b, c*), where $1 \leq a, b, c \leq n$. 2*n*. All of the pairs (*a*, *b*) are different, all the pairs (*a*, *c*) are different, and all the pairs (*b*, *c*) are different.

3.1.3 Sequence Covering Array

In the event-driven software, wrong sequence of occurrence of events may cause a failure. For example, a failure might occur while connecting device A, only if device B is already connected. We define a sequence covering array[5,1] SCA(*N, S, t*) as an *N* x *S* matrix where entries are from a finite set *S* of *s* symbols, such that every *t*-way permutation of symbols from *S* occurs in at least one row, the *t* symbols in the permutation are not required to be adjacent. Sequence covering arrays, as the name implies, are analogous to standard covering arrays, which include at least one of every *t*-way combination of any *n* variables, where t<n. A variety of algorithms are available for constructing covering arrays, but these are not usable for generating *t*-way sequences because they are designed to cover combinations in any order. Consider the problem of testing for four events, *a*, *b*, *c*, and *d*. For convenience, a *t*-way permutation of symbols is referred to as a t-way sequence. There are 4! = 24 possible ways of these four events, but we can test all 3-way sequences of these events with only six tests (see Table 4)

TABLE 4
SEQUENCE COVERING AN ARRAY FOR EVENTS

Test	**Event1**	**Event2**	**Event2**	**Event4**
1	A	d	B	c
2	B	a	c	d
3	B	d	c	a
4	C	a	B	d
5	C	d	B	a
6	D	a	C	b

3.2 Greedy Algorithms

Numbers of algorithms have been developed for constructing covering arrays. These algorithms are efficient with respect to time and speed. Because the problem of generating a minimum test suit for combinatorial testing is NP-complete, most of these algorithms follows a greedy [5] approach. Two most important algorithms have been explained below:

3.2.1 IPO (In-Parameter-Order) Parameter Based Greedy Algorithm

It builds a t-way test set in an incremental manner

- A t-way test set is constructed with all the possible values of first t parameters.
- The test set is then extended to generate a t-way test set for the first t + 1parameters.
- Extend the test set for each additional parameter.

- Horizontal growth: It extends each existing test case by adding one value of the new parameter that covers maximum uncovered pairs.
- Vertical growth: adds new tests, for adding uncovered pair in the list of pair values.

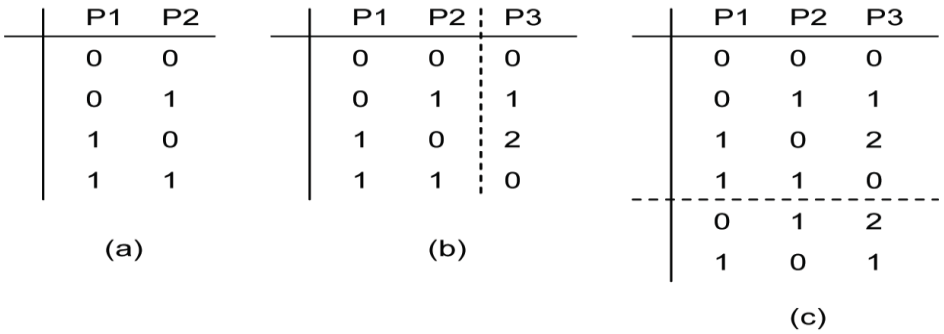

Fig. 5. Steps Involved in IPO

3.2.1.1 Steps for Implementing IPO

Step 1: Add first two parameters (p1, p2) in the table with all the possible pairs of values between them as in Fig. a.

Step 2: Add the third parameter (p3):

Suppose we are having a set S: (0, 1, 2) with all the values of parameter p3.

There are two more sets s1 (00,01,02,10,11,12) and s2 (00,01,02,10,11,12) having pair values of (p1-p3) and (p2-p3) respectively.

- Horizontal growth:
  Add value from set S such that adding this value of p3 will cover the maximum uncovered pairs from set S1and S2.Example: adding 0 in test case one of Fig. b, will cover 00 pairs of S1andS2. Adding 1 in test case two will cover 01 and 11 pairs of s1 and s2 respectively.
  Note: if two values of parameter p3 covers same number of uncovered pairs then take any of two values randomly for adding. After horizontal growth covered pairs of S1 and S2 are (00, 01, 12, 10) and (00, 11, 02, 10) respectively
- Vertical growth:
  Add test case which covers remaining uncovered pairs from set S1 and S2. As in Fig. c, this step covers 02 and 11 pairs of set S1 and 12 and 01 pairs of S2 respectively.

3.2.2 AETG (Automatic Efficient Test Case Generator) Test Case Based Greedy Algorithm

This method has provided many of the smallest test suites for different system configurations, however, at the cost of execution time to generate test suites [7].The proof of logarithmic growth [7] for the greedy algorithm assumes that at each stage it is possible to find a test case that covers the maximum number of uncovered pairs. It is not always computationally possible to find an optimal test case because there can be many possible test cases. Hence finding optimal test case suit is a NP-hard problem. We now outline a random greedy algorithm we developed.

For example: suppose we have a system with x test parameters and that the i-th parameter has v_i different possible values. Assume that we have already selected *r* test cases. We select the $r+1^{th}$ test case by first generating *M* different candidate test cases and then choosing one that covers the maximum new pairs. Each candidate test case is selected by the following greedy [5] algorithm.

- Choose a parameter p and a value v for p such that that parameter value appears in the greatest number of uncovered pairs.
- Let p_1 = p. Then choose a random order for other parameters. Then we have an order for all k parameters p_1 to p_x.
- Assume that values have been selected for parameters $p_1 ... p_j$. For v <= i <= j, let the selected value for p_i be called v_i. Then choose a value v_{j+1} for p_{j+1} as follows. For each possible value v for p_j, find the number of new pairs in the set of pairs {p_{j+1} = v and $p_i = v_i$ for v <= i <= j}. Then let v_{j+1} be one of the values that appeared in the greatest number of new pairs.

Note that in the above step, we have consider each parameter value only once for inclusion in a candidate test case and also, that choosing a value for parameter p_{j+1}, the possible values are compared with only the j values already chosen for parameters p_1 to p_j.

Step 1: adding a new row: To add first value new tests:

- Consider all the tuples not covered (UC)
- Choose a parameter p and a value v for p such that that parameter value appears in the greatest number of uncovered pairs.

- Fix p1 = p and choose a random order for the remaining parameters.

OS	CPU	IPV
WIN	INT	IPV4
WIN	AMD	IPV6

ow

UC	MISSING
LIN	INT
LIN	AMD
INT	IPV6
AMD	IPV4

s

Step 2:

- Take the next unassigned parameter
- Consider every value and choose the best
- Return step 1

F1	F2	F3
OS	CPU	IPV
	LIN	

CPU

F1	F2	F3
OS	CPU	IPV
LIN	INT	
LIN	AMD	

(LIN, INT)(LIN, AMD) both are new pairs

IPV

F1	F2	F3
OS	CPU	IPV
LIN	INT	IPV4
		IPV6

(INT, IPV6 is a new pair)

OS	CPU
IPV	
LIN	INT
	IPV4

Fig8. Adding a New Row in the Test Suit

IV History And Apllications Of Combinatorial Testing

Since 1980 combinatorial testing has been applied in various fields and applications. For example, ADA compiler was first tested by combinatorial coverage in 1985 by R. Mandl [10]. Brownlie et al**.** tested PELMOREX (Weather forecast generator) by pairwise combinatorial coverage in 1992 and developed the OATS (Orthogonal Array Testing Strategy) system for test case generation. Many papers have been published that share the experience of using CT in practice. Huller [11] in year 2000 uses CT to test the ground system of satellite communication. Borroughs, Jain and et al. [12] shows in their paper that, how CT improves the quality and efficiency of internet protocol testing. Williams and Probert [13] explained in 1996 that how CT can be used to test network interface has been also used in other application too. White and Almezen [9] in year 2000 proposed a method for testing GUI objects. Empirical study of this method shows that, if we follow CT strategies then a less number of test cases can also detect the defects of the GUI. Burr and Young [14] generated test cases with the help of CT to test the email system with AETG. Configuration testing and browser testing were also evolved as the child of CT.

V. Conclusions And Future Work

For last 30 years, a lot of research has been done in this field of testing and CT has proven itself as the best testing method in finding the interaction failures in software applications. This paper gives the answers to some of the basic questions about combinatorial testing like, what Combinatorial Testing is and its background, need of Combinatorial Testing, its application area, its various strategies to implement and its future in upcoming years. It is very obvious that all parameters in a system, does not involves in the interaction failures, so a better way is needed to find out the interaction and the dependency level between parameters and to determine values of each parameters . As the problem of finding optimal test suit is NP-hard, there is always a chance to find out improved test case generation methods. Integrating Combinatorial Testing with another type of testing could be a field of research too. We can combine it with unit testing or integration testing etc.

References

[1] D. R. Kuhn, N. Kacker, Yu Lei, "Practical Combinatorial Testing," NIST Special Publication Oct.2010.

[2] R. Othman and K. Zamli, "T-Way Strategies and Its Applications for Combinatorial Testing," International Journal on New Computer Architectures and Their Applications (IJNCAA): Pages 459-473, 2011.

[3] P. Flores and Y. Cheon, "Generating Test Cases for Pairwise Testing Using Genetic Algorithms," 18th IEEE International Symposium on Software Reliability Engineering (ISSRE'07), Dec.2007.

[4] B. Chen, J. Yan2 and J. Zhang, "Combinatorial Testing with Shielding Parameters," Asia Pacific Software Engineering Conference, 2010.

[5] M. Grindal, J. Offutt, S.F. Andler, "Combination Testing Strategies: A Survey Software Testing," Verification and Reliability, Vol. 15, Issue 3, pp. 167 – 199, 2005.
[6] Renée C. Bryce, Yu Lei and D.R Kuhn, DOI:10.4018/978-1-60566-731-7.ch014.
[7] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", IEEE Transactions On Software Engineering, Vol. 23, no. 7, Jul. 2008.
[8] C. Nie and H. Leung, "A Survey of Combinatorial Testing," 21st International Symposium ACM Computing Society, pp. 11-29, Jan.2011.
[9] L. White, H. Almezene, "Generating Test Cases for GUI Responsibilities using Complete Interaction Sequences," 11th International Symposium on Software Reliability Engineering (IISRE 2000), IEEE Computer Society, pp. 110-121.
[10] R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," Community ACM 28, 10, pp. 1054–1058, 1985.
[11] J. Huller, "Reducing Time to Market with Combinatorial Design Method Testing," International Council on Systems Engineering (INCOSE) Conference, 2000.
[12] K. Borroughs, A. Jain and R. L. Erickson, " Improved Quality of Protocol Testing through Techniques of Experimental Design," In Proceedings of the IEEE International Conference on Record, Serving Humanity Through Communications, Vol. 2, pp. 745–752, 1994.
[13] A. Williams and R. L. Probert, "A Practical Strategy for Testing Pair-wise Coverage of Network Interfaces," 7th International Symposium on Software Reliability Engineering (ISSRE'96), IEEE Computer Society, pp. 246, 1996.
[14] K. Burr, W. Young, "Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Code Coverage," International Conference on Software Testing Analysis and Review, pp. 503–513, 1998.

