A Chronological Survey on Combinatorial Testing Strategies

B.Vani and Dr. R. Deepa Lakshmi

1PG Scholar, Department of CSE, Velammal College of Engineering and Technology, Madurai, Tamil Nadu, INDIA.
2Professor, Department of CSE, Velammal College of Engineering and Technology, Madurai, Tamil Nadu, INDIA.

ABSTRACT

Combinatorial Testing (CT) can detect hard-to-find software faults triggered by interactions of parameters more efficiently than manual test case selection methods. It has been the active field of research today. Combinatorial strategy is a class of test case selection methods where test cases are identified by choosing values that satisfy the input criteria. Combinatorial Strategies are the test case selection methods where the test cases are identified by combining those values to test object input parameters. This paper targets to review the previous work on CT, present the evolution of CT, presents a clear picture of combinatorial testing strategies with basic algorithms and identifies the important issues, methods and applications of CT that directs a pathway to future research in CT. This survey also includes a hierarchy that attempts to relate various criteria associated with combination strategies.

INTRODUCTION

Testing, loosely considered to be the dynamic execution of test cases, consumes a significant amount of the resources in development projects. Thus, it is of great importance to investigate ways of increasing the efficiency and effectiveness in testing. Software Testing is a broad idea that involving many technical and non-technical areas. The general aim of testing is to affirm the quality of software systems by systematically exercising the software in controlled circumstances.

History Of Testing:

Dave Gelperin et al., and William C. Hetzel et al., in 1988 classified the phases and goals of software testing in the following stages:

- Until 1956 – Debugging Oriented
- 1957 – 1978 – Demonstration Oriented
- 1979 – 1982 – Destruction Oriented
- Later than 1988 – Prevention Oriented

Hierarchy Of Testing:

Combinatorial testing is a subject that crosses both computer science and mathematics. CT has attracted interest because we combine mathematical methods and computational methods from computer science to find solutions to testing problems.

Combination strategies define the way to select input parameters and combine them to test n-way interactions. Further the combination strategies are prioritized using criteria. This survey is an attempt to collect all this knowledge in one single place and form a basis for comparisons among the combinatorial strategies.
Fig. 1: Hierarchy of Testing.

Several existing test methods (eg., Equivalence partitioning, Boundary Value Analysis, Category Partitioning) are based on the model that the input space of the test object may be divided into subsets based on the assumption that all points in the same subset result in a similar behavior from the test object. This is called partition testing.

Even if the partition tests assumption is an idealization it has two important properties.
1) It lets the tester identify test suites of manageable size by selecting one or a few test cases from each subset.
2) It allows test effectiveness to be measured in terms of coverage with respect to the partition model used.

Test case selection using equivalence partitioning allows the tester to subdivide the input domain to relatively smaller set of subdomains. The equivalence classes are created assuming that program under test exhibits similar behavior. The tests generated by equivalence partitioning technique are not unique. The fault detection effectiveness depends on tester’s experience in test generation and familiarity in requirements and code. The effectiveness of Equivalence partitioning is less than one for most of the practical applications.

Bauer et al., Eduard et al. used the technique EP for test suite quality model transformation chains. Segura et al., Sergio et al. used the technique to reduce the number of test cases for functional testing of featured model analysis. Apa et al., Cecilia et al., and Oscar Dieste et al., used the technique in determining effectiveness for detecting faults within and outside the scope of testing techniques.

Experience indicates that programmers make mistakes in processing the boundary values of equivalence classes. The Boundary Value Analysis is a test case selection technique that target faults on the boundaries of equivalence classes whereas the EP method selects tests from within equivalence classes. Identify the boundary values for each partition. Select the test data such that boundary value occurs in at least one input. Itkonen et al., Juha, Mika V. Mantyla et al., and Casper Lassenius et al., proposed the role of the tester’s knowledge in exploratory software testing using BVA. Andrade, Javier, et al. developed an architectural model using BVA for software testing lesson learned systems.

A key issue in any partition testing approach is how partitions should be identified and how values should be selected from them. In early partition test methods like Equivalence Partitioning (EP) and Boundary Value Analysis (BVA), parameters of the test problem are identified. Each parameter is then analyzed in isolation to determine suitable partitions of that parameter. Support for identifying parameters and their partitions from arbitrary specifications are rather limited in EP and BVA. Ostrand et al., and Balcer et al., proposed the Category Partition (CP) method partially to address this problem.

The Category Partition method consists of a number of manual steps by which a natural language specification is systematically transformed into an equivalence class models for the parameters of test object.

Steps:

a) Identify functional units that may be tested separately.
b) Identify choices for each parameter and environment individually.
c) Determine the constraints among choices.
d) Generate all combinations of test frames that satisfy the constraint.
e) Transform the test frame into test case.

One alternative to partition testing is random testing, in which test cases are chosen randomly from some input distribution without exploiting information from the specification or previously chosen test cases.
Duran et al. and Ntafos et al. gave results that under certain very restrictive conditions, random testing can be as effective as partition testing. They showed consistent small differences in effectiveness between partition testing methods and random testing. Arcuri et al., Andrea et al., made a formal analysis of the effectiveness and predictability of random testing. These results were interpreted in favor of random testing.

Hamlet et al. and Taylor et al. later investigated and concluded that the Duran/Ntafos model was unrealistic. One main reason was that the overall failure probability was too high. Hamlet and Taylor theoretically strengthened the case for partition testing but made an important point that partition testing can be no better that the information that defines the subdomains. Guțăhjar followed up on Hamlet and Taylor’s results and showed theoretically that partition testing is consistently more effective than random testing under realistic assumptions. More recent results have been produced that favor partition testing over random testing in practical cases.

Reid et al., and Yin et al., Lebne-Dengelet et al., and Malayia et al., both performed experiments with different partition testing strategies and compared them with random testing. In all cases random testing was found to be less effective than the investigated partition testing methods. Thus adaptive random testing with combinatorial input domain is proposed to address the problem. Combination strategies can be used to decrease the number of test cases in the test suite.

**Paper outline:**

The section 2 deals with combinatorial testing, hierarchy of combinatorial testing strategies, combinatorial testing process with input space modeling. The section 3 deals with combinatorial techniques for test case generation. The section 4 deals with application of combinatorial testing. The section 5 gives concluding remarks of the paper.

**Combinatorial testing:**

Combinatorial Testing describes techniques for generating test set that is effective in discovery of faults due to interaction of various input variables. The Purpose of CT is to introduce techniques for generation of test configurations and test data using combinatorial technique, if detects with program inputs and their values respectively factors and levels. Allow selection of small set of test configurations from impractically large set. Effective in detecting faults arising out of factor interactions.

Software application designed to work in variety of environments. Eg: Combination of Factors OS, Network Connection, Hardware. Analogous Situation arises in testing of program having more than one input variables.

**Hierarchy Of Combinatorial Testing Strategies:**

The Combinatorial strategies labeled non-deterministic all depend to some degree of randomness. The simplest non-deterministic combination strategy is pure random selection of test cases. The groups of non-deterministic combination strategies also include two heuristic models CATS and AETG.

![Fig. 2.1: Combinatorial Testing Strategies.](image)

The deterministic combination strategies group is further divided into three subgroups, instant, iterative, and parameter-based. All of these combination strategies will always produce the same result from a specific input parameter model. The two instant combination strategies, Orthogonal Arrays (OA) and Covering Arrays (CA), produce the complete test suite directly. The largest group of combination strategies is iterative. They share the
property that the algorithms generate one test case at a time and add it to the test suite. Each Choice (EC), Partly Pair-Wise (PPW), Base Choice (BC), All Combinations (AC), and Anti-random (AR) all belong to the iterative combination strategies. The parameter-based combination strategy, In Parameter Order (IPO), starts by creating a test suite for a subset of the parameters in the input parameter model. Then one parameter at a time is added and the test cases in the test suite are modified to cover the new parameter. Completely new test cases may also need to be added.

The third main group of combination strategies, compound, includes all strategies in which two or more combination strategies are used together. Two such strategies are used to illustrate the concept.

**Combintorial Testing Process:**
The 3 Step process for generating TC for application under test. 
1. Modeling Input Space is not exclusive.
2. Combination of objects is array of factors and levels. i.e., Factor Covering Array generates test configurations.
3. Combinatorial Objects are used to generate Test Cases as per the requirement.

**Fig. 2.2:** Process for Generating Test Cases.

All Combinations generated might not be feasible, so omit infeasible combinations.

**Modeling the Input and Configuration Space:**
The Input Space of program P has K-tuples to be input during execution. Configuration Space of P consists of all possible settings of environment variables under which P is used. 
Example: Input Space Model for Online Pizza Delivery.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Large</td>
</tr>
<tr>
<td>Toppings</td>
<td>Custom</td>
</tr>
<tr>
<td>Address</td>
<td>Valid</td>
</tr>
<tr>
<td>Phone</td>
<td>Valid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Medium</td>
</tr>
<tr>
<td>Toppings</td>
<td>Preset</td>
</tr>
<tr>
<td>Address</td>
<td>Invalid</td>
</tr>
<tr>
<td>Phone</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Total Factor Combinations: $2^4 + 2^3 = 24$ Combinations
If Seven Levels for Toppings then no. of combinations = 52

**Combinatorial techniques:**
This section provides the overview of the different combinatorial technique along with the explanation for each strategy. The following are the list combinatorial techniques for the test case generation.
(a) Fault Model – Vectors
(b) Latin Squares
(c) Mutually Orthogonal Latin Squares
(d) Pairwise Design : Binary Factors
(e) Pairwise Design : Multi valued Factors
(f) Orthogonal Array
(g) Mixed Level Orthogonal Arrays
(h) Covering Arrays
(i) Mixed-Level Covering Arrays
(j) Arrays of Strength >2

(a) Fault Model – Vectors:
Test input and configurations reveal certain faults called interaction faults. Combination of t greater than or equal to one input values causes the program to enter invalid state. Faults triggered by some value of input
 variable if \( t=1 \) then it is called as simple faults, if \( t=2 \) then pairwise interaction faults and if \( t>2 \) then called as \( t \)-way interaction faults or \( t \)-factor faults. Interaction faults occur due to program interaction.

Example: Consider the program with 3 inputs \( X,Y,Z \). Values assigned prior to execution \( X=\{x_1,x_2,x_3\} \), \( Y=\{y_1,y_2,y_3\} \), \( Z=\{z_1,z_2\} \)

<table>
<thead>
<tr>
<th>Program:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
</tr>
<tr>
<td>int X,Y,Z;</td>
</tr>
<tr>
<td>input (X,Y,Z);</td>
</tr>
<tr>
<td>if (X==x_1 and Y==y_2)</td>
</tr>
<tr>
<td>output (f(X,Y,Z));</td>
</tr>
<tr>
<td>else if (X==x_2 and Y==y_1)</td>
</tr>
<tr>
<td>output (g(X,Y));</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>output ((f(X,Y,Z)+g(X,Y)) // Statement is not protected correctly</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

This program contains 1 error. The error can be revealed by executing program with \( X=x_1 \) and \( Y=y_1 \) and any value of \( Z \). Here \( Z \) doesn’t play a role in triggering the fault but needed to reveal the fault.

Fault Vectors:

\( V = \) Fault Vector for program P if execution (P) with test cases derived from v triggers fault in P.

Note: \( t \)-way fault vector for P triggers \( t \)-way fault in P. Given \( k \) factors, there are \( (t \cdot k) \) don’t care entries in \( V \).

Example: 2-way fault vector \((2,3,*\})\)
i.e., factor 1 set to 2, factor 2 set to 3, factor 3 set to don’t care entry. Practically if \( t \) is set to 2, reveals pairwise interaction faults. If \( t \)-way runs reveals higher level interaction faults.

(b) Latin Squares

The name derived from ancient arrangement of Latin alpha in square. Latin squares and Mutual Orthogonal Latin Squares (MOLS) are ancient devices found useful in selecting subset of factor combinations from complete set. Latin Squares generate fewer factor combinations than brute-force technique.

Description: Let \( S \) be the finite set of \( n \) symbols. A Latin Square of Order(n) = \{ nxn matrix / no symbol appears > once in rows and columns \}

Example: Given \( S = \{A,B\} \) we have 2 Latin Squares of Order(2)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

Given \( S = \{1,2,3\} \) we have 3 Latin Squares of Order(3)

\[
\begin{array}{ccc|cc}
1 & 2 & 3 & 2 & 1 \\
2 & 3 & 1 & 3 & 2 \\
3 & 1 & 2 & 1 & 2 \\
\end{array}
\]

Latin Squares of Order (n) constructed by creating a row of \( n \) distinct symbols. Additional rows created by permuting first row. Latin Squares of Order \( n>2 \) is constructed easily by doing modulo arithmetic.

(c) Mutually Orthogonal Latin Squares (MOLS):

Let \( N_1 \) and \( N_2 \) be 2 Latin Squares with Order (n). Creating an nxn matrix from \( N_1 \) and \( N_2 \), i.e., simply juxtapose the corresponding elements.

\[
\begin{array}{cc|ccc}
1 & 2 & 3 & 2 & 3 \\
2 & 3 & 1 & 1 & 2 \\
3 & 2 & 1 & 3 & 2 \\
\end{array}
\]

\[
N_1 = \begin{array}{ccc}
1 & 2 & 3 \\
3 & 1 & 2 \\
2 & 3 & 1 \\
\end{array}, \quad N_2 = \begin{array}{ccc}
1 & 2 & 3 \\
3 & 1 & 2 \\
2 & 3 & 1 \\
\end{array}, \quad L = \begin{array}{ccc}
12 & 23 & 31 \\
32 & 13 & 21 \\
21 & 31 & 12 \\
\end{array}
\]

i.e., each element appears exactly once and hence \( M_1 \) and \( M_2 \) are mutually orthogonal. MOLS do not exist for \( n=2 \) and \( n=6 \) called as Eulerian numbers.

(d) Pairwise Design: Binary Factors:

Pairwise Design focus on programs where input space can be modeled as factors assuming 1 or 2 values.

Example: Consider factor combination \( 2^3 \) with 3 variables \( X, Y, Z \). \( X=\{x_1, x_2\}, Y=\{y_1, y_2\}, Z=\{z_1, z_2\} \). If paired then there are 12 pairs. A set of four possible combinations suffices, i.e., \( (x_1 y_1 z_2), (x_1 y_2 z_1), (x_2 y_1 z_1), (x_2 y_2 z_2) \). Generalizing problem of pairwise design to \( n>2 \) factors. Define \( S_{2k-1} = \{\text{Binary String of length } 2k-1\} \), i.e., exactly \( \binom{2k-1}{k} \) strings in \( S_{2k-1} \).
Procedure: SAMNA
Input: n: Number of two-valued input variables
Output: set of factor combinations by all pairs of input values covered.
Begin Procedure: SAMNA
/* x1, x2, …, xn denote n input variables. xi’ denote one of the 2 possible value for xi, 1<=i<=n. One Value = 0 & other = 1. */
Step 1: Compute the smallest integer k/n<=| S2k-1 |
Step 2: Select subset of n strings from S2k-1. Arrange to nx(2k-1) matrix with columns of different bits.
Step 3: Append column of 0’s to end of string of size =n. Increase the size from 2k-1 to 2k.
Step 4: Each 2k column contain a bit pattern of combination (x1*, x2*, … xn*) and variable selected is column i.
End Procedure

(e) Pairwise Design: Multi valued factors:
MOLS are used to construct test configurations where number of factors is two or more, number of levels for each factor >2, all factors having same number of levels.

Procedure: PDMOLS
Input: n: number of factors
Output: A set of factor combinations therefore all level pairs are covered.
Begin Procedure: PDMOLS
/* F1’, F2’, …Fn’ denote n factors. X_{i,j} denote jth level of ith factor */
Step 1: Relabel the factors F1, F2, …Fn such that ordering is satisfied.
Step 2: Prepare a table containing n columns and bxk rows divided into b blocks.
Step 3: Fill column F1 with 1’s in block1 and 2’s in block2 and so on. Fill column F2 with sequence 1,2, …, k.
Step 4: Find s=n(k) MOLS of Order k. Denote MOLS M1, M2, …Ms.
Step 5: Fill block 1 column F3 with entries from Column1 of M1. If number of block b=b1>k then reuse columns M1 to fill remaining
Step 6: This lists nine factors. Each combination used to generate more tests.
End Procedure

Shortcomings of MOLS for test design:
(1) A sufficient number of MOLS might not exist for the problem at hand.
(2) While MOLS assist with generation of balanced design in which all interaction pairs are covered an equal number of times. The number of test configurations is larger than achieved using other methods.

(f) Orthogonal Arrays:
An Orthogonal Array is an NxN matrix in which entries are from finite set S of s symbols. Nxsubarray contains each t-tuple exactly same number of times. It is denoted by OA(N, k, s, t). The index of OA is denoted by λ = N/s^t. N is the number of runs and t is the strength of orthogonal array.
Example: OA(4, 3, 2, 2). K=3

Table 3.1: Orthogonal Array Runs.
<table>
<thead>
<tr>
<th>Run</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Index λ = 4/2^2 = 1. Each pair appears exactly once in 4x2 subarray. There are 4 pairs (1,1) , (1,2), (2,1), (2,2).

(g) Mixed Level Orthogonal Arrays:
The OA in previous example is known as fixed level orthogonal arrays. The Mixed Level Orthogonal Array of strength t is denoted by MA(N, S1,k1, S2,k2, ... Sn,kn, t) indicates N runs where k1 factors at S1 level and k2 factors at S2 level. The formula used for calculating λ does not apply to MLOA.
Example: MA(8,2^4,2)

Table 3.2: Mixed Level Orthogonal Array Runs.
<table>
<thead>
<tr>
<th>Run</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
So far the techniques examined for generating balanced combinatorial designs and how to use test cases [9]. The balance requirement is essential in statistical experiments, it is not so in software testing.

For Example: Software application tested once for a given pair of factor levels, there is no need to test the same pair unless it behaves non-deterministically. One might test the application again to ensure reliability. The balance requirements can be relaxed by the following techniques:

(i) **Covering Arrays:**
A Covering Array CA(N, k, s, t) is an N×k matrix in which entries are from a finite set S of s symbols such that N×t subarray contains each possible t-tuple at least λ times. N denotes number of runs, k the number of factors, s the number of levels, t the strength and λ the index.

**Key Difference between OA and CA:**
In OA each possible t-tuple occur λ times in any N×t subarray. In CA each possible t-tuple occur at least λ times in N×t subarray. This key difference leads to combinatorial designs that are often smaller in size than OA. CA is often referred to as unbalanced design.

Example: OA(8, 5, 2, 2) denoted in CA as (6, 5, 2, 2)

(ii) **Mixed Level Covering Arrays:**
MCA(N, S₁, k₁, S₂, k₂, … , S_p, k_p, t) refers to N×Q matrix entries such that Q= Σ_k_i p_i = 1, each N×t subarray contains at least one occurrence of each t-tuple corresponding to t columns.

(iii) **Arrays of Strength > 2:**
In all the previous techniques the strength of the array is t=2. These tests are targeted at discovering errors due to pairwise interactions. Sometimes higher strength is needed to achieve higher confidence to ensure correctness of software.

Example: Consider t=3 leads to test that targets errors due to 3-way interactions.(Air vehicle). These devices are controlled by several complicated algorithms for sensing, computing and control monitors. Testing the parameters in operational flight software is difficult and complex activity. They are tested against combinations of input parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Levels</th>
<th>Levels</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take Off Mode</td>
<td>AA1</td>
<td>VV1</td>
<td>DD-R</td>
</tr>
<tr>
<td>Autopilot</td>
<td>Normal</td>
<td>Prolonged</td>
<td>Shortened</td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>Normal</td>
<td>Degraded</td>
<td>Upgraded</td>
</tr>
<tr>
<td>GPS Controller</td>
<td>Normal</td>
<td>Degraded</td>
<td>Upgraded</td>
</tr>
<tr>
<td>Aided Navigator</td>
<td>Normal</td>
<td>Degraded</td>
<td>Upgraded</td>
</tr>
</tbody>
</table>

Due to the high reliability of operational flight software there is a need to ensure that no pairwise faults or higher level interaction fault occur.

**Application Of Combinatorial Testing:**
Burroughs et al. (1994) reported how quality and efficiency of protocol testing were improved by CT. Williams and Probert et al., (1996) described a pairwise strategy of network interface. Dunietz et al. (1997) examined the correlation between t-way coverage and achieved code coverage. Huller et al., (2000) used CT to test the ground system for satellite communications.


**Conclusion:**
In last 20 years, combinatorial testing has been widely used, studied and applied. It is well accepted testing methodology that has proven its ability of detecting interaction failures. This article makes the systematic and complete survey of CT, focusing mainly on test case generation and selection. The focus of CT for the research in the near future can include the expansion in different levels of testing. The focus can be extended to detect the failures caused by interactions of more than 2 factors in system under test.
REFERENCES


Andrade, Javier, 2013. "An architectural model for software testing lesson learned systems." Information
and Software Technology, 55.1: 18-34.

Apa, Cecilia and Oscar Dieste, 2014. "Effectiveness for detecting faults within and outside the scope of

Arcuri, Andrea, Muhammad ZohaibIqbal and Lionel Briand, 2010. "Formal analysis of the effectiveness and
predictability of random testing." Proceedings of the 19th international symposium on Software testing and
analysis.ACM.


chains." Objects, Models, Components, Patterns.Springer Berlin Heidelberg, 3-19.

Software Engineering, 16(12): 1402-1411.


Yin, H., Z. Lebne-Dengel and Y.K. Malaysia, Automatic Test Generation using Checkpoint Encoding and

Huang, Rubing, Jinfu Chen and Yansheng Lu, 2014. "Adaptive Random Testing with Combinatorial Input

Itkonen, Juha, Mika V. Mantyla and Casper Lassenius, 2013. "The role of the tester's knowledge in


Myers, J., Glenford, 2011. Corey Sandler et al., and Tom Badgett et al. The art of software testing.John
Wiley & Sons.

(CSUR), 43.2: 11.

Nie et al., Changhai et al., Hareton Leung et al., and Kai-Yuan Cai et al. "Adaptive Combinatorial

Poon, Pak-Lok, 2011. "Contributions of tester experience and a checklist guideline to the identification of

Reid, C., Stuart, 1997. "An empirical analysis of equivalence partitioning, boundary value analysis and

Reid, S., 2011. An empirical analysis of equivalence partitioning, boundary value analysis and random
testing.In Proceedings of the 4th International Software Metrics Symposium (METRICS'97), Albuquerque, New
Mexico, USA, 5-7, pages, 64-73. IEEE.

Segura, Sergio, David Benavides and Antonio Ruiz-Cortés, 2011. "Functional testing of feature model

Shahbazi, Ali, Andrew, F. Tappenden and James Miller, 2013. "Centroidalvoronoi tessellations-a new


Software, 20.6: 1393-1405.