DEVELOPMENT OF INTERACTION TEST DATA GENERATION STRATEGY WITH INPUT-OUTPUT MAPPING SUPPORTS

ONG HUI YEH

UNIVERSITI SAINS MALAYSIA

2012

DEVELOPMENT OF INTERACTION TEST DATA GENERATION STRATEGY WITH INPUT-OUTPUT MAPPING SUPPORTS

by

ONG HUI YEH

Thesis submitted in fulfillment of the requirements
for the degree of
Master of Science

January 2012

ACKNOWLEDGEMENTS

First of all, I would like to show my sincere gratitude and thank to my supervisor Assoc. Prof. Dr. Kamal Zuhairi Zamli. His endless enthusiasm, constantly advice have encouraged me in completing this research work of master degree.

Next, my appreciations are expressed towards the administrative staffs in the School of Electrical and Electronic Engineering and also Institute of Postgraduate Studies, for their passionate to handle all forms of official tasks during my master study, especially dedicated to the Dean of School of Electrical and Electronic Engineering, Professor Dr. Mohd Zaid Abdullah. The appreciations also go to my postgraduate friends, particularly Miss Lim Su Rong, Mr. Tiang Tow Leong and Mr. Eng Swee Kheng. Their helpful supports and encouragements are highly appreciated.

Last but not least, this research work of master study is funded and supported by USM Postgraduate Research Grant Scheme (PRGS) - “Development of Interaction Test Data Generation Using Input-Output Mapping Supports”, the generous fundamental grants - “Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique” from Ministry of Higher Education (MOHE), USM research university grant - “Development of Variable Strength Interaction Testing Strategy for T-Way Test Data Generation” and USM postgraduate fellowship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACA	Ant Colony Algorithm
ACO	Ant Colony Optimization
ACS	Ant Colony System
AETG	Automatic Efficient Test Generator
AURA	Automated Random Algorithm
GA	Genetic Algorithm
GA-N	Genetic Algorithm *n*-way
IDE	Integrated Development Environment
IPO	In-parameter-order
IPO-N	In-parameter-order *n*-way
IPOG	In-parameter-order-general
IPOG-D	In-parameter-order-general D-construction
JDK	Java Development Kit
JVM	Java Virtual Machine
MC-MIPOG	Multi-core Modified-in-parameter-order-general
MDI	Multiple Document Interface
MIPOG	Modified-in-parameter-order-general
PICT	Pairwise Independent Combinatorial Testing
SA	Simulated Annealing
SUT	System under Test
TCG	Test Case Generator
TConfig	Test Configuration
TVG	Test Vector Generator

PEMBANGUNAN STRATEGI PENJANAAN DATA UJIAN INTERAKSI DENGAN SOKONGAN PEMETAAN MASUKAN-KELUARAN

ABSTRAK

Pengujian dengan kekuatan seragam *t*-way (di mana *t* mewakili kekuatan interaksi) adalah asas pengujian interaksi. Walau bagaimanapun, *t* jarang seragam di dunia sebenar kerana bukan semua kesalahan interaksi semata-matanya dibentukkan oleh *t*-interaksi yang tetap. Oleh yang demikian, satu penyelesaian umum diperkenalkan: pengujian interaksi berasaskan hubungan masukan-keluaran. Walaupun berguna, kebanyakan pelaksanaan strategi-strategi yang sedia ada kurang menekankan sokongan pemetaan masukan-keluaran berautomatik (untuk menterjemahkan keluaran simbol kembali ke bentuk data sebenar) dan kebolehlenturan penjanaan kes-kes ujian. Untuk menangani isu-isu tersebut, satu pengujian interaksi strategi yang berasaskan tidak berketentuan hubungan masukan-keluaran, AURA, telah dibangunkan. Strategi AURA juga bersepadu dengan sokongan pemetaan masukan-keluaran pascapemprosesan berautomatik dan kemampuan mengawal lelaran boleh lentur untuk menyokong kebolehlenturan penjanaan kes-kes ujian. Keputusan ujikaji menunjukkan bahawa strategi AURA menjana saiz kes-kes ujian bersaing terhadap strategi-strategi yang sedia ada (Density, ParaOrder, Union, TVG, PICT, AETG, ACA, GA-N, IPO-N, IPO, Jenny, SA dan ACS). Khususnya, strategi ini mampu menjanakan saiz kes-kes ujian yang optimum seperti strategi-strategi lain bagi masukan yang tertentu. Akhir sekali, sokongan pemetaan masukan-keluaran pascapemprosesan berautomatik dan kemampuan mengawal lelaran boleh lentur dinilai dengan menjalankan ujikaji.

DEVELOPMENT OF INTERACTION TEST DATA GENERATION STRATEGY WITH INPUT-OUTPUT MAPPING SUPPORTS

ABSTRACT

Uniform strength *t*-way testing (where *t* represents interaction strength) forms the basis of interaction testing. However, *t* is rarely uniform in real world as not all interaction faults are solely constituted by these fixed *t*-interactions. Consequently, a general solution has been introduced: input-output based relationship interaction testing. Although useful, most existing strategy implementations are lacking in terms of the automated input-output mapping support (to translate the symbolic outputs back into actual data form) and test suite generation flexibility. In order to address these aforementioned issues, a non-deterministic input-output based relationship interaction testing strategy, AURA, has been developed. AURA strategy also integrated with post-processing automated input-output mapping support and flexible iteration control capability to support test suite generation flexibility. Experimental results indicated that AURA strategy is generating competitive test suite size against existing strategies (Density, ParaOrder, Union, TVG, PICT, AETG, ACA, GA-N, IPO-N, IPO, Jenny, SA and ACS). Specifically, this strategy is capable to generate the test suite size as optimized as other strategies for certain inputs. Lastly, the post-processing automated input-output mapping support and flexible iteration control capability are evaluated with experiments.

CHAPTER 1

INTRODUCTION

Software is often referred to the written programs or procedures or rules and associated documentation that pertaining to the operations of a computer system (Catty, 2010; Khurana, 2010; Partsch, 1990). Nowadays, software systems affect almost all aspects of our lives. In fact, most hardware implementations are gradually being substituted by their software counterpart whenever possible such as washing machine controllers, mobile phone applications as well as sophisticated airplane control systems (Younis, Zamli, & Isa, 2008b).

Several numbers of factors are attributed on the growing dependency on software instead of hardware. Firstly, unlike hardware products that would degrade along the timeline, software products never wear out. Besides that, software products could be duplicated easily and less expensive if compared with hardware that required costly setup budgets to be reproduced. Therefore, the use of software able to reduce maintenance costs. In addition, software is malleable, i.e. easy to be changed and manipulated as required, which ease product improvement (Younis et al., 2008b).

Despite of the advantages that software possessed, software failures certainly have diverse effects ranging from personal inconvenience up to catastrophic disaster. For instance, it is widely known that most software systems likely contain undetected faults with unknown (and perhaps unknowable) consequences (Chung, 1994). Different from hardware faults where one circuit can often be backed up by another

spare, software faults sometimes will have a cascade effect of propagating exponentially far beyond their origin point resulting in large-scale disruption. Therefore, one possible analogy is to view software systems as time bombs since the detonation time is unknown (Chung, 1994).

Hence, software reliability has become an essential factor especially when it is employed in harsh, life threatening or critical (safety) applications such as airplane control systems and biomedical instrumental devices. Rigorous software testing is required to ensure the conformance and quality of software (Younis et al., 2008b).

1.1 Software Testing

Software testing plays an important role in the process of creating and delivering high quality software products (Berndt & Watkins, 2005; Yuan & Gu, 2006). Meanwhile, by covering as much as 30 to 50 percent of overall costs, software testing is an essential part in software system development lifecycle (Cui, Li, & Yao, 2009; Gao & Hu, 2009; Schroeder, Eok, Arshem, & Bolaki, 2003; Younis, Zamli, & Isa, 2008a). Technically, software testing can be regarded as any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results (Myers, Badgett, Thomas, & Sandler, 2004; Watkins & Mills, 2010).

Software testing is also defined as the process of executing a program or system with the intent of finding errors (Myers et al., 2004; Sobh, 2010). This testing phase is crucial to ensure quality (i.e. reliability, functionality, usability, efficiency

maintainability and portability) of the end products before they can be delivered to the users (Younis & Zamli, 2010).

1.2 Test Case Design

In order to ensure software quality and conformance to specifications, there is a need to exhaustively test them. Yet, exhaustive testing is practically impossible (Younis & Zamli, 2009a). Addressing this issue, many test case design strategies have been developed in the literature (e.g. equivalence partitioning, boundary value analysis, decision tables, and random testing) to help sample out test data into manageable ones (Basili & Selby, 1987; Beer & Mohacsi, 2008; Kuhn, Wallace, & Gallo, 2004; Reid, 1997):

- Equivalence Partitioning

  Equivalence partitioning divides the set of all possible inputs into equivalence classes. The equivalence relation describes the properties for which input sets are belonging to the same partition. This strategy will reduce the number of combinations of inputs and output values that used for testing, thereby increasing the coverage and reducing the testing effort (Cechich, Piattini, & Vallecillo, 2003; Ramesh, 2009).

- Boundary Value Analysis

  Boundary value analysis is based on the assumption that bugs are likely detected when inputs or state values are at or very near to a minimum or maximum of an equivalence partition. Therefore, this strategy will select test

cases that exercising bounding values (i.e. boundaries of the input domain) (Binder, 2000; Dasso & Funes, 2007).

- Decision Tables

  A decision table is consisted of the decision variables, the conditions (or values) by each of the decision variables, and the actions to take in each combination of conditions. This strategy is often used to express rules and regulations for embedded systems and administrative systems and effective to generate test cases in scenarios which depends on the values of decision variables (Hass, 2008; Ramesh, 2009).

- Random Testing

  Random testing is a strategy that based on the random selection of test cases from the entire input domain. This strategy is often applied in conjunction with other test case design strategies. For example, after identifying the equivalence partitions, random testing can be used to select test cases from each of the identified equivalence partitions (Gao, Tsao, & Wu, 2003; Saleh, 2009).

Although useful, these test case design strategies do not sufficiently cater for faults due to interaction (Calvagna, Gargantini, & Tramontana, 2009). For this reason, interaction testing strategies have started to emerge. In interaction testing, a set of test cases is generated to cover a subset of the possible combinations of the system's input parameters, rather than trying to cover all possible combinations.

It is noted that from software engineering perspective, a test case is defined as a set of conditions or variables under which a tester will determine whether a software system is working correctly or not (David, Michal, Nabendu, & Natarajan, 2011). A set of test cases also can be denoted as test suite or test data.

The most fundamental interaction testing is *t*-way testing. *t*-way testing works on detecting faults that caused by *t*-way interaction of variables whereby *t* indicates the interaction strength (Kuhn, Yu, & Kacker, 2008). Conventionally, the rationale for *t*-way testing stemmed from the fact that from empirical observation, the number of parameters involved in software failures is relatively small (i.e. in the order of 2 to 6), in some classes of software (Kimoto, Tsuchiya, & Kikuno, 2008; Kuhn & Vadim, 2006; Kuhn et al., 2004; Zamli & Younis, 2010). If *t* or fewer variables are known to cause fault, test suite can be generated on some *t*-way combinations, then give rise to a smaller size of test suite (Younis & Zamli, 2009b).

1.3 Problem Statement

Numerous efficient *t*-way testing strategies have been proposed in the past literatures (Ahmed & Zamli, 2010; Kim, Choi, Hoffman, & Jung, 2007; Klaib, Zamli, Isa, Younis, & Abdullah, 2008; Shi, Nie, & Xu, 2005; Xu, Xu, Nie, Chu, & Chang, 2003) to generate optimized test cases for software system under test (SUT). However, *t* is rarely uniform in real world as not all interaction faults from typical software SUT are solely constituted by these *t*-interactions (Cohen, Gibbons, Mugridge, & Colbourn, 2003a). Therefore, variable strength interaction testing strategy is then been proposed to support this aforementioned concern (Cohen et al., 2003a; Cohen, Gibbons, Mugridge, Colbourn, & Collofello, 2003b; Wang, Xu, & Nie, 2008; Zamli

& Younis, 2010). This approach no doubt solves some of real considerations by allowing certain subsets to cover higher *t*-interactions, though; it is still insufficient to generate test cases based on actual interactions (Wang et al., 2008). To overcome this limitation, a general solution has been introduced: input-output based relationship interaction testing, which focus on those input combinations that affect a program output, rather than considering all possible input combinations (Patrick & Bogdan, 2000; Patrick, Pat, & Bogdan, 2002; Wang, Xu, & Nie, 2007; Zabil, Zamli, & Othman, 2011).

Meanwhile, as far as implementation is concerned, most existing strategy implementations (Ahmed & Zamli, 2010; Lei, Kacker, Kuhn, Okun, & Lawrence, 2007; Younis & Zamli, 2010; Yu & Tai, 1998) generate their outputs in terms of symbolic parameters for ease of data manipulation. This could be straightforward but not user friendly approach because test engineers have to manually map these symbolic values to actual data one by one before they could execute on them. As the test case number is predominantly large especially in highly configurable software systems, these could be another problematic issue in term of time and cost consumed as well as the accuracy of test cases (due to the potential of human errors on manually mapping process) (Zamli, Klaib, Younis, Isa, & Abdullah, 2011). Hence, there is a need for automated input-output mapping to seamlessly translate the symbolic outputs back into the actual data form.

Apart from automated input-output mapping, existing strategy implementations are also lacking as far as flexibility of test suite generation is concerned. Here, the problem of interaction test suite generation can be seen as two sides of the same coin

with optimal size and test generation time being the sides. On one side of the coin, when the optimality of test suite size is preferred than generation time, a strategy need to be adaptable to generate more optimized test suite. On the other side of same coin, a strategy needs to be flexible enough to generate fast test suite but in expense of optimality (Cohen, Dalal, Fredman, & Patton, 1997).

1.4 Thesis Aim and Objectives

The main aim of this research is to develop and evaluate a flexible input-output based interaction testing strategy with automated input-output mapping supports, called Automated Random Algorithm (AURA), for combinatorial test data generation. The main objectives of the work undertaken are:

i. To develop and investigate AURA strategy as a test data generation tool.
ii. To integrate the post-processing input-output mapping supports as part of AURA strategy.
iii. To integrate the flexible iteration control for constructing interaction test suite, as part of AURA strategy.
iv. To evaluate and compare the performance of AURA strategy in terms of test size against existing works (Density, ParaOrder, Union, TVG, PICT, AETG, ACA, GA-N, IPO-N, IPO, Jenny, SA and ACS).

1.5 Research Methodology

Generally, the research methodology of this thesis consists of a number of phases as shown:

- **Phase 1: Literature Review**

  In this phase, a comprehensive literature survey is performed to establish the state-of-the-art on test case design of interaction testing.

- **Phase 2: Development of AURA Strategy**

  Upon completion of literature review, the proposed solution, AURA strategy is developed and tested based on research aim and objectives.

- **Phase 3: Experimental Verification and Evaluation**

  After AURA strategy is developed, the correctness of this strategy is then verified practically. Meanwhile, this phase also evaluates AURA strategy by conducting several experiments which includes characterization and benchmarking inputs evaluation of the strategy.

- **Phase 4: Research Documentation**

  Finally, a concrete conclusion with possible future work is also discussed here. In this phase, all research details are summarized as a whole in the thesis for documentation purpose.

For instance illustration, Figure 1.1 depicts the general flow of research methodology.

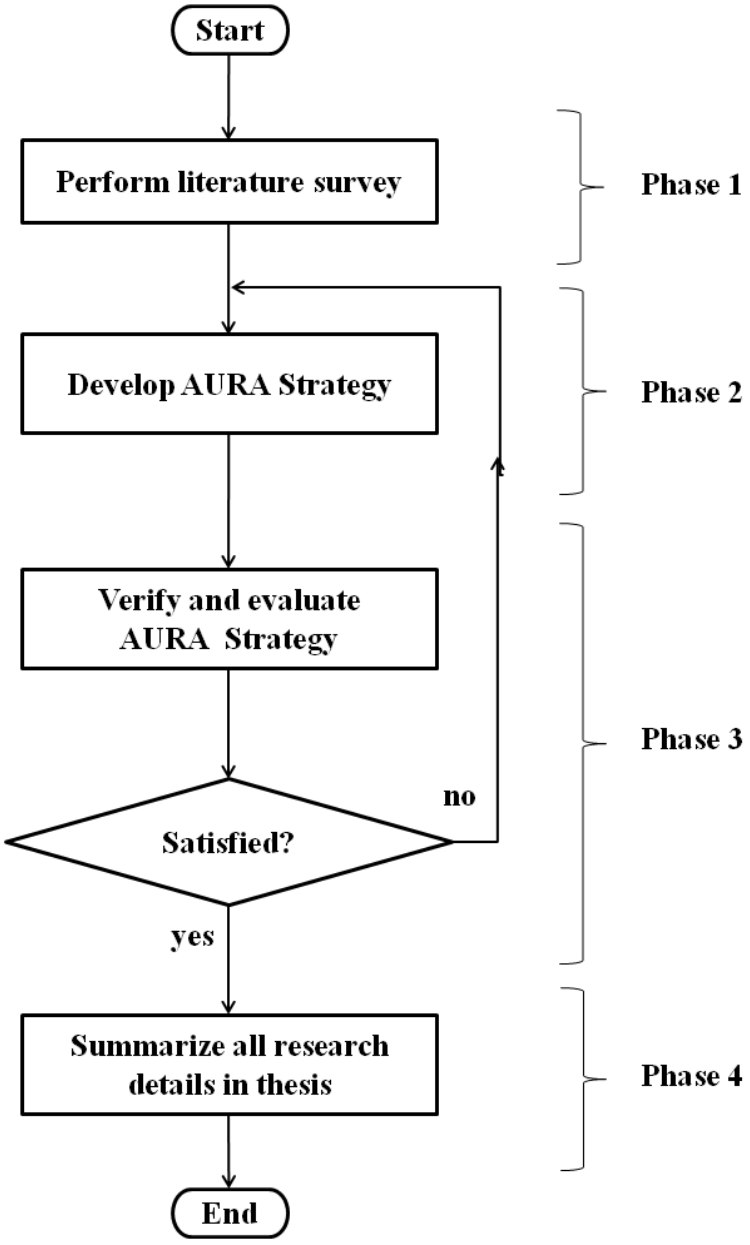

Figure 1.1 The Research Methodology Flow of AURA Strategy

1.6 Section Outlines

The rest of this thesis is structured into four chapters as follows.

Chapter 2 includes relevant literature review on interaction testing. For instance, there are three interaction testing approaches have been taken into account: uniform strength interaction testing, variable strength interaction testing and input-output based relationship interaction testing. For each approach, the mathematical background, overview as well as the recent published strategies are included in the discussion. Lastly, some literature on automation supports for interaction testing and flexibility of interaction test suite generation are also presented as one of the research scopes.

Chapter 3 illustrates the design of AURA strategy. In the chapter, all important design considerations of this strategy have been mentioned in order to develop AURA strategy as a test data generation tool that is supporting the automated input-output mapping and the flexible test suite generation capability. After that, the development of AURA strategy is detailed out and discussed. Subsequently, the corresponding development details of this strategy are also covered.

Chapter 4 describes the experimental setup and describes the findings of AURA strategy based on the results obtained. For instance, the flexible iteration control capability is characterized. Furthermore, this chapter also demonstrates and justifies the automated input-output mapping supports that proposed in AURA strategy. Additionally, the benchmarking inputs on AURA strategy have been executed and

compared against other published strategies in order to demonstrate the competitiveness of this strategy.

Lastly, the research work in this thesis is concluded as a whole in Chapter 5. Based on the findings obtained through experimental results together with the discussions been made in earlier chapters, a concrete conclusion has been made. In addition, the possible further work is also been discussed in this chapter.

CHAPTER 2

LITERATURE REVIEW

Before the design of AURA strategy is detailed out, there is a need to present relevant literature survey in this thesis. In this chapter, a comprehensive literature survey on interaction testing is presented. First of all, interaction testing is illustrated from mathematic perspective where covering arrays and its variants are used to express interaction test suites. Next, the insight overview of interaction testing approaches which included uniform strength, variable strength and input-output based relationship interaction testing have been included. Meanwhile, recent significant published strategies for each approach are also covered in the discussion here. Lastly, this chapter highlighted the issue of automation support in interaction testing as well as flexibility of interaction test suite generation.

2.1 Mathematical Background

Interaction testing test suite can be described in mathematical formulation forms. Based on different considerations in interaction testing (i.e. uniform strength, variable strength and input-output based relationship interaction testing), covering array and its variants (i.e. mixed-level covering array, variable strength covering array and input-output based relationship covering array) are often used to express the test suite mathematically (Hartman & Raskin, 2004).

In this section, these different types of covering arrays will be defined accordingly. Meanwhile, each of these covering arrays will be discussed with an example in order

to illustrate how these covering arrays can be used to abstract the interaction testing test suites.

2.1.1 Covering Array

Uniform strength interaction testing or *t*-way testing is an approach that used to systematically sample the set of inputs in such a way that all *t*-way combinations of inputs are included (Dubois, 2009). In brief, this approach will exhaustively explore *t*-strength interaction between input parameters in order to sample out the intended test cases.

Mathematically, *t*-way testing test suite can be abstracted to a covering array. Covering array is a combinatorial object that been extensively used to generate interaction test cases in software systems when all factors (parameters) have equal number of levels (options or values) (Myra, Colbourn, & Alan, 2003).

A covering array, CA (*N*; *t, k, v*), is an array with *N* rows and *k* columns that satisfies the criteria that each *t*-tuple occurs at least once within these rows (Dean, Charles, & Douglas, 2005). When *N* is unknown or unspecified, the notation CA (*t, k, v*) can be used (i.e. *t* is interaction strength, *k* is the number of factors and *v* is the number of options associated with each factor). For covering array, the value of *v* is the same for all *k* (Cemal, Myra, & Adam, 2006; Myra et al., 2003).

In order to illustrate the relationship between *t*-way testing test suite and covering array, consider the input base value set as shown in Figure 2.1(a), which consisted of

3 parameters (A, B and C) where each has 2 possible values (a0, a1, b0, b1, c0 and c1). As far as covering array is concerned, the values for all parameters concerned have to be the uniform (i.e. same number of values) in this example.

For the discussion here, it is desired to configure 2-way interaction (i.e. when *t* is fixed to 2) test cases. Therefore, the possible resultant test cases are generated as shown in Figure 2.1(b), which have included all 2-way combinations of input A, B and C (i.e. AB, AC and BC). It is noted that the resultant test cases number is 4.

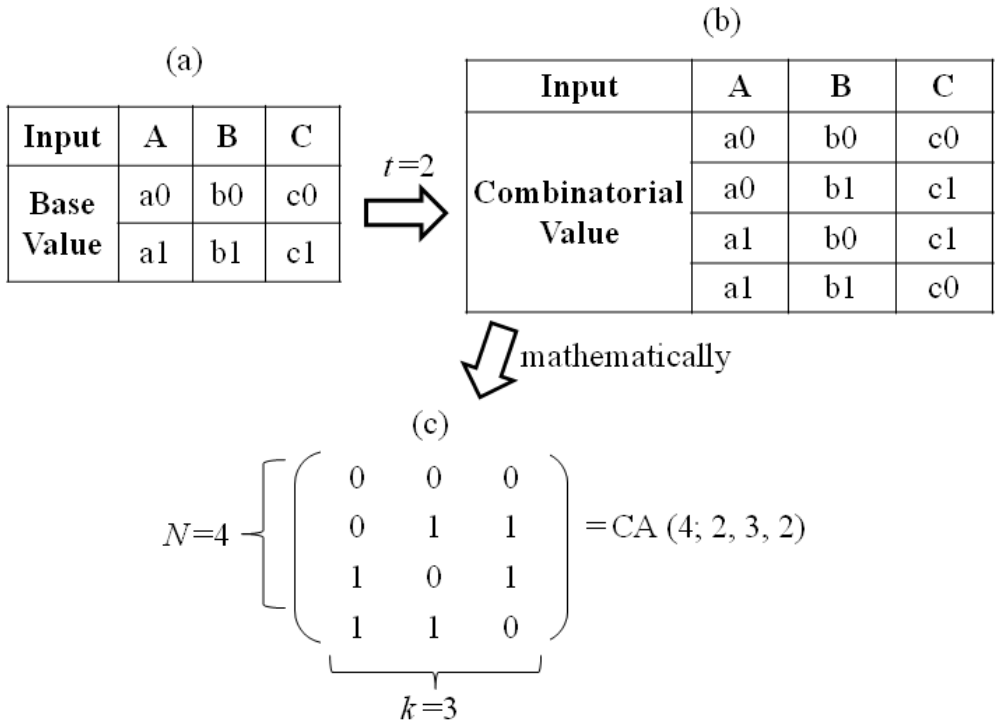

Figure 2.1 Illustration of Relationship between *t*-way Testing Test Suite and Covering Array: (a) Input Base Value Set; (b) Resultant Test Cases; (c) Covering Array

Given these test cases, covering array can be used to abstract them as shown in Figure 2.1(c). For CA (*N*; *t*, *k*, *v*), it is known that $N = 4$, $t = 2$, $k = 3$ and $v = 2$ in this

example. Therefore, the test cases can be termed as CA (4; 2, 3, 2). As a whole, Figure 2.1 depicts the illustration of relationship between *t*-way testing test suite and covering array.

2.1.2 Mixed-level Covering Array

As discussed earlier, covering array is used in *t*-way testing for inputs with uniform number of values. For certain inputs, the number of values for each parameter might not be ideally uniform. Therefore, mixed-level covering array has been proposed to overcome this limitation (Colbourn et al., 2006; Dean, Renee, & Charles, 2004).

Mixed-level covering array is a generalization of covering array that allows for different alphabet sizes for different rows. The mixed-level covering array is denoted as MCA ($N; t, k, (v_1, v_2, \ldots, v_k)$), an N x k array on v symbols, where $v = \sum_{i=1}^{k} v_i$, with the following properties (Bryce & Colbourn, 2006, 2007; Yan & Jian, 2006):

- Each column i ($1 \leq i \leq k$) contains only elements from a set S_i with $| S_i | = v_i$.
- The rows of each N x t sub-array cover all t-tuples of values from the t columns at least once.

In fact, a shorthand notation can be used to describe mixed-level covering array (also applicable for covering array, variable strength covering array and input-output relationship covering array) by combining the same v_i*'s* and representing this number as a superscript (Yan & Jian, 2006). For instance, three v_i*'s* each with two options is

written as 2^3. In this manner, an MCA (N; t, k, (v_1, v_2, …, v_k)) can also be written as an MCA(N; t, (s_1^{p1}, s_2^{p2}, …, s_r^{pr})) where $k = \sum_{i=1}^{r} p_i$.

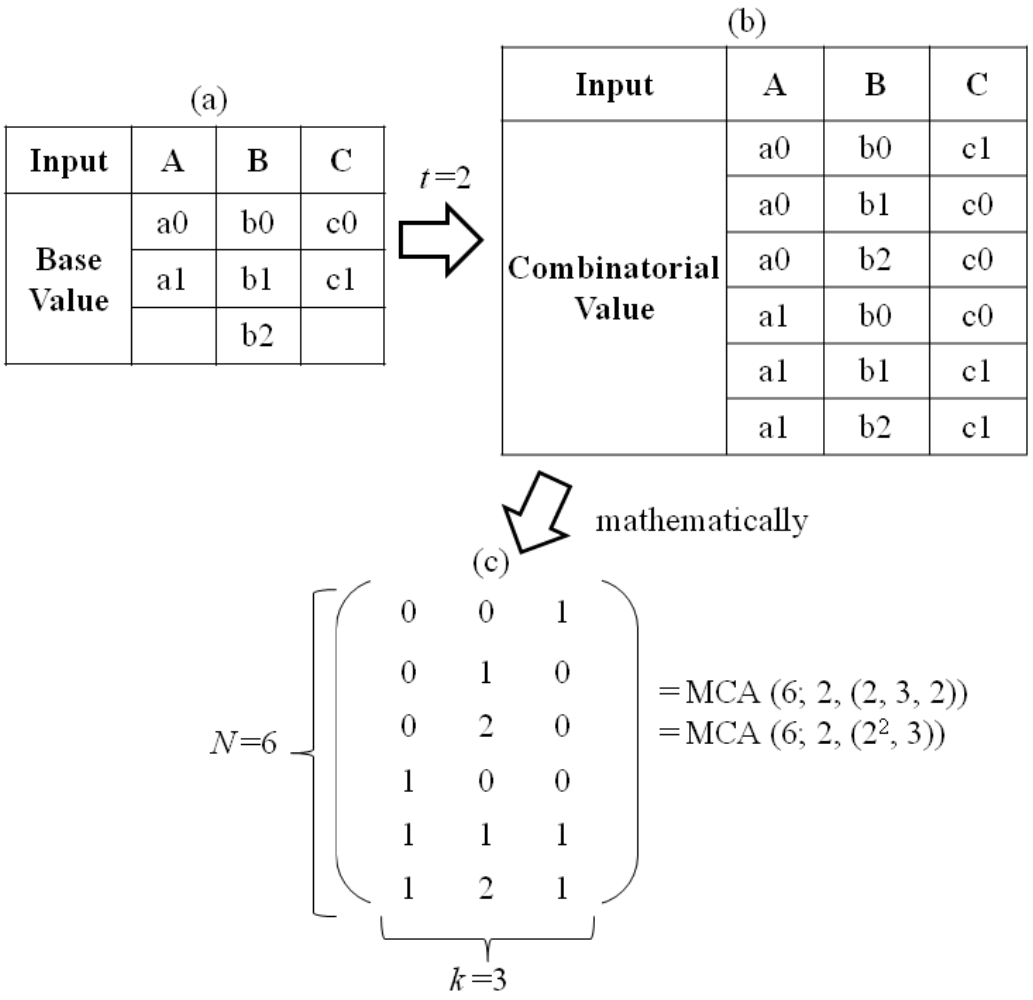

Figure 2.2 Illustration of Relationship between *t*-way Testing Test Suite and Mixed-level Covering Array: (a) Input Base Value Set; (b) Resultant Test Cases; (c) Mixed-level Covering Array

To illustrate the relationship between *t*-way testing test suite and mixed-level covering array, the input base value set as shown in Figure 2.2(a) has been taken into account, which consisted of 3 parameters (A, B and C) with non-uniform values. In

this case, parameters A and C each has 2 possible values (a0, a1, c0 and c1) whereas parameter B has 3 values (b0, b1 and b2).

For this example, assumed that it is desired to configure 2-way interaction (i.e. when *t* is fixed to 2) test cases. Therefore, the possible resultant test cases are generated as shown in Figure 2.2(b), which have included all 2-way combinations of input A, B and C (i.e. AB, AC and BC). It is noted that the resultant test cases number is 6 in the discussion here.

For these test cases, mixed-level covering array can be used to abstract them as shown in Figure 2.2(c). As $N = 6$, $t = 2$, $k = 3$, $v_1 = 2$, $v_2 = 3$, and $v_3 = 2$ for MCA (N; t, k, (v_1, v_2, …, v_k)), are known in this example. Therefore, these test cases can be termed as MCA (6; 2, (2, 3, 2)) or MCA (6; 2, (2^2, 3)). In general, Figure 2.2 shows the illustration of relationship between *t*-way testing test suite and mixed-level covering array.

2.1.3 Variable Strength Covering Array

In previous sections, the relationship between t-way testing test suite and both covering array and mixed-level covering array have been illustrated. Here, when variable strength interaction testing which allows for different strengths of coverage for subsets of parameters is concerned, variable strength covering array can be used to abstract this consideration into mathematical form (Cohen et al., 2003a; Mathur, 2008).

Variable strength covering array, denoted as VCA (N; t, (v_1, v_2, …, v_k), C), is an N x k mixed level covering array, of strength t containing C, a vector of covering arrays each of strength greater than t and defined on a subset of the k columns. Ordering of the columns in the representation of a VCA is important since the columns of the covering arrays in C are listed consecutively from left to right (Chen, Gu, Li, & Chen, 2009; Cohen et al., 2003b; Wang et al., 2008).

The input base value as shown in Figure 2.3(a) has been considered in order to illustrate the relationship between variable strength interaction testing test suite and variable strength covering array. The inputs consisted of 4 parameters (A, B, C and D) where each has 2 possible values (a0, a1, b0, b1, c0, c1, d0 and d1 respectively).

Here, it is wanted to configure 2-way interaction (i.e. when t is fixed to 2) test cases. Meanwhile, it is also desired to include 3-way interaction (i.e. when t is fixed to 3) for parameters A, B and C as far as variable strength interaction testing test suite is concerned.

Based on this set of t-way interactions, the possible resultant test cases are generated as shown in Figure 2.3(b), which have included all 2-way combinations of input A, B, C and D (i.e. AB, AC, AD, BC, BD and CD) and also 3-way combinations of input A, B and C (i.e. ABC). It is noted that the resultant test cases number is 8.

Given these test cases, variable strength covering array can be used to abstract them as shown in Figure 2.3(c). For VCA (N; t, (v_1, v_2, …, v_k), C), it is known that N = 8, t = 2, k = 4, v_1 = v_2 = v_3 = v_4 = 2 and C = CA (3, 2^3) in this case. Hence, it can be

termed as VCA (8; 2, (2, 2, 2, 2), CA (3, 2^3)) or VCA (8; 2, 2^4, CA (3, 2^3)). As a whole, Figure 2.3 depicts the illustration of relationship between variable strength interaction testing test suite and variable strength covering array.

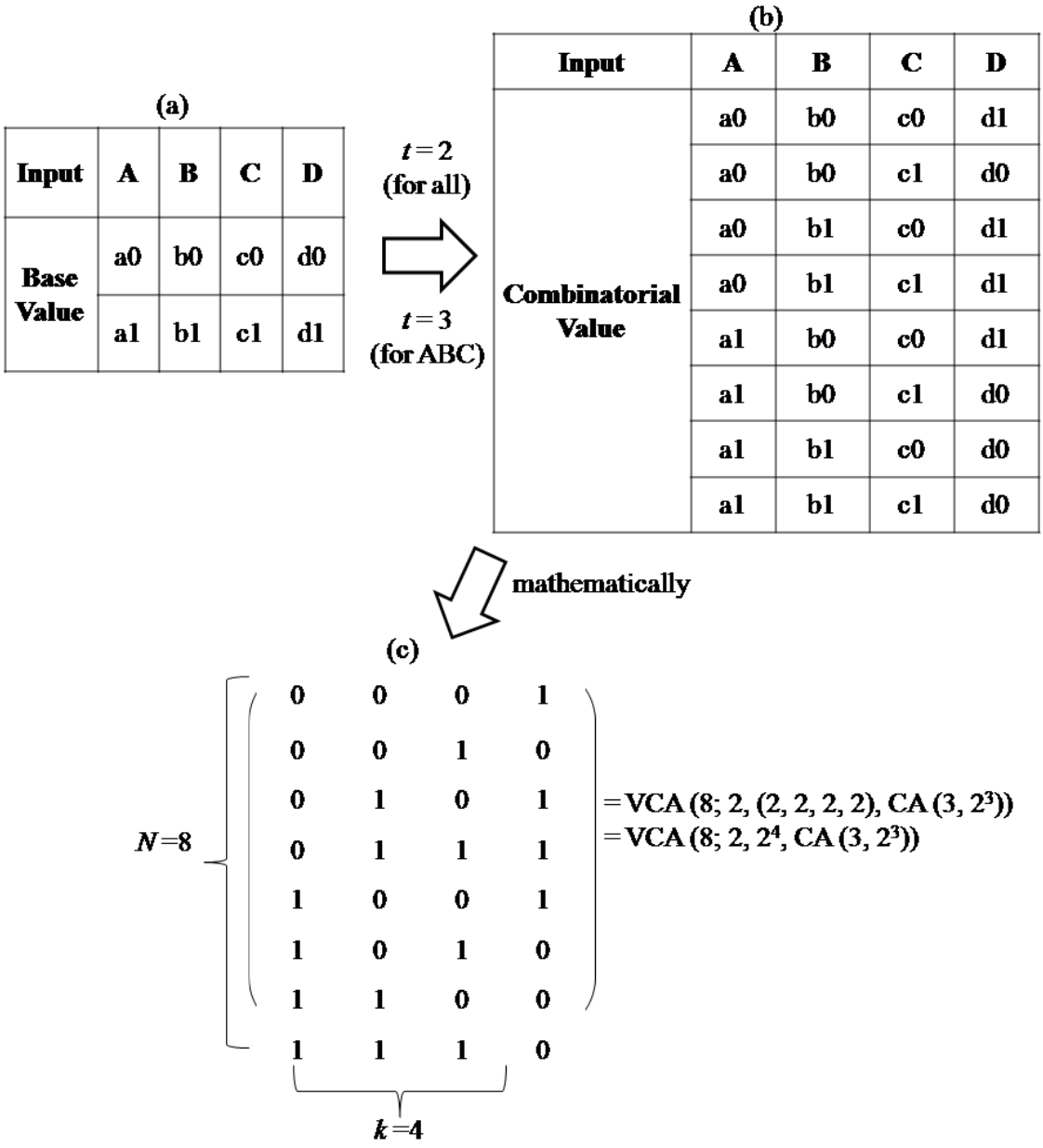

Figure 2.3 Illustration of Relationship between Variable Strength Interaction Testing Test Suite and Variable Strength Covering Array: (a) Input Base Value Set; (b) Resultant Test Cases; (c) Variable Strength Covering Array

2.1.4 Input-output Based Relationship Covering Array

Differing from *t*-way testing and variable strength interaction testing, input-output based relationship interaction testing needs not generate test cases to cover all *t*-way or a set of *t*-way combinations for a given set of input parameter. Indeed, this approach concerns on the actual interactions based on input-output relationship and abstracted as input-output based relationship covering array (Cheng, Dumitrescu, & Schroeder, 2003; Patrick et al., 2002).

Input-output based relationship covering array can be denoted as IOR (*N*; (v_1, v_2, …, v_k), *R*), an *N* x *k* mixed level covering array which covers interaction relationship, *R*, of a typical software SUT. *R* is consisted of *w* number of interaction coverage requirement, *r*, which specified the actual interactions for that SUT and is defined as $R = \{r_1, r_2, ..., r_w\}$. Each *r* indicates a set of inputs (factors) that are interacting and is constitute to a specified interaction coverage requirement (Patrick & Bogdan, 2000; Wang et al., 2007).

In order to illustrate the relationship between input-output based relationship interaction testing and input-output based relationship covering array, the input base value set as shown in Figure 2.4(a) has been considered. These inputs are consisted of 5 parameters (A, B, C, D and E) where each has 2 possible values (a0, a1, b0, b1, c0, c1, d0, d1, e0 and e1 respectively).

In this example, assuming that it is desired to configure test cases based on actual interactions AB and CDE. Therefore, Figure 2.4(b) depicts the resultant test cases

that included all possible combinations of both actual interactions AB and CDE. The resultant test cases number is 8.

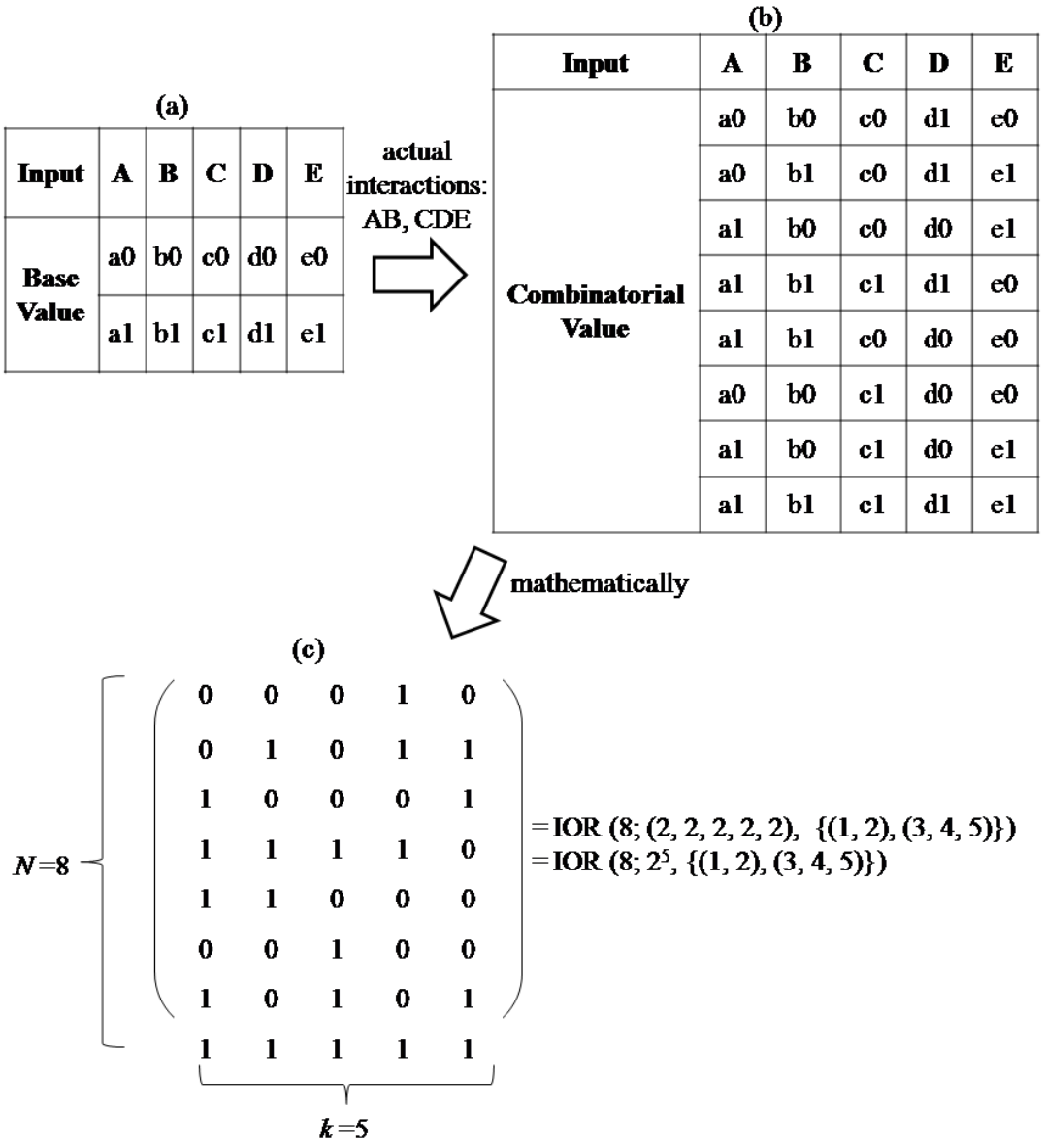

Figure 2.4 Illustration of Relationship between Input-output Based Relationship Interaction Testing Test Suite and Input-output Based Relationship Covering Array: (a) Input Base Value Set; (b) Resultant Test Cases; (c) Input-output Based Relationship Covering Array

For these test cases, input-output based relationship covering array can be used to abstract them as shown in Figure 2.4(c). For IOR (N; (v_1, v_2, …, v_k), R), these shown that $N = 8$, $v_1 = v_2 = v_3 = v_4 = v_5 = 2$ and $R = \{(f_1, f_2), (f_3, f_4, f_5)\}$. Thus, the test cases can be termed IOR (8; (2, 2, 2, 2, 2), $\{(f_1, f_2), (f_3, f_4, f_5)\}$) or IOR (8; 2^5, $\{(f_1, f_2), (f_3, f_4, f_5)\}$). In brief, the illustration of relationship between input-output based relationship interaction testing test suite and input-output based relationship covering array is depicted in Figure 2.4.

2.2 Overview and Approaches towards Interaction Testing

The main aim of interaction testing is to generate effective test data for detecting faults that due to interaction. As this strategy is found useful, there are different levels of interaction possibilities have been considered in the existing literatures, as part of interaction testing strategies which included uniform strength interaction testing, variable strength interaction testing as well as input-output based relationship interaction testing. As a result, many interaction strategies have been developed based on these approaches (Zabil et al., 2011).

In this section, the overview of all interaction testing approaches (i.e. uniform strength, variable strength and input-output based relationship interaction testing) will be explored. Also, the recent significant published strategies on these approaches are cited accordingly.

2.2.1 Uniform Strength Interaction Testing (*t*-way Testing)

The most fundamental interaction testing is uniform strength interaction testing (also known as *t*-way testing). *t*-way testing forms the basis of interaction testing which

works on detecting faults that caused by *t*-way interaction of parameters (where *t* indicates the interaction strength). According to empirical studies, the rationale for *t*-way testing stemmed from the fact that the number of parameters involved in software failures is relatively small (i.e. in the order of 2 to 6), in some classes of software (Burr & Young, 1998; Kimoto et al., 2008; Kuhn & Michael, 2002; Kuhn & Vadim, 2006; Kuhn et al., 2004).

Given a set of input, adopting all combinations is seemed impossible especially in a highly configurable software system. If *t* or fewer parameters interactions are known to cause fault or failure for software SUT, test suite can be constructed on some *t*-way combinations, then reduce to a smaller size of test suite (instead of considering all possible combinations) without decreasing the fault detection capability (Younis & Zamli, 2009b).

For example, an internet-based software system has been taken into account. The users may use a variety of browsers (i.e. Netscape, Internet Explorer and other). In addition, they may be using different operating systems (i.e. Windows, Macintosh and GNU/Linux), connection types (i.e. Local Area Network, Point-to-point Protocols and Integrated Services Digital Network) and printer configurations (i.e. Local, Networked and Screen) (Cohen et al., 2003a). The components of such software system have been summarized as shown in Table 2.1.

In order to completely test this software system, it is desired to consider all of the possible supported configurations (combinations). In this case, there are 3^4 or 81 combinations needed to test all possible interactions for the software system. Here, it

is assumed that the interaction faults for the software system are constituted of 2-way interactions (i.e. $t = 2$) among the components.

Table 2.1 Summary of Components for an Internet-based Software System (Cohen et al., 2003a)

Components			
Web Browser	**Operating System**	**Connection Type**	**Printer Configuration**
Netscape	Windows	Local Area Network	Local
Internet Explorer	Macintosh	Point-to-point Protocol	Networked
Other	GNU/Linux	Integrated Services Digital Network	Screen

Instead of adopting all 81 possible combinations, *t*-way testing can be used to reduce the number of combinations (test cases) into 9 based on 2-way interactions as depicted in Table 2.2 (Cohen et al., 2003a). In this case, these 9 test cases have covered all possible combinations for 2-way interactions among the components (i.e. Web Browser, Operating System, Connection Type and Printer Configuration). Hence, any interaction faults that due to 2-way interactions within these components can be discovered by adopting these test cases.

As the uniform strength interaction testing approach is found useful to reduce the test suite size and detect faults according to the specified interaction strength (i.e. *t*), numerous strategies have been proposed. For instance, the significant strategies

included: In-parameter-order-general (IPOG) strategy (Lei et al., 2007), Automatic Efficient Test Generator (AETG) strategy (Cohen et al., 1997; Cohen, Dalal, Kajla, & Patton, 1994; Cohen, Dalal, Parelius, & Patton, 1996), GTWay strategy (Zamli et al., 2011) as well as Test Configuration (TConfig) (Williams, 2002) strategy. All these strategies will be further elucidated in following subsections.

Table 2.2 The Resultant Test Cases for 2-way Interactions (Cohen et al., 2003a)

Test Number	**Browser**	**Operating System**	**Connection Type**	**Printer Configuration**
1	Netscape	Windows	Local Area Network	Local
2	Netscape	GNU/Linux	Integrated Services Digital Network	Networked
3	Netscape	Macintosh	Point-to-point Protocol	Screen
4	Internet Explorer	Windows	Integrated Services Digital Network	Screen
5	Internet Explorer	Macintosh	Local Area Network	Networked
6	Internet Explorer	GNU/Linux	Point-to-point Protocol	Local
7	Other	Windows	Point-to-point Protocol	Networked
8	Other	GNU/Linux	Local Area Network	Screen
9	Other	Macintosh	Integrated Services Digital Network	Local