Postprint of article in *Proceedings of the 3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (*SNPD 2002*), International Association for Computer and Information Science, Mt. Pleasant, MI, pp. 99–106 (2002)

An Experimental Analysis of the Identification of Categories and Choices from Specifications*†

T. Y. Chen
School of Information Technology
Swinburne University of Technology
Hawthorn 3122, Australia
tychen@it.swin.edu.au

Pak-Lok Poon
Department of Accountancy
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
acplpoon@inet.polyu.edu.hk

Sau-Fun Tang
School of Business and Administration
The Open University of Hong Kong
Ho Man Tin, Kowloon
Hong Kong
stang@ouhk.edu.hk

T. H. Tse‡
Department of Computer Science and Information Systems
The University of Hong Kong
Pokfulam Road, Hong Kong
tse@csis.hku.hk

Abstract

The category-partition method and classification-tree method both help construct test cases from specifications. To achieve this end, a set of categories and the associated choices (also known as classifications and the associated classes) have to be identified. However, the identification process is often done in an ad hoc manner. We have conducted a case study to examine the common mistakes made by software testers during this process. The result of our study will facilitate researchers and practitioners in the development of systematic identification methods.

† This work is supported in part by grants of the Research Grants Council of Hong Kong (Project Nos. CityU 1048/01E and HKU 7029/01E) and a research and conference grant of The University of Hong Kong.

‡ **Contact author.**

1. Introduction

Various people [1, 2, 3] have maintained that software testing is a labor-intensive and expensive process, which may account for 50% of the total project cost. Among the various activities in software testing, test case construction is an important one, because it determines the scope and hence the quality of the tests [4]. This has motivated researchers and practitioners to explore effective ways of constructing test cases.

Most of the studies have focused on the *white-box* approach, where test cases are derived from the source code [4, 5, 6]. The *black-box* approach, on the other hand, aims at constructing test cases directly from specifications without reference to the source code. Most proposals in the black-box approach involve the construction of test cases based on formal specifications [7, 8]. However, real-life commercial specifications are usually written in an informal manner, such as in narrative English or graphic languages. Hence, the applicability of proposals based on formal languages would be rather limited. The category-partition method (CPM) [9, 10] and the classification-tree method (CTM) [11, 12, 13, 14] are two exceptions to this tendency of relying solely on formal languages. CPM and CTM can be applied to both formal and informal specifications. These two methods will be outlined below.

Readers may like to note that "categories", "choices", and "test frames" in CPM are equivalent

to "classifications", "classes", and "test cases" in CTM. Hence, we shall use the terms "categories" and "classifications" and the terms "choices" and "classes" interchangeably for the rest of the paper. Because of the popularity of the term "test cases", we shall use the latter throughout this paper.

(*a*) Category-Partition Method (CPM)

Given a specification, CPM involves the identification of (i) categories and their associated choices, and (ii) the constraints among choices.

A *category* is defined as "a major property or characteristic of a parameter or an environment condition" [10]. *Choices* are defined as "all the different kinds of values that are possible for the category" [10]. Consider, for instance, a SQRT program which reads an input file F that contains two real numbers m and n, and outputs the value of $\sqrt{m+n}$. In this situation, [Status of F] and $[m+n]$ are two possible categories, defined with respect to an environment condition and an input parameter of SQRT, respectively. The first category [Status of F] has three associated choices, namely |Status of F: Does Not Exist|, |Status of F: Exists but Empty|, and |Status of F: Exists and Non-Empty|.[1] On the other hand, the second category $[m+n]$ has $|m+n:<0|$ and $|m+n:\geq 0|$ as its associated choices. An example of constraint between choices is that choice |Status of F: Does Not Exist| cannot be combined with choice $|m+n:\geq 0|$ in any test case.

After defining all the categories, choices, and constraints among choices, test cases can be formed using an associated generator tool. This tool generates all the possible combinations of choices so that each combination corresponds to a test case. During the generation process, it suppresses as far as possible all combinations of choices that are incompatible with the defined constraints, such as the combination |Status of F: Does Not Exist| and $|m+n:\geq 0|$.

For example, using the associated generator tool, the following four test cases will be generated for SQRT:

(i) |Status of F: Does Not Exist|

(ii) |Status of F: Exists but Empty|

(iii) |Status of F: Exists and Non-Empty|, $|m+n:<0|$

(iv) |Status of F: Exists and Non-Empty|, $|m+n:\geq 0|$

(*b*) Classification-Tree Method (CTM)

This method helps generate test cases via the construction of classification trees. Basically, a *classification tree* organizes classifications and their associated classes at alternate levels in a tree structure. In essence, this tree structure captures the constraints among classifications.

A major difference between CTM and CPM is that the former constructs test cases based on constraints defined at the classification-level, whereas the latter achieves this from constraints at the choice-level. For the SQRT program, the four test cases generated by CPM, as listed in (a) above, will also be generated by CTM. Readers may refer to [11, 12, 13, 14] for details.

Both CPM and CTM require a set of categories and choices (or classifications and classes) to be identified from the specification. We observe that this identification process is often done in an ad hoc manner. In the absence of a systematic method, it is doubtful whether there will be any quality assurance on the defined categories, choices, and the resulting test cases, particularly when the specification is informal. This paper reports on an experimental study to assess the situation, with a view to helping researchers and practitioners develop better methodologies.

The rest of the paper is organized as follows. Section 2 describes an experiment to study the mistakes commonly made during an ad hoc identification of categories and choices from informal specifications. Section 3 identifies the various types of poorly defined category and choice. Section 4 describes the results and observations from the experimental study. Finally, Section 5 concludes the paper.

2. An experimental study

We believe it is useful to investigate into the mistakes likely to be made when categories and choices are identified by ad hoc approaches. This can facilitate the development of more effective identification methodologies. In addition, we can also verify the effectiveness of these methodologies in terms of the ability to eliminate or reduce such mistakes.

[1] Throughout the paper, categories are enclosed by square brackets [ ] while choices are enclosed by vertical bars | |. Furthermore, the notation $|X:x|$ denotes choice x in category X.

Because of this, we have conducted an experimental study involving a group of 48 computer-science final year students. We asked the students to:

(a) Identify as many categories and associated choices as possible from two given specifications denoted by ***SALES*** and ***REWARDS***.

(b) For every identified category or choice, state the rationale of its identification.

The ***SALES*** specification is related to the credit sales of goods to customers in a trading company, based on their credit limits authorized by the company. On the other hand, the ***REWARDS*** specification is included below for the purpose of discussion. It should be noted that ***REWARDS*** is slightly more complex than ***SALES***.

The *REWARDS* Specification:

Number-One Bank is a US-based bank that issues credit cards to approved customers. These customers will earn certain reward points for every dollar charged to their cards, regardless of the goods purchased. The number of reward points earned by the customers determines the types of benefit to which they are entitled, such as free air tickets, free shopping vouchers, and so on.

For each purchase, the program shall accept the transaction details together with the various information of the credit card in order to determine whether this transaction should be approved. If the purchase is approved, the program will calculate the number of reward points. Further details are given below. [Details for calculating the number of reward points are not included here.]

(*a*) Credit card details

(i) Two types of credit card are issued by Number-One Bank, namely corporate and personal.

(ii) Personal credit cards issued to principal holders are called *principal cards*, while those issued to other persons sponsored by principal holders are called *additional cards*.

(iii) Corporate and principal cardholders must have a bank account with Number-One Bank, while additional cardholders need not have such an account.

(iv) There are three classes for corporate and personal credit cards, namely diamond, gold, and classic.

(v) The credit limit of a classic card is \$1000 and that for a gold card is \$4000. On the other hand, the credit limit of a diamond card can be either \$4000 or \$8000.

(vi) For personal credit cards, the class and the credit limit of any associated additional card are identical to those of the corresponding principal card.

(*b*) Purchase details

(i) For purchase of air tickets:

Any air ticket must be of First, Business, or Economy class.

"Extra reward" points can be earned for tickets of any class purchased from Holiday Airlines, because of a special agreement with Number-One Bank.

"Extra reward" points can be earned for First or Business class tickets purchased from any other airlines.

(ii) For the purchase of goods other than air tickets, "extra reward" points can be earned if such goods are acquired from bonus shops.

(iii) Purchase can be made within the USA or overseas. The earning of reward points is not affected by the geographical location of the purchase. For any purchase in a currency other than US\$, the purchase amount will be converted to US\$ before the calculation of the number of reward points, based on the relevant exchange rate applicable at the time of purchase. In any case, the purchase amount will be maintained with two places after decimal.

(*c*) Conditions for rejecting a purchase

(i) The credit card has already expired.

(ii) The credit card has been suspended for some reason. For example, the cardholder may not have settled the bill promptly, or the card may have been reported to be lost.

(*iii*) The total of the current purchase amount and the cumulative balance exceeds the credit limit of the card. *Cumulative balance* is defined as the total purchase amount of all previously approved transactions that have not yet been settled, via the principal card as well as all the associated additional cards.

3. Poorly defined categories and choices

Although the meaning of "test cases" is intuitively understood, it is often subject to different interpretations. We would like to define it formally in order to avoid ambiguities when discussing other important concepts in this section.

Definition 1 (Test Cases)
*A **test case** tc is a set of choices such that, if a single element is selected from each choice, a standalone input will be formed.*

Example 1 (Test Cases)
Refer to the SQRT program in Section 1. Consider the set

$tc = \{|\text{Status of } F\text{: Exists and Non-Empty}|, |m+n : \geq 0|\}$.

If we select a single element from each choice in *tc*, such as

Status of F = Exists and Non-Empty, $m+n = 123.45$,

then the result will form a standalone input to the SQRT program. Thus, the set *tc* is a test case of SQRT. ■

Before discussing the results of our experiment, we have to introduce the following definitions first. Note that all examples used for the rest of this paper refer to ***REWARDS*** in Section 2.

Definition 2 (Validity of Choices)
*Given a category $[X]$, any choice $|X : x|$ is said to be **valid** if all the values of $|X : x|$ fall within the input domain. Otherwise, $|X : x|$ is an **invalid choice** and $[X]$ is called a **category with invalid choices**.*

Example 2 (Validity of Choices)
Consider the category [Class of Seat] and its associated choices |Class of Seat: First|, |Class of Seat: Business|, |Class of Seat: Economy|, and |Class of Seat: Others|. In this situation, |Class of Seat: Others| is an invalid choice. This is because, according to ***REWARDS***, any ticket must be of First, Business, or Economy class. As a result, [Class of Seat] is a category with an invalid choice. ■

Definition 3 (Omission of Choices)
*Any category $[X]$ is said to have **missing choices** if there exists some element in the input domain that is associated with $[X]$ but is not related to any of its choices.*

Example 3 (Omission of Choices)
Consider Example 2 again. Suppose the category [Class of Seat] is defined with only two choices, |Class of Seat: First| and |Class of Seat: Business|. Then obviously [Class of Seat] is a category with a missing choice. ■

Definition 4 (Overlapping of Choices)
*Given a category $[X]$, two or more of its distinct choices are said to be **overlapping** if there exists an element y in the input domain such that y is associated with all of these choices. If this happens, $[X]$ is called a **category with overlapping choices**.*

Example 4 (Overlapping of Choices)
Consider the category $\big[$Credit Limit − Cumulative Balance (US$)$\big]$ with |Credit Limit − Cumulative Balance (US$): ≤ 0| and |Credit Limit − Cumulative Balance (US$): ≥ 0| as its associated choices. In this situation, |Credit Limit − Cumulative Balance (US$): ≤ 0| and |Credit Limit − Cumulative Balance (US$): ≥ 0| are overlapping choices, and $\big[$Credit Limit − Cumulative Balance (US$)$\big]$ is a category with overlapping choices. ■

Before we proceed further, we have to introduce the notions of function models and function rules [15, 16]. A *function model* represents the behavior of the system at an abstract level, so that the software engineer and the user can agree on what the system has to do without the need for programming details. The mapping between a given set of inputs and its corresponding outputs is expressed by means of a *function rule*. This rule must state clearly the *preconditions* of the inputs (that is, conditions under which the inputs are valid) and how the outputs are related to the acceptable inputs. In the function model, we assume that the system is *deterministic*, or in other

words, one and only one output is possible for a given input.

Definition 5 (Composition of Choices)
Given any set of choices B', let $TC(B')$ denote the set of test cases such that $B' \subseteq tc$ for any $tc \in TC(B')$.

Given a category $[X]$ and any of its choices $|X : x_1|$, suppose we can find two other choices $|X : x_2|$ and $|X : x_3|$ of the same category satisfying the following conditions:

(*a*) *$|X : x_1|$, $|X : x_2|$, and $|X : x_3|$ are distinct from one another,*

(*b*) *$TC(\{|X : x_1|\}) = TC(\{|X : x_2|\}) \cup TC(\{|X : x_3|\})$, and*

(*c*) *there exists a set of choices B and some test cases $tc' \in TC(\{|X : x_2|\} \cup B)$ and $tc'' \in TC(\{|X : x_3|\} \cup B)$ such that an element from tc' and one from tc'' will execute different function rules in the function model of the specification.*

*Then $|X : x_1|$ is a **composite choice** and $[X]$ is called a **category with composite choices**.*

Example 5 (Composition of Choices)
Suppose the category [Status of Customer Master File] is defined with |Status of Customer Master File: Valid| and |Status of Customer Master File: Invalid| as its associated choices, such that

(*a*) |Status of Customer Master File: Valid| covers the scenario where the Customer Master File exists and is non-empty.

(*b*) |Status of Customer Master File: Invalid| covers both of the following scenarios:

  (*i*) The Customer Master File does not exist.

  (*ii*) The Customer Master File exists but is empty.

Suppose, further, that [Status of Customer Master File] and |Status of Customer Master File: Invalid| correspond to $[X]$ and $|X : x_1|$ in Definition 5, respectively. For category [Status of Customer Master File], we can define two other choices |Status of Customer Master File: Does Not Exist| and |Status of Customer Master File: Exists but Empty| such that the former covers scenario $(b)(i)$ above and the latter covers scenario $(b)(ii)$. Note that |Status of Customer Master File: Does Not Exist| and |Status of Customer Master File: Exists but Empty| correspond to $|X : x_2|$ and $|X : x_3|$ in Definition 5, respectively. In this situation, we have $TC(\{|\text{Status of Customer Master File: Invalid}|\}) = TC(\{|\text{Status of Customer Master File: Does Not Exist}|\}) \cup TC(\{|\text{Status of Customer Master File: Exists but Empty}|\})$.

Let $B = \{|\text{Class of Seat: Economy}|, |\text{Airline Company: Holiday}|, |\text{Place of Purchase: USA}|, |\text{Purchase Amount: } \leq \$1000|\}$. Suppose $tc' \in TC(\{|\text{Status of Customer Master File: Does Not Exist}|\} \cup B)$ and $tc'' \in TC(\{|\text{Status of Customer Master File: Exists but Empty}|\} \cup B)$, and that the program for the specification ***REWARDS*** will handle tc' and tc'' differently. In other words, an element from tc' and one from tc'' will execute different function rules in the function model for ***REWARDS***. Then, |Status of Customer Master File: Invalid| is a composite choice and [Status of Customer Master File] is a category with a composite choice. |Status of Customer Master File: Invalid| should preferably be replaced by two different choices |Status of Customer Master File: Does Not Exist| and |Status of Customer Master File: Exists but Empty|. ■

Definition 6
(Types of Category with Composite Choices)
*Let $[X]$ be a category with composite choices. It is said to be a category with **forced** composite choices if the definition of the category entails the introduction of composite choices. Otherwise, it is said to be a category with **avoidable** composite choices.*

Example 6
(Types of Category with Composite Choices)
Suppose that [Validity of Customer Master File] is the only category defined to reflect the environment condition of the Customer Master File. The choices |Validity of Customer Master File: Valid| and |Validity of Customer Master File: Invalid| naturally result from the definition of [Validity of Customer Master File]. Furthermore, |Validity of Customer Master File: Invalid| is a composite choice because it may refer to either of the following scenarios:

(*a*) The Customer Master File does not exist.

(*b*) The Customer Master File exists but is empty.

Thus, [Validity of Customer Master File] is a category with a forced composite choice.

Now, suppose that [Validity of Customer Master File] is replaced by another category [Status of Customer Master File], with |Status of Customer Master File: Valid| and |Status of Customer Master File: Invalid| as its associated choices. Obviously, |Status of Customer Master File: Invalid| is a composite choice. However, the introduction

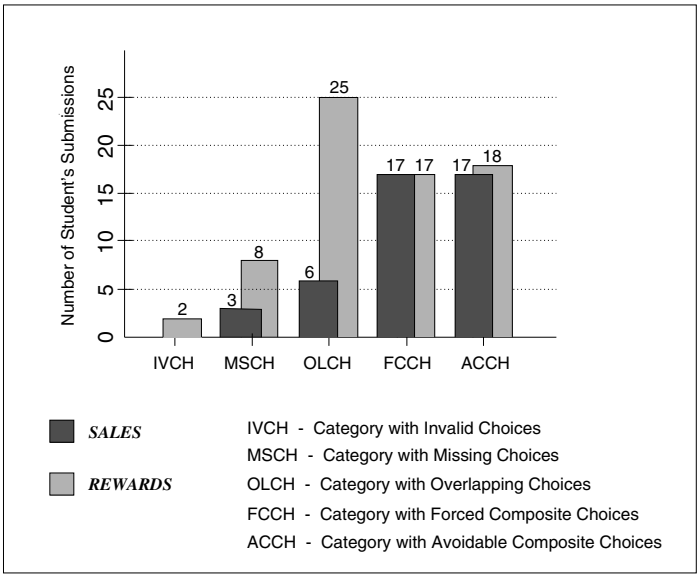

Figure 1. Number of submissions with respect to types of poorly defined category

of this composite choice is not directly due to the category [Status of Customer Master File]. Given this category, the software tester could have defined two choices |Status of Customer Master File: Does Not Exist| and |Status of Customer Master File: Exists but Empty| to replace |Status of Customer Master File: Invalid|. In this case, [Status of Customer Master File] is a category with an avoidable composite choice. ■

Definition 7 (Poorly Defined Categories)
Any category $[X]$ *is said to be* ***poorly defined*** *if one or more of the following conditions are satisfied:*

(a) $[X]$ *is a category with invalid choices.*

(b) $[X]$ *is a category with missing choices.*

(c) $[X]$ *is a category with overlapping choices.*

(d) $[X]$ *is a category with composite choices.*

4. Observations and discussion

From the submissions in our experiment, we find that there are a total of 265 and 475 categories identified for the input domain of ***SALES*** and ***REWARDS***, respectively. Among the 265 categories defined for ***SALES***, 43 (16.2%) are poorly defined. For the 475 categories defined for ***REWARDS***, 79 (16.6%) are poorly defined. Our results confirm that problems may arise when categories and choices are identified in an ad hoc manner, especially by inexperienced users. These problems may be due to a lack of domain knowledge or a misunderstanding of categories and choices.

Figure 1 shows the number of submissions with respect to the types of poorly defined category. Note that one or more types of poorly defined category may be encountered in the same submission. Consider also Figure 2, which shows the total number of each type of poorly defined category identified in the experiment. From these two figures, we observe that:

(a) In both figures, the frequency distributions of poorly defined categories are consistent between ***SALES*** and ***REWARDS***, with the exception of categories with overlapping choices (OLCH).

(b) Among the various types of poorly defined category, categories with composite choices (FCCH and ACCH) are more likely to be defined. On the other hand, categories with invalid choices (IVCH) and missing choices (MSCH) are relatively less likely to be defined. A plausible reason is that categories with composite choices (FCCH and ACCH) can only be detected and avoided through additional domain knowledge.

We further note that all categories with forced composite choices (FCCH) defined by students involve two or more input parameters of a program. An example is the category [Cumulative Balance > Credit Limit (US$)] with |Cumulative Balance > Credit Limit (US$): Yes| and |Cumulative Balance > Credit Limit (US$): No| as its associated

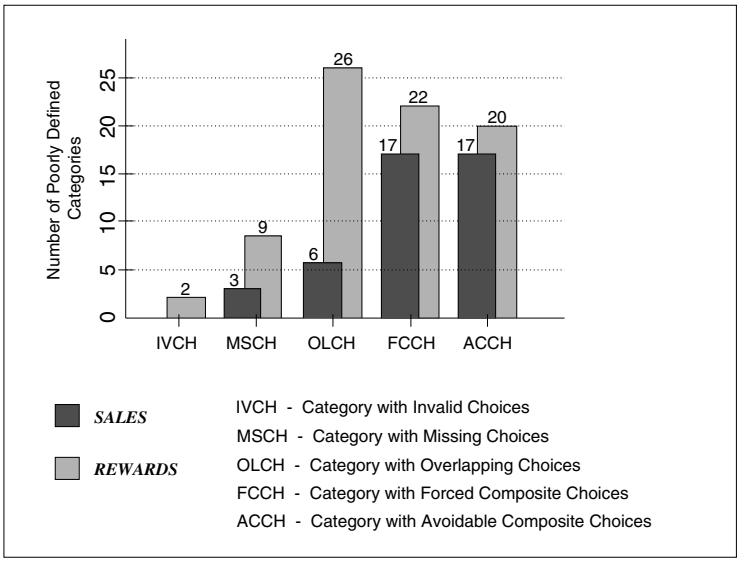

Figure 2. Number of poorly defined categories identified by students

choices. For this category, |Cumulative Balance $>$ Credit Limit (US$): No| is a composite choice. Hence, whenever a category involves two or more input parameters, special attention should be given to avoid composite choices.

5. Conclusion

We have described an experiment to identify and analyze common problems encountered when using ad hoc approaches to define categories and choices from informal specifications. To facilitate the analysis of experimental results, we have classified poorly defined categories into four types:

- Categories with invalid choices,
- Categories with missing choices,
- Categories with overlapping choices, and
- Categories with composite choices.

Our results confirm that poorly defined categories may arise when the identification of categories and choices is performed in an ad hoc manner, especially by inexperienced users. Furthermore, categories with composite choices are more likely to be defined than categories with invalid and missing choices.

The contributions of our experiment are twofold. First, by identifying the various types of poorly defined category and choice, and highlighting them to inexperienced users, testers will be less likely to define these categories and choices. Secondly, information of poorly defined categories and choices will help researchers and practitioners develop identification methodologies and evaluate them in terms of their effectiveness in screening out unwarranted categories and choices.

References

[1] R. Ferguson and B. Korel, "The Chaining Approach for Software Test Data Generation", *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 1, pp. 63–86, 1996.

[2] B. Korel, "Automated Test Data Generation for Programs with Procedures", in *Proceedings of the 1996 International Symposium on Software Testing and Analysis* (*ISSTA '96*), New York: ACM Press, 1996, pp. 209–215.

[3] J. Sanders and E. Curran, *Software Quality: A Framework for Success in Software Development and Support*, Wokingham, UK: Addison Wesley, 1994.

[4] T. Chusho, "Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing", *IEEE Transactions on Software Engineering*, vol. SE-13, no. 5, pp. 509–517, 1987.

[5] S. C. Ntafos, "A Comparison of Some Structural Testing Strategies", *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 868–874, 1988.

[6] E. J. Weyuker, "The Evaluation of Program-Based Software Test Data Adequacy Criteria", *Communications of the ACM*, vol. 31, no. 6, pp. 668–675, 1988.

[7] T. Y. Chen and M. F. Lau, "Test Case Selection Strategies Based on Boolean Specifications", *Software Testing, Verification and Reliability*, vol. 11, no. 3, pp. 165–180, 2001.

[8] E. J. Weyuker, T. Goradia, and A. Singh, "Automatically Generating Test Data from a Boolean Specification", *IEEE Transactions on Software Engineering*, vol. 20, no. 5, pp. 353–363, 1994.

[9] M. J. Balcer, W. M. Hasling, and T. J. Ostrand, "Automatic Generation of Test Scripts from Formal Test Specifications", in *Proceedings of the 3rd ACM Annual Symposium on Software Testing, Analysis, and Verification* (*TAV '89*), New York: ACM Press, 1989, pp. 210–218.

[10] T. J. Ostrand and M. J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests", *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[11] T. Y. Chen and P. L. Poon, "Construction of Classification Trees Via the Classification-Hierarchy Table", *Information and Software Technology*, vol. 39, no. 13, pp. 889–896, 1997.

[12] T. Y. Chen, P. L. Poon, and T. H. Tse, "An Integrated Classification-Tree Methodology for Test Case Generation", *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 6, pp. 647–679, 2000.

[13] M. Grochtmann and K. Grimm, "Classification Trees for Partition Testing", *Software Testing, Verification and Reliability*, vol. 3, no. 2, pp. 63–82, 1993.

[14] H. Singh, M. Conrad, and S. Sadeghipour, "Test Case Design Based on Z and the Classification-Tree Method", in *Proceedings of the 1st IEEE International Conference on Formal Engineering Methods* (*ICFEM '97*), Los Alamitos, California: IEEE Computer Society Press, 1997, pp. 81–90.

[15] A. Paradkar, K.-C. Tai, and M. A. Vouk, "Specification-Based Testing Using Cause-Effect Graphs", *Annals of Software Engineering*, vol. 4, pp. 133–157, 1997.

[16] J. B. Wordsworth, *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*, Wokingham, UK: Addison Wesley, 1992.