Postprint of article in *Information and Software Technology* **46** (13): 887–898 (2004)

On the identification of categories and choices for specification-based test case generation ☆,☆☆

T.Y. Chen[a], Pak-Lok Poon[b], Sau-Fun Tang[a], T.H. Tse[c,*]

[a] *Centre for Software Analysis and Testing, Swinburne University of Technology, Hawthorn 3122, Australia*
[b] *School of Accounting and Finance, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[c] *Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong*

Abstract

The category-partition method and the classification-tree method help construct test cases from specifications. In both methods, an early step is to identify a set of categories (or classifications) and choices (or classes). This is often performed in an ad hoc manner due to the absence of systematic techniques. In this paper, we report and discuss three empirical studies to investigate the common mistakes made by software testers in such an ad hoc approach. The empirical studies serve three purposes: (a) to make the knowledge of common mistakes known to other testers so that they can avoid repeating the same mistakes, (b) to facilitate researchers and practitioners develop systematic identification techniques, and (c) to provide a means of measuring the effectiveness of newly developed identification techniques. Based on the results of our studies, we also formulate a checklist to help testers detect such mistakes.

Key words: Category-partition method; Choice relation framework; Classification-tree method; Specification-based testing; Test frame

1. Introduction

There are two broad categories of software testing: white-box and black-box approaches. Any test case generation method or technique falls under one or both of these approaches. In *white-box* (or *implementation-based*) *testing*, test cases are generated according to information derived from the source code of the program under test. White-box testing typically requires the coverage of certain aspects of the program structures. Control flow testing [12, 15], data flow testing [10, 12, 15], and domain testing [7, 13] are some examples.

In contrast to white-box testing, *black-box* (or *specification-based*) *testing* generates test cases without the knowledge of the internal structure of the program. In most black-box testing methods, test cases are generated according to the specifications. These specifications can be written in a formal language such as Z [14], or informally such as in narrative English. Black-box testing methods have been developed both for formal and informal specifications. Chen et al. [4] argue that many real-life commercial specifications are informal, and hence the applicability of black-box testing based on formal specifications is rather restricted. In this respect, the category-partition method (CPM) [1, 6, 9] and the classification-tree method (CTM) [5, 8] are considered to be very useful, because they can also be applied to informal specifications.

In CPM and CTM, an early step is to identify a set of categories (also known as classifications) and their

☆This is an extended and revised version of [4].

☆☆This research is supported in part by a grant of the Research Grants Council of Hong Kong (Project No. HKU 7029/01E), an Australian Research Council Discovery Grant (Project No. DP 0345147), and an Internal Competitive Research Grant of The Hong Kong Polytechnic University (Project No. A-PC30).

*Corresponding author. Tel.: +852 2859 2183; fax: +852 2858 4141.

Email addresses: tchen@it.swin.edu.au (T.Y. Chen), afplpoon@inet.polyu.edu.hk (Pak-Lok Poon), sftang@hkbu.edu.hk (Sau-Fun Tang), thtse@cs.hku.hk (T.H. Tse)

associated choices (also known as classes), which in turn forms the basis for the subsequent generation of test cases.[1] Obviously, the chance of detecting faults from the software depends on the comprehensiveness of the generated test cases, which in turn depends on the comprehensiveness of the identified categories and choices. If, for example, a valid choice is missing, then any fault associated with this choice may not be detected. We observe that neither the original developers of CPM/CTM nor follow-up researchers have proposed a systematic method for identifying categories and choices from informal specifications. As a result, this identification process is often performed in an ad hoc manner. The quality of the resulting test cases may be in question, especially when the specification is informal.

Motivated by this problem, we have conducted three empirical studies on the identification of categories and choices from informal specifications.[2] Our primary objective is to find out the common mistakes made by software testers when identifying categories and choices in the absence of a systematic process. Our studies have three contributions: (a) to reduce the chance of repeating these mistakes by making them known to testers, (b) to shed light on the development of systematic techniques for identifying categories and choices, and (c) to provide a means of measuring the effectiveness of newly developed identification techniques in terms of their ability to avoid or reduce mistakes. Based on the observations of the studies, we also formulate a checklist to help testers detect such mistakes.

The rest of this paper is organized as follows. Section 2 outlines the background concepts of CPM and CTM. Section 3 gives an overview of function models and function rules. Section 4 discusses important terminology and definitions, particularly on various types of problematic category and choice. Section 5 describes the settings of our empirical studies. Section 6 reports and discusses the results and observations of our studies, and recommends an identification checklist to help testers detect problematic categories and choices. Section 7 discusses the validity of the studies. Finally, Section 8 concludes the paper.

[1] Classifications and classes in CTM are equivalent to categories and choices in CPM, respectively. Hence, in the rest of the paper, the terms "classifications" and "categories" will be used interchangeably, and so are the terms "classes" and "choices".

[2] Part of the results of these studies has been reported in Chen and Poon [2], which focuses mainly on how to teach CTM to undergraduates in the computer science and software engineering disciplines.

2. Background concepts of CPM and CTM

2.1. Overview of CPM

In CPM [1, 9], an *environment condition* is a characteristic of the state of the system at the time of executing a functional unit. A *parameter* is an explicit input to a functional unit, supplied either by the user or by another program. For the ease of discussion, we shall collectively refer to environment conditions and parameters as *factors* in this paper.

Categories are defined as the major properties or characteristics of factors that affect the execution behavior of a functional unit. The values of each category are partitioned into distinct subsets known as *choices*. In this paper, (a) categories are enclosed by square brackets [ ], (b) choices are enclosed by vertical bars | |, and (c) the notation |X: x| denotes a choice |x| in the category [X]. For example, the choice |Number of Students: = 0| is the subset {0}, whereas the choice |Number of Students: > 0| is the subset $\{1, 2, \ldots\}$.

Basically, CPM consists of the following six steps:

(1) Decompose a specification into functional units $\mathcal{U}$'s that can be tested independently. For each $\mathcal{U}$ selected for testing, repeat steps (2)–(6) below.

(2) Identify the factors that affect the execution behavior of $\mathcal{U}$. Hence, identify the categories and their associated choices.

(3) Determine constraints among the identified choices. For example, one choice may require another to be present or absent.

(4) Use a generator tool to generate test frames based on the categories and choices identified in (2) and the constraints in (3). Each test frame is a set of choices, with each category contributing to no more than one choice.

(5) For every test frame generated in (4), check whether it is complete or incomplete. Complete test frames are useful for testing, whereas incomplete test frames are discarded.[3]

(6) For each complete test frame, generate a test case by selecting one single element from every choice in that test frame.

Let us use the following example to illustrate the above concepts:

[3] Formal discussions on test frames and their completeness will be given in Section 4.

Example 1 (CPM) Consider, for instance, a program $\mathcal{P}$ that reads an input file F containing two integers m and n and outputs the value of $1/(m+n)$. Let $\mathcal{S}_{\mathcal{P}}$ denote the specification for $\mathcal{P}$. Suppose that, because of the simplicity of $\mathcal{S}_{\mathcal{P}}$, it can be treated as one functional unit $\mathcal{U}_{\mathcal{P}}$ in its entirety, and hence no decomposition is needed as listed in step (1) of CPM above. In step (2), [Status of F] and [$m + n$] are two possible categories identified with respect to an environment condition and a parameter, respectively, that affect the execution behavior of $\mathcal{P}$. The 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|. On the other hand, the category [$m + n$] has two associated choices, namely |$m + n$: $\neq$ 0| and |$m + n$: = 0|. The choice |$m + n$: $\neq$ 0| corresponds to a well-defined result of $1/(m + n)$, whereas the choice |$m + n$: = 0| corresponds to an undefined result involving division by zero.

After identifying the categories and choices, step (3) involves the identification of constraints among choices according to $\mathcal{U}_{\mathcal{P}}$. Here, a possible constraint is that |Status of F: Does Not Exist| cannot be combined with any choice in [$m + n$] to form part of any complete test frame. This is because the values of m and n are irrelevant if F does not exist.

In step (4) of CPM, a generator tool is used to construct a set of test frames based on the identified categories, choices, and constraints. Suppose the generator tool produces six test frames, namely B_1 = {|Status of F: Does Not Exist|}, B_2 = {|Status of F: Exists but Empty|}, B_3 = {|$m + n$: = 0|}, B_4 = {|$m + n$: $\neq$ 0|}, B_5 = {|Status of F: Exists and Non-Empty|, |$m+n$: = 0|}, and B_6 = {|Status of F: Exists and Non-Empty|, |$m + n$: $\neq$ 0|}.

After the above test frames have been generated, the next step is to determine whether they are complete or incomplete by checking with $\mathcal{U}_{\mathcal{P}}$. In this example, B_1, B_2, B_5, and B_6 are complete and hence useful for testing. On the other hand, B_3 and B_4 are incomplete and are discarded. We shall differentiate complete test frames from others by writing B^c instead of B.

Finally, one test case is generated from each complete test frame B^c by selecting one value from every choice in B^c. Consider, for example, B_6^c (= B_6). A possible test case is {Status of F = Exists and Non-Empty, $m + n$ = 78}. ■

Recently, Chen et al. [6] observed several problems that would hinder the effective application of CPM. This observation motivated them to develop a *choice relation framework* for supporting category-partition test case generation. They also conducted empirical studies to demonstrate the effectiveness of the framework. Readers may refer to Chen et al. for details.

2.2. Overview of CTM

Grochtmann and Grimm [8] have proposed CTM as an alternative to CPM for generating test cases from specifications. This method has subsequently been refined by Chen et al. [5].

In CTM, a classification tree organizes classifications and classes at alternate levels in a hierarchical structure. (Recall from footnote 1 that classifications and classes in CTM are equivalent to categories and choices, respectively, in CPM.) The basic approach of CTM is very similar to that of CPM — both of them aim at constructing a model of the constraints in the input domain so that combinations of compatible choices (or classes) can be generated and combinations of incompatible choices (or classes) can be suppressed as far as possible. Since the identification of categories (or classifications) and choices (or classes) is common to both CPM and CTM, our discussions in Sections 5–8 will only refer explicitly to CPM.

3. Overview of function models and function rules

Before we proceed further, we have to introduce the notions of function models and function rules [14]. A *function model* represents the behavior of the system at an abstract level, so that software developers and users can agree on the system behavior without the need for programming details. The mapping between a given set of inputs and the corresponding set of outputs is expressed by means of a *function rule*. This rule states precisely the preconditions for the function to execute and how the outputs are related to the inputs. Note that most function models assume that the system is deterministic, or in other words, the same set of inputs would always lead to the same system behavior and, hence, the same set of outputs. In this paper, we also make this assumption.

Consider Example 1 in Section 2.1 again. The function model for program $\mathcal{P}$ is depicted in Table 1. Each row corresponds to a function rule. The rightmost element in each row is a possible complete test frame, from which a test case can be generated to execute the rule associated with this row. Without doubt, sufficient complete test frames should be generated with a view to uncovering any possible fault associated with each rule.

Function Rule	Set of Valid Inputs	Corresponding Output	Corresponding Complete Test Frame
1	F does not exist	Output the message "F does not exist"	B_1^c = {|Status of F: Does Not Exist|}
2	F exists but is empty	Output the message "F is empty"	B_2^c = {|Status of F: Exists but Empty|}
3	F exists and it contains m and n such that $m + n = 0$	Output the message "Undefined solution"	B_5^c = {|Status of F: Exists and Non-Empty|, $\|m + n: = 0\|$}
4	F exists and it contains m and n such that $m + n \neq 0$	Output the value of $1/(m + n)$	B_6^c = {|Status of F: Exists and Non-Empty|, $\|m + n: \neq 0\|$}

Table 1: A Function Model for Program $\mathcal{P}$

4. Terminology and definitions

As introduced in Section 2.1, *categories* are the major properties or characteristics of factors that affect the execution behavior of a functional unit. For every category $[X]$ proposed by the subjects in our studies, it may either be identified in accordance with the definition, or incorrectly identified with something else in mind. In view of this situation, we shall refer to any $[X]$ identified by the subjects as a *potential category*. Similarly, any $|X: x|$ identified by the subjects is called a *potential choice*.

Definition 1 (Complete and Incomplete Test Frames) *A **test frame** B is a set of potential choices. B is **complete** with respect to $\mathcal{U}$ if, whenever a single element is selected from every potential choice in B, a standalone input for $\mathcal{U}$ is formed. Otherwise, B is **incomplete**.*

As mentioned in Section 2.1, complete test frames are used to generate test cases for testing. On the other hand, incomplete test cases are not useful for testing. They should either be discarded or extended into complete test frames [6]. Examples of complete and incomplete test frames have been given in Example 1.

Further terminology and definitions will be required to lay the foundation for the problems related to the identification of categories and choices. They will be introduced in this section. Related examples will be given in Appendix A.

Definition 2 (Set of Complete Test Frames Related to a Potential Category) *Let $TF_{\mathcal{U}}$ denote the set of complete test frames of $\mathcal{U}$. Given any potential category $[X]$ for $\mathcal{U}$ and all its associated potential choices $|X: x_1|$, $|X: x_2|$, ..., $|X: x_n|$, we define the **set of complete test frames related to** $[X]$ as $TF_{\mathcal{U}}([X]) = \{B^c \in TF_{\mathcal{U}} : |X: x_i| \in B^c$ for some $1 \leq i \leq n\}$.*

Definition 3 (Set of Complete Test Frames Related to a Potential Choice) *Given any potential choice $|X: x|$ in $\mathcal{U}$, we define the **set of complete test frames related to** $|X: x|$ as $TF_{\mathcal{U}}(|X: x|) = \{B^c \in TF_{\mathcal{U}} : |X: x| \in B^c\}$.*

Definition 4 (Set of Complete Test Frames Related to a Test Frame) *Given any test frame B for $\mathcal{U}$, we define the **set of complete test frames related to B** as $TF_{\mathcal{U}}(B) = \{B^c \in TF_{\mathcal{U}} : B \subseteq B^c\}$. A test frame B is said to be **valid** if $TF_{\mathcal{U}}(B) \neq \emptyset$. Otherwise, it is said to be **invalid**.*

We observe from Definitions 1 and 4 that a valid test frame may or may not be complete.

Definition 5 (Relevant and Irrelevant Categories) *Given any potential category $[X]$ for $\mathcal{U}$, if $TF_{\mathcal{U}}([X]) \neq \emptyset$, then $[X]$ is known as a **relevant category**, or simply as a **category**. Otherwise, $[X]$ is known as an **irrelevant category**.*

Definition 6 (Missing Category) *Let $[X_K]$ denote the category associated with the factor K. Suppose PC is a set of potential categories and their associated potential choices identified for $\mathcal{U}$. If K affects the execution behavior of $\mathcal{U}$ but $[X_K] \notin PC$, then $[X_K]$ is a **missing category in PC**. In this case, we also say that PC is a **set with a missing category**.*

Intuitively, some complete test frames may not be constructed because of missing categories. As a result, some function rules of $\mathcal{U}$ are not being tested, so that any faults associated with such rules may not be detected.

Definition 7 (Valid and Invalid Choices) *Given a category $[X]$ for $\mathcal{U}$, any potential choice $|X: x|$ in $[X]$ is said to be **valid** if $TF_{\mathcal{U}}(|X: x|) \neq \emptyset$. Otherwise, $|X: x|$ is **invalid** and $[X]$ is a **category with invalid choices**.*

By Definition 5, given any (relevant) category $[X]$, since $TF_{\mathcal{U}}([X]) \neq \emptyset$, it contains some complete test frame B^c that contains a choice $|X: x|$ in $[X]$. By Definitions 1 and 7, any choice $|X: x| \in B^c$ must be valid. Hence, at least one choice $|X: x|$ in $[X]$ must be valid.

Definition 8 (Missing Choice) *Given a category $[X]$ for $\mathcal{U}$ and all the associated valid choices $|X: x_1|$, $|X: x_2|$, ..., $|X: x_n|$ in $[X]$, if there exists some other valid choice $|X: x|$ yet to be identified and some value $v \in |X: x|$ such that $v \notin |X: x_i|$ for every $1 \leq i \leq n$, then $|X: x|$ is a **missing choice**. In this case, we also say that $[X]$ is a **category with a missing choice.***

Similar to missing categories as defined in Definition 6, the existence of categories with missing choices will cause the omission of some complete test frames. As a result, we may overlook the testing of some parts of the system.

Definition 9 (Overlapping Choices) *Given a category $[X]$ for $\mathcal{U}$, two distinct valid choices $|X: x_1|$ and $|X: x_2|$ are said to be **overlapping** if $|X: x_1| \cap |X: x_2| \neq \emptyset$. In this case, $[X]$ is a **category with overlapping choices**.*

Definition 10 (Combinable Choices) *Suppose $[X]$ is a category for $\mathcal{U}$. Two distinct valid choices $|X: x_1|$ and $|X: x_2|$ in $[X]$ are said to be **combinable** if, for any test frame B, both of the following conditions are satisfied:*

(a) $(B \cup \{|X: x_1|\})$ is a complete test frame if and only if $(B \cup \{|X: x_2|\})$ is a complete test frame.

(b) If $(B \cup \{|X: x_1|\})$ and $(B \cup \{|X: x_2|\})$ are complete test frames, then they are associated with the same function rule of $\mathcal{U}$.

*In this case, $[X]$ is known as a **category with combinable choices**.*

Following Definition 10, we should combine the valid choices $|X: x_1|$ and $|X: x_2|$ into a single valid choice $|X: x_1| \cup |X: x_2|$ so as to reduce the number of complete test frames and hence save testing effort. This replacement will not jeopardize the coverage of the resulting set of complete test frames with respect to the execution of the function rules of $\mathcal{U}$.

Definition 11 (Composite Choice) *Given a category $[X]$ for $\mathcal{U}$, any valid choice $|X: x|$ is said to be **composite** if there exist valid, non-overlapping, and non-combinable choices $|X: x_1|$ and $|X: x_2|$ in $[X]$ such that $|X: x_1| \cup |X: x_2| \subseteq |X: x|$. In this case, $[X]$ is known as a **category with composite choices**.*

It is obvious from Definition 11 that we should consider replacing the composite choice $|X: x|$ by valid choices $|X: x_1|$ and $|X: x_2|$ in order to improve on the preciseness of the complete test frames with respect to the execution of the function rules of $\mathcal{U}$.

Definition 12 (Problematic Choice) *A potential choice $|X: x|$ in a category $[X]$ for $\mathcal{U}$ is said to be **problematic** if at least one of the following criteria is satisfied:*

(a) $|X: x|$ is an invalid choice.

(b) $|X: x|$ is one of the overlapping choices.

(c) $|X: x|$ is one of the combinable choices.

(d) $|X: x|$ is a composite choice.

Definition 13 (Problematic Category) *A potential category $[X]$ for $\mathcal{U}$ is said to be **problematic** if at least one of the following criteria is satisfied:*

(a) $[X]$ is an irrelevant category.

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

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

It should be noted that a problematic category may satisfy more than one criterion listed in Definitions 12 and 13. Consider the category [Number of Regular Meals in First-Class Cabin] in Example 9 of Appendix A. Suppose this category is identified with three associated choices, namely |Number of Regular Meals in First-Class Cabin: < 0|, |Number of Regular Meals in First-Class Cabin: = 0|, and |Number of Regular Meals in First-Class Cabin: ≥ 0|. As explained in Example 9, |Number of Regular Meals in First-Class Cabin: = 0| and |Number of Regular Meals in First-Class Cabin: ≥ 0| are overlapping choices. Furthermore, |Number of Regular Meals in First-Class Cabin: < 0| is an invalid choice because $TF_{\mathcal{U}_{\text{MEAL}}}$(|Number of Regular Meals in First-Class Cabin: < 0|) = $\emptyset$. Hence, [Number of Regular Meals in First-Class Cabin] is a category with overlapping choices as well as an invalid choice.

Definition 14 (Problematic Set of Potential Categories and Potential Choices) *Given a set PC of potential categories and their associated potential choices for $\mathcal{U}$, it is said to be **problematic** if at least one of the following criteria is satisfied:*

(a) PC has missing categories.

(b) PC has problematic categories.

5. Setting of the empirical studies

We have conducted three empirical studies to find out the common mistakes made by testers during an ad hoc identification of categories and choices from informal specifications. The respective specifications used in the three studies are denoted by $\mathcal{S}_{\text{TRADE}}$, $\mathcal{S}_{\text{PURCHASE}}$, and $\mathcal{S}_{\text{MOS}}$.

The first specification $\mathcal{S}_{\text{TRADE}}$ is related to the credit sales of goods by a wholesaler to retail customers, and is mainly in the form of narrative descriptions. In general terms, the system determines whether credit sales should be approved for individual retail customers based on various factors. These factors include the credit status and the credit limit of the customer, the invoice amount of the transactions, and any special management approval by the wholesaler.

The second specification $\mathcal{S}_{\text{PURCHASE}}$ is related to the purchase of goods using credit cards issued by an international bank. There are a variety of credit cards determined by various attributes such as status (diamond, gold, or classic), type (corporate or personal), and credit limit. The main functions of the system are to determine whether a purchase transaction using a credit card should be approved, and to calculate the number of reward points that will result from an approved purchase. The number of reward points further determines the type of benefit (such as free airline tickets and shopping vouchers) the customer is entitled to. Similar to $\mathcal{S}_{\text{TRADE}}$, $\mathcal{S}_{\text{PURCHASE}}$ is mainly written in narrative descriptions.

Finally, $\mathcal{S}_{\text{MOS}}$ is a real-life specification prepared for an international company providing catering service for many different airlines. The company prefers to remain anonymous and will only be referred to as AIR-FOOD. In order to protect the identity of AIR-FOOD and to make $\mathcal{S}_{\text{MOS}}$ suitable for our study, we have slightly amended the original specification before our third study commences. The majority of the content of the original $\mathcal{S}_{\text{MOS}}$, however, has remained intact.

$\mathcal{S}_{\text{MOS}}$ has been produced for a meal ordering system (MOS) that helps AIR-FOOD determine the types and numbers of meals to be prepared and loaded onto each flight. $\mathcal{S}_{\text{MOS}}$ contains various components such as narrative descriptions, screen layouts, and report layouts. MOS has been fully developed and released for production use in AIR-FOOD for several years. Since MOS is relatively more complex, more mistakes have occurred when identifying categories and choices for $\mathcal{S}_{\text{MOS}}$ than when $\mathcal{S}_{\text{TRADE}}$ and $\mathcal{S}_{\text{PURCHASE}}$ are processed. Hence, the majority of the examples discussed in Appendix A refer to $\mathcal{S}_{\text{MOS}}$.

For empirical studies 1 and 2, the subjects are 48 final-year undergraduates in the computer science and software engineering programs at The University of Melbourne (UM). On the other hand, for empirical study 3, the subjects are a mix of 44 undergraduates and postgraduates in the computer science, software engineering, and information technology programs in Swinburne University of Technology (SU). In both universities, a one-hour lecture was devoted to the introduction of CPM and CTM. Teaching of the methods was supported by related literature such as Chen et al. [3], Grochtmann and Grimm [8], and Singh et al. [11]. The lecture was reinforced by a one-hour tutorial with various examples (including the one used in Grochtmann and Grimm [8], which involves a program counting the number of times an element occurs inside a list). The subjects in both universities were being taught by the same instructor using the same teaching materials.

In study 3, after the subjects have learned CPM and CTM, we asked them to carry out the following tasks:

(a) Decompose the specification $\mathcal{S}_{\text{MOS}}$ into several functional units that can be tested independently. For example, there may be a unit $\mathcal{U}_{\text{MEAL}}$ directly related to the generation of daily meal schedules and another unit related to the maintenance of the airline codes.

(b) Suppose we focus on the functional unit $\mathcal{U}_{\text{MEAL}}$. Identify from it a set of categories and their associated choices.

(c) For every identified category or choice, state the reason of its identification.

In studies 1 and 2, on the other hand, we asked the subjects to treat each of the specifications $\mathcal{S}_{\text{TRADE}}$ and $\mathcal{S}_{\text{PURCHASE}}$ denoted by $\mathcal{U}_{\text{TRADE}}$ and $\mathcal{U}_{\text{PURCHASE}}$, respectively, as a single functional unit. This is because $\mathcal{S}_{\text{TRADE}}$ and $\mathcal{S}_{\text{PURCHASE}}$ are less complex than $\mathcal{S}_{\text{MOS}}$ and can therefore be tested in their entirety. For each of $\mathcal{U}_{\text{TRADE}}$ and $\mathcal{U}_{\text{PURCHASE}}$, the subjects are asked to identify a set of categories and their associated choices, and to provide justifications similarly to tasks (b) and (c) above. For all the three studies, the subjects were asked to complete tasks (a)–(c) in about three weeks.

6. Findings, discussions, and recommendations

An initial examination of the potential categories and potential choices identified by the subjects for the three functional units $\mathcal{U}_{\text{TRADE}}$, $\mathcal{U}_{\text{PURCHASE}}$, and $\mathcal{U}_{\text{MEAL}}$ reveals the following:

(a) Table 2 shows the statistics of the potential categories and potential choices identified for each functional unit. Every subject is asked to identify one and only one set of potential categories and their associated potential choices. We shall use *PC*'s to denote these sets. Thus, the number of *PC*'s is equal to the number of subjects.

We observe that the numbers of potential categories and potential choices increase with the complexity of the functional unit, with $\mathcal{U}_{\text{TRADE}}$ being the least complex and $\mathcal{U}_{\text{MEAL}}$ the most complex. We also note that these numbers vary substantially among the subjects, as evidenced by the large ranges and standard derivations of the numbers of potential categories and potential choices identified. The latter observation indicates that the quality of *PC*'s, identified by the subjects in an ad hoc manner, also varies significantly — an argument that we have put forward in Section 1.

(b) The mean numbers of potential choices in each potential category are 2.2 (= $\frac{579}{265}$), 2.4 (= $\frac{1\,138}{475}$), and 2.4 (= $\frac{1\,488}{615}$) for $\mathcal{U}_{\text{TRADE}}$, $\mathcal{U}_{\text{PURCHASE}}$, and $\mathcal{U}_{\text{MEAL}}$, respectively. Hence, the number of potential choices in each potential category is fairly small, even though all the potential choices in a potential category should cover all the input elements relevant to that category. The main reason for a small number of potential choices in each potential category is that a potential choice consists of a set of values. For example, the valid choice $|m+n: \neq 0|$ in Example 1 consists of all integers except zero.

(c) Table 3 shows the statistics of missing and problematic categories for each functional unit. Similar to the numbers of potential categories and potential choices as reported in (a), the numbers of missing categories and problematic categories also increase with the complexity of the functional unit. Note the high percentages of *PC*'s with missing categories and/or problematic categories in all the three functional units. Here, we have two observations:

- The occurrence of missing categories in *PC*'s would mean that the *PC*'s are not comprehensive, since they do not contain sufficient relevant categories (and associated valid choices) to generate enough complete test frames for testing the function rules of each functional unit.
- The occurrence of problematic categories in *PC*'s would mean that the *PC*'s are not effective, since these problematic categories will cause the generation of incomplete test frames.

Let us further analyze the problematic categories identified by the subjects. Consider Table 4 that shows the numbers and percentages of different types of problematic category, and Table 5 that shows the numbers and percentages of *PC*'s containing different types of problematic category. A closer examination reveals that 42 (87.5%), 46 (95.8%), and 41 (93.2%) of the *PC*'s for $\mathcal{U}_{\text{TRADE}}$, $\mathcal{U}_{\text{PURCHASE}}$, and $\mathcal{U}_{\text{MEAL}}$, respectively, contain at least one problematic category.

Refer to the second columns from the left in Tables 4 and 5. Out of the 615 potential categories identified for $\mathcal{U}_{\text{MEAL}}$, we find that 123 (20.0%) are irrelevant with respect to $\mathcal{U}_{\text{MEAL}}$. These irrelevant categories occur in 33 (75.0%) *PC*'s, and are identified with regard to factors related to the execution of functional units other than $\mathcal{U}_{\text{MEAL}}$ in $\mathcal{S}_{\text{MOS}}$. The occurrence of irrelevant categories clearly indicates that the logical decomposition of a specification into several independent functional units is not a trivial task that can be performed effectively without the help of systematic methodologies. For $\mathcal{U}_{\text{TRADE}}$ and $\mathcal{U}_{\text{PURCHASE}}$, no irrelevant category is detected. The main reason for the absence of irrelevant categories in this case is that, for each specification $\mathcal{S}_{\text{TRADE}}$ and $\mathcal{S}_{\text{PURCHASE}}$, the subjects were asked to treat it as one single functional unit and hence no decomposition is required. Thus, it is impossible to identify irrelevant categories for factors outside $\mathcal{U}_{\text{TRADE}}$ and $\mathcal{U}_{\text{PURCHASE}}$.

If we compare Tables 4 and 5, we observe that:

(i) The relative frequency distributions of different types of problematic category are fairly similar across all three studies.

(ii) Categories with composite choices are the most common. On the other hand, categories with combinable choices are the least common.

The above observations together clearly suggest that the ad hoc identification approach is highly ineffective. Without doubt, there is a strong need for systematic methods for identifying (relevant) categories and valid choices from informal specifications.

Based on the above observations and discussions, we formulate the following checklist to help testers detect the existence of missing categories, problematic categories, and problematic choices.

Functional Unit	Number of Sets of Potential Categories and Potential Choices	Number of Potential Categories (Choices)			
		Total	Mean*	Range*	Standard Derivation
$\mathcal{U}_{\text{TRADE}}$	48	265 (579)	5.5 (12.1)	4–9 (10–20)	0.9 (1.5)
$\mathcal{U}_{\text{PURCHASE}}$	48	475 (1 138)	9.9 (23.7)	6–14 (15–35)	2.0 (4.4)
$\mathcal{U}_{\text{MEAL}}$	44	615 (1 488)	14.0 (33.8)	4–40 (10–83)	7.8 (16.7)

(*) by each subject

Table 2: Statistics of Potential Categories and Potential Choices Identified for Each Functional Unit

Functional Unit	Number of *PC*'s*	Number of Missing Categories	Number (%) of *PC*'s* with Missing Categories	Average Number of Missing Categories in Each *PC**	Number of Problematic Categories	Number (%) of *PC*'s* with Problematic Categories	Average Number of Problematic Categories in Each *PC**
$\mathcal{U}_{\text{TRADE}}$	48	1	1 (2.1%)	0.02	43	42 (87.5%)	0.90
$\mathcal{U}_{\text{PURCHASE}}$	48	33	23 (47.9%)	0.69	79	46 (95.8%)	1.65
$\mathcal{U}_{\text{MEAL}}$	44	158	44 (100.0%)	3.59	158	41 (93.2%)	3.59

(*) *PC* = Set of potential categories and potential choices

Table 3: Statistics of Missing and Problematic Categories for Each Functional Unit

A Checklist for Detecting Missing Categories, Problematic Categories, and Problematic Choices:

(1) Due care should be taken when decomposing an informal specification into $\mathcal{U}$'s. In particular, check whether there exist any irrelevant categories identified for factors that are not related to the execution behavior of the selected $\mathcal{U}$.

(2) Check whether there exists any factor K that affects the execution behavior of the selected $\mathcal{U}$ but is not associated with any potential category. If this happens, there will be missing categories that we fail to identify.

(3) For every potential choice $|X: x|$, check whether $TF_{\mathcal{U}}(|X: x|) = \emptyset$. If so, $|X: x|$ is an invalid choice.

(4) For every category $[X]$, check whether the union of all its valid choices identified so far covers all the input values relevant to $[X]$. If not, $[X]$ contains missing choices yet to be identified.

(5) For any non-empty set of valid choices in every category, determine whether these choices are overlapping by checking the existence of common elements.

(6) When identifying potential categories and potential choices, consider also (a) the constraints among potential choices in the formation of complete test frames, and (b) the function rules involving these choices. This will help detect the occurrence of combinable choices and composite choices. The detection of categories with composite choices is particularly important, since our studies have indicated that they are the most common among various types of problematic category.

We cannot guarantee that a process based on the above checklist will necessarily detect all possible missing categories, problematic categories, and problematic choices. According to our analysis and empirical studies, however, such unwarranted cases can be greatly reduced.

7. Validity of the empirical studies

The empirical studies have the following limitations due to the respective settings:

(a) The subjects of studies 1 and 2 were all undergraduates in UM, whereas those of study 3 were

Functional Unit	Number (%) of					
			Categories with Problematic Choices			
	Irrelevant Categories	Categories with Missing Choices	Categories with Invalid Choices	Categories with Overlapping Choices	Categories with Combinable Choices	Categories with Composite Choices
$\mathcal{U}_{\text{TRADE}}$	0 (0.0%)	3 (1.1%)	0 (0.0%)	6 (2.3%)	0 (0.0%)	34 (12.8%)
$\mathcal{U}_{\text{PURCHASE}}$	0 (0.0%)	9 (1.9%)	2 (0.4%)	26 (5.5%)	0 (0.0%)	42 (8.8%)
$\mathcal{U}_{\text{MEAL}}$	123 (20.0%)	12 (2.0%)	14 (2.3%)	4 (0.7%)	5 (0.8%)	4 (0.7%)

Table 4: Numbers and Percentages of Different Types of Problematic Category

Functional Unit	Number (%) of Sets of Potential Categories and Potential Choices (*PC*'s) Containing					
			Categories with Problematic Choices			
	Irrelevant Categories	Categories with Missing Choices	Categories with Invalid Choices	Categories with Overlapping Choices	Categories with Combinable Choices	Categories with Composite Choices
$\mathcal{U}_{\text{TRADE}}$	0 (0.0%)	3 (6.3%)	0 (0.0%)	6 (12.5%)	0 (0.0%)	34 (70.8%)
$\mathcal{U}_{\text{PURCHASE}}$	0 (0.0%)	8 (16.7%)	2 (4.2%)	25 (52.1%)	0 (0.0%)	31 (64.6%)
$\mathcal{U}_{\text{MEAL}}$	33 (75.0%)	7 (15.9%)	11 (25.0%)	4 (9.1%)	3 (6.8%)	4 (9.1%)

Table 5: Numbers and Percentages of *PC*'s Containing Different Types of Problematic Category

a mix of undergraduates and postgraduates in SU. Because of the differences in universities and the subjects' calibers, readers are recommended not to compare the mistakes made by the subjects among these studies, which is not the main objective of the paper. Instead, the paper aims to identify the types of common mistake made by the subjects when identifying categories and choices in the absence of a systematic process.

(b) For studies 1 and 2, the specifications $\mathcal{S}_{\text{TRADE}}$ and $\mathcal{S}_{\text{PURCHASE}}$ were given to the subjects as one single assignment. Hence, we do not know which specification the subjects worked on first, although we think that the majority of the subjects should have started with $\mathcal{S}_{\text{TRADE}}$ because it is less complex. One can argue that the subjects may gain in experience after doing the first case. However, this effect should be minimal, if any, because the subjects were advised of their errors only after they have completed all the tasks for both specifications.

(c) Obviously, the results of our studies might differ if the subjects were real software testers with substantial commercial software development experience instead of being undergraduates and postgraduates. If such were their backgrounds, they might make fewer mistakes in an ad hoc identification of categories and choices. We observe, however, that even in study 3, where most of the postgraduates in SU had real-life IT working experience, problematic categories and choices were identified. This observation supports our earlier argument that the ad hoc identification approach cannot assure the quality of the resulting categories and choices, regardless of the caliber of the subjects.

8. Conclusion

We have analyzed and discussed the common mistakes when software testers use an ad hoc approach to identifying categories and choices from informal specifications. We have conducted three empirical studies via different specifications and testers. To facilitate the analysis of our empirical results, we have formally defined missing categories and various types of problematic category and choice. We have also discussed plausible reasons for the identification of such categories and choices. Our results confirm that missing categories, problematic categories, and problematic choices are likely to occur when the identification of categories and choices is performed in an ad hoc manner. There is, therefore, a great demand for the introduction of systematic identification techniques to improve on the quality of the process and eventually the quality of the resulting test cases.

The contributions of our empirical studies are threefold. First, by defining missing categories and the various types of problematic category and choice and

highlighting them to inexperienced users, testers will be alerted to avoid them. Secondly, the knowledge of such categories and choices and plausible reasons for their identification give researchers and practitioners an insight into the development of systematic identification methods. Thirdly, the effectiveness of any developed identification method can be measured in terms of its ability to screen out missing categories, problematic categories, and problematic choices.

Based on the results of our empirical studies, we have developed an identification checklist to help testers detect the existence of missing categories, problematic categories, and problematic choices when the identification process is performed in an ad hoc manner.

Acknowledgments

We are grateful to the students of the Department of Computer Science and Software Engineering at The University of Melbourne, and those of the School of Information Technology at Swinburne University of Technology who have participated in this study.

Appendix A. Examples to illustrate terminology and definitions

Examples 2–6 refer to the functional unit $\mathcal{U}_{\mathcal{P}}$ in Example 1 of Section 2.1.

Example 2 (Set of Complete Test Frames Related to a Potential Category) The set of complete test frames for $\mathcal{U}_{\mathcal{P}}$ is $\{B_1^c, B_2^c, B_5^c, B_6^c\}$. Hence, the set of complete test frames related to the potential category $[m + n]$ is $TF_{\mathcal{U}_{\mathcal{P}}}([m + n]) = \{B_5^c, B_6^c\}$. ■

Example 3 (Set of Complete Test Frames Related to a Potential Choice) The set of complete test frames related to the potential choice |Status of F: Exists but Empty| is $TF_{\mathcal{U}_{\mathcal{P}}}(|\text{Status of } F\text{: Exists but Empty}|) = \{B_2^c\}$. ■

Example 4 (Set of Complete Test Frames Related to a Test Frame) Consider the test frame B = {|Status of F: Exists and Non-Empty|}. The set of complete test frames related to B is $TF_{\mathcal{U}_{\mathcal{P}}}(B) = \{B_5^c, B_6^c\}$. ■

Example 5 (Relevant and Irrelevant Categories) Refer to Example 2 again. Since $TF_{\mathcal{U}_{\mathcal{P}}}([m+n]) \neq \emptyset$, $[m+n]$ is a relevant category.

Suppose a tester identifies $[m]$ as a potential category with $|m < 0|$, $|m = 0|$, and $|m > 0|$ as its associated potential choices. According to Example 2, the set of complete test frames for $\mathcal{U}_{\mathcal{P}}$ is $\{B_1^c, B_2^c, B_5^c, B_6^c\}$. Any potential choice of $[m]$, however, does not appear in B_1^c, B_2^c, B_5^c, and B_6^c. Hence, $TF_{\mathcal{U}_{\mathcal{P}}}([m]) = \emptyset$ and $[m]$ is therefore an irrelevant category. ■

Example 6 (Missing Category) Suppose [Status of F] is the only category identified for $\mathcal{U}_{\mathcal{P}}$, as if the category $[m + n]$ did not exist. In such circumstances, [Status of F] is the only category that exists in the set PC of potential categories and choices identified for $\mathcal{U}_{\mathcal{P}}$. Consider the factor "$m + n$" that affects the execution behavior of $\mathcal{U}_{\mathcal{P}}$, and in particular whether "$m + n = 0$" or "$m + n \neq 0$". Let $[X_{m+n}]$ denote the category corresponding to the factor "$m+n$". Since $[X_{m+n}] \notin PC$, $[X_{m+n}]$ is a missing category in PC, and PC is said to have a missing category.

Consider again the complete test frames B_5^c = {|Status of F: Exists and Non-Empty|, $|m + n: = 0|$} and B_6^c = {|Status of F: Exists and Non-Empty|, $|m + n: \neq 0|$} in Table 1. Both of them contain a choice in the category $[m + n]$. Thus, if $[m + n]$ is missing, B_5^c and B_6^c will be omitted by mistake because they cannot be constructed solely from the choices in [Status of F]. ■

All the following examples refer to the specification $\mathcal{S}_{\text{MOS}}$ and the functional unit $\mathcal{U}_{\text{MEAL}}$ in Section 5.

Example 7 (Valid and Invalid Choices) Every master flight schedule (MFS) contains a data element called "Weekly Departure Pattern" (WDP), which indicates whether a flight departs on a daily basis. For a non-daily flight, WDP further indicates the day(s) of the week that the flight will depart. Consider, for example, the following two values of WDP:

- (*a*) **WDP = "1234567":** The flight is a daily-flight. Note that a "1", "2", ..., and "7" in WDP indicate that the flight departs on Mondays, Tuesdays, ..., and Sundays, respectively.
- (*b*) **WDP = "––345––":** The flight is a non-daily flight. It only departs on Wednesdays, Thursdays, and Fridays.

According to the specification $\mathcal{S}_{\text{MOS}}$, the MFS for a daily flight will always be used to generate the corresponding daily meal schedule (DMS) on every day of the week without further checks, as long as this MFS is "current". (To avoid lengthy discussion, we shall skip the criteria for determining whether a given MFS is current.) On the other hand, further checking is required for a non-daily flight even though its MFS is current,

in order to determine whether the corresponding DMS should be generated on a particular day of the week.

In study 3, some subjects have identified [WDP] as a category for $\mathcal{U}_{\text{MEAL}}$ with three potential choices, namely |WDP: Daily|, |WDP: Non-Daily|, and |WDP: Others|. Since every flight must depart on a daily or non-daily basis, $TF_{\mathcal{U}_{\text{MEAL}}}$(|WDP: Daily|) $\neq \emptyset$, $TF_{\mathcal{U}_{\text{MEAL}}}$(|WDP: Non-Daily|) $\neq \emptyset$, and $TF_{\mathcal{U}_{\text{MEAL}}}$(|WDP: Others|) $= \emptyset$. Hence, the potential choices |WDP: Daily| and |WDP: Non-Daily| are valid while the potential choice |WDP: Others| is invalid. In this case, [WDP] is a category with an invalid choice. ■

Example 8 (Missing Choice) According to the specification $\mathcal{S}_{\text{MOS}}$, there are three different types of MFS, namely "Outdated", "Current", and "Future". (To avoid lengthy discussion, the details of how to determine the type of an MFS are not included here.) The types of MFS, together with some other information such as WDP introduced in Example 7, determine which MFSs are used to generate the corresponding DMSs on a particular date. It is also specified in $\mathcal{S}_{\text{MOS}}$ that the kitchen of AIR-FOOD installs a monitor to display all types of MFS. This arrangement helps kitchen staff to plan and produce the required meals. Outdated, current, and future MFSs are displayed on the monitor in "Red", "Blue", and "Green" colors, respectively.

Given the above information, [Type of MFS] should be identified as a category, with |Type of MFS: Outdated|, |Type of MFS: Current|, and |Type of MFS: Future| as its associated valid choices. We observe that some subjects have identified only |Type of MFS: Current| and |Type of MFS: Future| as valid choices. Since $TF_{\mathcal{U}_{\text{MEAL}}}$(|Type of MFS: Outdated|) $\neq \emptyset$ and |Type of MFS: Outdated| has not been identified by these subjects, |Type of MFS: Outdated| is a missing choice and [Type of MFS] is a category with a missing choice. ■

Example 9 (Overlapping Choices) The specification $\mathcal{S}_{\text{MOS}}$ states that every seat in a flight belongs to one of the three classes, namely "First", "Business", and "Economy". For each class, passengers can order regular meals or special meals such as vegetarian meals. The types of regular and special meals are different across the three cabin classes. As expected, the regular and special meals offered in the first-class cabin are of better quality than those in the other cabins. The numbers of regular and special meals for each cabin class in a flight is kept in an MFS associated with this flight, so that a corresponding DMS can be generated later.

With the above information, the following categories and valid choices should be identified for $\mathcal{U}_{\text{MEAL}}$:

(a) [Number of Regular Meals in First-Class Cabin] with |Number of Regular Meals in First-Class Cabin: = 0| and |Number of Regular Meals in First-Class Cabin: > 0| as its associated choices.

(b) [Number of Special Meals in First-Class Cabin] with |Number of Special Meals in First-Class Cabin: = 0| and |Number of Special Meals in First-Class Cabin: > 0| as its associated choices.

(c) [Number of Regular Meals in Business-Class Cabin] with |Number of Regular Meals in Business-Class Cabin: = 0| and |Number of Regular Meals in Business-Class Cabin: > 0| as its associated choices.

(d) [Number of Special Meals in Business-Class Cabin] with |Number of Special Meals in Business-Class Cabin: = 0| and |Number of Special Meals in Business-Class Cabin: > 0| as its associated choices.

(e) [Number of Regular Meals in Economy-Class Cabin] with |Number of Regular Meals in Economy-Class Cabin: = 0| and |Number of Regular Meals in Economy-Class Cabin: > 0| as its associated choices.

(f) [Number of Special Meals in Economy-Class Cabin] with |Number of Special Meals in Economy-Class Cabin: = 0| and |Number of Special Meals in Economy-Class Cabin: > 0| as its associated choices.

Thus, by selecting one valid choice from each of the above categories, $2^6 = 64$ different situations are possible when generating test frames.

Consider [Number of Regular Meals in First-Class Cabin] in (a) above. Suppose it is now identified with two associated valid choices |Number of Regular Meals in First-Class Cabin: = 0| and |Number of Regular Meals in First-Class Cabin: $\geq$ 0|. The two valid choices are overlapping because the element (Number of Regular Meals in First-Class Cabin = 0) exists in both choices. Accordingly, [Number of Regular Meals in First-Class Cabin] is a category with overlapping choices. ■

Example 10 (Combinable Choices) Consider the category [Number of Regular Meals in First-Class Cabin] in bullet (a) of Example 9 again. Suppose this category is identified with the following three associated distinct and valid choices:

(i) |Number of Regular Meals in First-Class Cabin: = 0|,

(ii) |Number of Regular Meals in First-Class Cabin: = 1|, and

(iii) |Number of Regular Meals in First-Class Cabin: > 1|.

According to the specification, however, $\mathcal{U}_{\text{MEAL}}$ should treat (*ii*) and (*iii*) in exactly the same way under the same function rules. In particular, given any set B of valid choices, B combines with (*ii*) to form a complete test frame B_1^c if and only if B combines with (*iii*) to form a complete test frame B_2^c. Furthermore, B_1^c and B_2^c should produce the same test results because there is no function rule that states otherwise. In such circumstances, (*ii*) and (*iii*) are combinable into a single choice |Number of Regular Meals in First-Class Cabin: ≥ 1| = |Number of Regular Meals in First-Class Cabin: = 1| ∪ |Number of Regular Meals in First-Class Cabin: > 1|. Thus, [Number of Regular Meals in First-Class Cabin] is a category with combinable choices.

By combining the valid choices (*ii*) and (*iii*), we can reduce the number of complete test frames generated and hence alleviate the testing effort. On the other hand, the coverage of the function rules is not compromised. ■

Example 11 (Composite Choice) The information captured in MFSs will be used to generate the corresponding DMSs. During the generation process, some information in MFSs can be overridden by the following exceptional schedules/records if they exist:

- Exceptional flight schedules (EFSs): They allow users to change the estimated time of departure (ETD) of a flight on a particular date after the MFS of this flight has been created.
- Exceptional crew configuration records (ECCRs): They allow users to change the number of crew members in a flight on a particular date after the MFS of this flight has been created. Such records are necessary since AIR-FOOD needs to prepare meals for the crews as well as the passengers.

Because of the EFSs and ECCRs, any given MFS falls into one of the following situations:

(a) It is associated with an EFS but not an ECCR.

(b) It is associated with an ECCR but not an EFS.

(c) It is associated with an EFS and an ECCR.

(d) It is not associated with any EFS or ECCR.

In order to generate sufficient complete test frames to cover the above four situations, one approach is to identify [Existence of EFS] and [Existence of ECCR] as two categories. The former has |Existence of EFS: Yes| and |Existence of EFS: No| as associated valid choices, whereas the latter has |Existence of ECCR: Yes| and |Existence of ECCR: No| as associated valid choices. Thus, all of the above situations can be catered for by selecting one valid choice in [Existence of EFS] and one in [Existence of ECCR]. For example, the valid choices |Existence of EFS: Yes| and |Existence of ECCR: Yes| will cover situation (c).

In study 3, some subjects have identified [Existence of Exceptional Schedules / Records] as one category instead of defining [Existence of EFS] and [Existence of ECCR]. This new category has |Existence of Exceptional Schedules / Records: EFS and ECCR Do Not Exist| and |Existence of Exceptional Schedules / Records: Otherwise| as associated valid choices. As a result, |Existence of Exceptional Schedules / Records: Otherwise| applies to situations (a), (b), and (c) above, whereas |Existence of Exceptional Schedules / Records: EFS and ECCR Do Not Exist| applies to situation (d). Note that |Existence of Exceptional Schedules / Records: Otherwise| cannot be used to generate different complete test frames to cover situations (a), (b), and (c) separately.

Consider |Existence of Exceptional Schedules / Records: Otherwise|. It can be replaced by three valid and non-overlapping choices, namely |Existence of Exceptional Schedules / Records: Only EFS Exists|, |Existence of Exceptional Schedules / Records: Only ECCR Exists|, and |Existence of Exceptional Schedules / Records: Both EFS and ECCR Exist|. We note the following:

(i) (|Existence of Exceptional Schedules / Records: Only EFS Exists| ∪ |Existence of Exceptional Schedules / Records: Only ECCR Exists|) ⊂ |Existence of Exceptional Schedules / Records: Otherwise|.

(ii) Let

- B be a valid but incomplete test frame {|WDP: Daily|, |Type of MFS: Current|, |Number of Regular Meals in First-Class Cabin: > 0|, |Number of Special Meals in First-Class Cabin: = 0|, |Number of Regular Meals in Business-Class Cabin: > 0|, |Number of Special Meals in Business-Class Cabin: > 0|, |Number of Regular Meals in Economy-Class Cabin: > 0|, |Number of

Special Meals in Economy-Class Cabin: > 0|, ...}.[4]

- |Existence of Exceptional Schedules / Records: Only EFS Exists|) be a complete test frame.
- |Existence of Exceptional Schedules / Records: Only ECCR Exists|) be a complete test frame.

B_1^c and B_2^c are associated with different function rules because, according to the specification $\mathcal{S}_{\mathrm{MOS}}$, B_1^c and B_2^c correspond to two different sources of overriding information in MFSs during the generation of DMSs. In other words, the valid choices |Existence of Exceptional Schedules / Records: Only EFS Exists| and |Existence of Exceptional Schedules / Records: Only ECCR Exists| are non-combinable.

Hence, |Existence of Exceptional Schedules / Records: Otherwise| is a composite choice and [Existence of Exceptional Schedules / Records] is a category with a composite choice. Obviously, the composite choice could be replaced by |Existence of Exceptional Schedules / Records: Only EFS Exists|, |Existence of Exceptional Schedules / Records: Only ECCR Exists|, and |Existence of Exceptional Schedules / Records: Both EFS and ECCR Exist|, with a view to improving the comprehensiveness of the resulting set of complete test frames. ■

References

[1] M.J. Balcer, W.M. Hasling, T.J. Ostrand, Automatic generation of test scripts from formal test specifications, in: Proceedings of the ACM SIGSOFT 3rd Symposium on Software Testing, Analysis, and Verification (TAV 3), ACM, New York, NY, 1989, pp. 210–218.

[2] T.Y. Chen, P.-L. Poon, Experience with teaching black-box testing in a computer science/software engineering curriculum, IEEE Transactions on Education 47 (1) (2004) 42–50.

[3] T.Y. Chen, P.-L. Poon, S.-F. Tang, A systematic method for auditing user acceptance tests, ISACA Journal 5 (1998) 31–36.

[4] T.Y. Chen, P.-L. Poon, S.-F. Tang, T.H. Tse, An experimental analysis of the identification of categories and choices from specifications, 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, 2002, pp. 99–106.

[5] T.Y. Chen, P.-L. Poon, T.H. Tse, An integrated classification-tree methodology for test case generation, International Journal of Software Engineering and Knowledge Engineering 10 (6) (2000) 647–679.

[6] T.Y. Chen, P.-L. Poon, T.H. Tse, A choice relation framework for supporting category-partition test case generation, IEEE Transactions on Software Engineering 29 (7) (2003) 577–593.

[7] L.A. Clarke, J. Hassell, D.J. Richardson, A close look at domain testing, IEEE Transactions on Software Engineering 8 (4) (1982) 380–390.

[8] M. Grochtmann, K. Grimm, Classification trees for partition testing, Software Testing, Verification and Reliability 3 (2) (1993) 63–82.

[9] T.J. Ostrand, M.J. Balcer, The category-partition method for specifying and generating functional tests, Communications of the ACM 31 (6) (1988) 676–686.

[10] S. Rapps, E.J. Weyuker, Selecting software test data using data flow information, IEEE Transactions on Software Engineering 11 (4) (1985) 367–375.

[11] H. Singh, M. Conrad, 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 1997), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 81–90.

[12] J. Wang, R. Hao, J. Wu, TUGEN: an automatic test suite generator integrating data-flow and control-flow methods, in: Digital Technology — Spanning the Universe: Proceedings of the IEEE International Conference on Communications (ICC 1998), vol. 1, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 286–290.

[13] L.J. White, E.I. Cohen, A domain strategy for computer program testing, IEEE Transactions on Software Engineering 6 (3) (1980) 247–257.

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

[15] S.J. Zeil, Selectivity of data-flow and control-flow path criteria, in: Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis, IEEE Computer Society, Washington, DC, 1988, pp. 216–222.

[4] To avoid lengthy discussions, we only list the valid choices in B that have been introduced in earlier examples.