To appear in ***Software Quality Journal***

Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing

Pak-Lok Poon · T. H. Tse · Sau-Fun Tang · Fei-Ching Kuo

Revised: August 7, 2010

Abstract An early step for most black-box testing methods is to identify a set of categories and choices (or their equivalents) from the specification. The identification is often performed in an ad hoc manner, thus the quality of categories and choices is in doubt. Poorly identified categories and choices will affect the comprehensiveness of test cases. In this paper, we describe several comparative studies using three commercial specifications and discuss the major results. The objectives of our studies are: (a) to investigate the differences in the types and amounts of mistakes made between inexperienced and experienced software testers in an ad hoc identification approach, and (b) to determine the extent of mistake reduction after discussing the mistakes with the software testers and providing with them an identification checklist.

Keywords Black-box testing · Choice relation framework · Classification-tree methodology · Software testing

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project no. 717308), a Discovery Grant of the Australian Research Council (project no. DP09847600), and a Departmental General Research Fund of The Hong Kong Polytechnic University (project no. 1-ZV2H).

P.-L. Poon (**Corresponding author**)
School of Accounting and Finance, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
Tel.: +852-2766-7072, Fax: +852-2774-9364
E-mail: afplpoon@inet.polyu.edu.hk

T. H. Tse
Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
E-mail: thtse@cs.hku.hk

S.-F. Tang · F.-C. Kuo
Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn 3122, Australia
E-mail: s.tang6@pgrad.unimelb.edu.au, dkuo@swin.edu.au

1 Introduction

There are many chances to make mistakes in software development, resulting in software design and programming faults (Boehm and Basili, 2001). Reported cases of faulty systems and their associated catastrophes are abundant (Grottke and Trivedi, 2007; National Research Council, 1991; Neumann, 1991; Paulk et al., 1995). Software quality is of utmost importance to both developers and users of software systems. Testing remains the most practical means of assuring the quality of software (Bache and Müllerburg, 1990; Yu et al., 2001).

In general, testing helps reveal failures due to software faults and prevents them from propagating to the final production system, where the cost of fault removal would be far greater (Boehm and Basili, 2001; Miller et al., 1992; Shepard et al., 2001). Studies by IBM and others have shown that to correct a fault after coding is at least ten times as costly as before it, and to correct a production fault is at least 100 times as costly (Perry, 2006). Similar observations are reported in other literature — the cost-escalation factors range from 5:1 to 100:1, depending on the types and sizes of the software systems (Boehm and Basili, 2001; Grottke and Trivedi, 2007).

On average, a software developer spends 40–50% of predelivery development costs on testing in order to achieve reasonable quality levels (Sanders and Curran, 1994; Shepard et al., 2001). Since testing is expensive and labor intensive, it should be well planned, organized, and executed. Among the various activities in testing, the generation of *test suites* (that is, sets of test cases) is particularly important and receives much attention. This is because the comprehensiveness of a test suite determines the scope of testing and in turn the chance of revealing software failures. Not surprisingly, numerous software practitioners and researchers have spent a lot of effort to develop test suite generation methodologies, including domain testing (Beizer, 1990), equivalence partitioning (Myers, 2004), in-parameter-order (Lei and Tai, 1998; Tai and Lei, 2002), the category-partition method (Ostrand and Balcer, 1988), and the classification-tree method (Grochtmann and Grimm, 1993; Hierons et al., 2003; Singh et al., 1997). Recently, an integrated classification-tree methodology (Cain et al., 2004; Chen et al., 2000) and a **CHO**i**C**e re**LAT**ion fram**E**work (CHOC'LATE) (Chen et al., 2003; Poon et al., 2010) were also proposed. (The integrated classification-tree methodology (Cain et al., 2004; Chen et al., 2000) is an extension to the "original" classification-tree method (Grochtmann and Grimm, 1993; Hierons et al., 2003; Singh et al., 1997). For ease of presentation, they will be collectively referred to as the **C**lassification-**T**ree **M**ethodology (CTM).) All these methods fall under the *black-box* approach, where test suites are generated from specifications without knowledge of the internal structure of the programs under test.

In CHOC'LATE and CTM, we observe that identifying categories and choices (which form the basis for subsequent test case generation) from an entire *informal* specification [1] is often done in an ad hoc manner because of the absence of any systematic identification technique. The ad hoc identification approach will certainly pose a threat to the quality of the test cases.

Motivated by this problem, we have conducted several comparative studies using three commercial specifications written primarily in an informal manner. They are follow-up work of our previous studies in Chen et al. (2004) to be outlined in Section 3. The present

[1] In general, there are two types of specifications, namely formal and informal. *Formal* specifications are written in a mathematical notation such as Z (Wordsworth, 1992) and Boolean predicates (Lau and Yu, 2005; Tai, 1996), whereas *informal* specifications are mainly written in natural or graphical languages. Relatively speaking, informal specifications are more popular than formal ones in the commercial software industry.

comparative studies serve two purposes: (a) to verify how the types and numbers of mistakes made by the testers in an ad hoc identification approach vary with their working experience in software development and testing, and (b) to determine, after discussing the mistakes with the testers and providing them with our checklist as a simple guideline for detecting problematic categories and choices (Chen et al., 2004), how many mistakes can be avoided in the next identification exercises.

2 Identification of categories, choices, and their relations

Given a specification (or its *functional units*, which are smaller units of the specification whose corresponding subsystems can be tested independently), an early step in CHOC'LATE (Chen et al., 2003; Poon et al., 2010) and CTM (Cain et al., 2004; Chen et al., 2000; Grochtmann and Grimm, 1993; Hierons et al., 2003; Singh et al., 1997) is to identify categories and their associated choices. (Note that identifying categories and choices, or their equivalents, is also a necessary step in domain testing (Beizer, 1990), equivalence partitioning (Myers, 2004), and in-parameter-order (Lei and Tai, 1998; Tai and Lei, 2002). Thus, the findings reported in this paper are also largely applicable to these methods.) A *category* is defined as a major property or characteristic of a parameter or an environment condition of the software system that affects its execution behavior. The possible values associated with each category are partitioned into disjoint subsets known as *choices*. Similarly, an early step in CTM is to identify classifications and their associated classes. "Classifications" in CTM are equivalent to "categories" in CHOC'LATE, and "classes" in CTM are equivalent to "choices" in CHOC'LATE. For the ease of presentation, we will refer to them only as categories and choices, respectively. For further ease of presentation, parameters and environment conditions are collectively known as *factors* in this paper. In addition, any factor is said to be *influencing* if it affects the execution behavior of a system.

Consider an undergraduate award classification system AWARD, whose main function is to determine whether a student is eligible for graduation. A possible category for AWARD is "Cumulative Number of Credits (N)", with "Cumulative Number of Credits $(N)_{0\leq N<120}$" and "Cumulative Number of Credits $(N)_{N\geq 120}$" as its two associated choices.[2] This category and its associated choices are identified according to the information in the specification that a student needs to accumulate at least 120 credits for graduation. Thus, "Cumulative Number of Credits $(N)_{0\leq N<120}$" corresponds to the situation where a student is not eligible for graduation. On the other hand, "Cumulative Number of Credits $(N)_{N\geq 120}$" corresponds to the situation where a student may or may not be eligible for graduation, depending on other influencing factors such as their average GPA score. Note that a choice may comprise a set of possible values. For example, "Cumulative Number of Credits $(N)_{0\leq N<120}$" $= \{0,1,\ldots,119\}$.

After identifying a set of categories and choices, *constraints* or *relations* at the choice level (in CHOC'LATE) or the category level (in CTM) are defined by software testers. Suppose "Cumulative Number of Credits $(N)_{0\leq N<120}$", "Cumulative Number of Credits $(N)_{N\geq 120}$", "Number of Full Years of Study $(Y)_{0\leq Y<3}$", "Average GPA Score $(S)_{3.5\leq S\leq 4.0}$", and some other choices are identified for AWARD. Suppose further that it is impossible for a student with less than three full years of study to accumulate 120 or more credits. In this case, a constraint (or relation) exists, namely "Cumulative Number of Credits $(N)_{N\geq 120}$" cannot be

[2] In this paper, we will use the notation Q to denote a category, and Q_x to denote a choice x of category Q. When there is no ambiguity, we will simply refer to Q_x as x.

combined with "Number of Full Years of Study $(Y)_{0 \leq Y < 3}$" to form part of any valid choice combination. Another constraint is that "Cumulative Number of Credits $(N)_{0 \leq N < 120}$" or "Cumulative Number of Credits $(N)_{N \geq 120}$" may or may not be combined with "Average GPA Score $(S)_{3.5 \leq S \leq 4.0}$" to form part of any valid choice combination, depending on the final score of each course that a student has obtained.

The identified choices and constraints will then be processed by predefined algorithms in CHOC'LATE or CTM for generating complete test frames. In brief, a *test frame* is a set of choices. A test frame is said to be *complete* if, whenever a single value is selected from each choice, a test case is formed. Otherwise, it is said to be *incomplete*. In the rest of the paper, we will use B, B^c, and *tc* to denote a test frame, a complete test frame, and a test case, respectively. Details of the algorithms provided by CHOC'LATE and CTM to generate B^cs are outside the scope of this paper. Readers may refer to the relevant literature (Cain et al., 2004; Chen et al., 2000, 2003; Grochtmann and Grimm, 1993; Hierons et al., 2003; Poon et al., 2010; Singh et al., 1997) for details.

Once a set of B^cs is generated, test cases can be formed. Consider, for instance, $B^c_1 =$ {Cumulative Number of Credits $(N)_{0 \leq N < 120}$, Number of Full Years of Study $(Y)_{0 \leq Y < 3}$, Average GPA Score $(S)_{3.5 \leq S \leq 4.0}$, ...}. By selecting a single value from each choice in B^c_1, some *tc*s can be formed. An example of such a *tc* is (Cumulative Number of Credits $(N) = 90$, Number of Full Years of Study $(Y) = 2$, Average GPA Score $(S) = 3.7$, ...).

If a valid choice x is missing, for instance, then no B^c containing x will be generated. Consequently, any failure associated with x may not be detected. We note that identifying categories and choices from an entire *informal* specification is often done in an ad hoc manner because of the absence of a systematic identification technique.[3] The quality of the test cases resulting from such an ad hoc approach may be in question.

3 Previous work on category and choice identification

Grochtmann and Grimm (1993) have investigated the feasibility of applying artificial intelligence techniques to automatically identify categories and choices from informal specifications. Their work does not result in much success. They argue that identifying categories and choices is a creative process that probably can never be done automatically in its entirety (Grochtmann and Grimm, 1993). They have then shifted their attention to the identification process based on *formal* specifications. Other researchers such as Amla and Ammann (1992), Hierons et al. (2003), and Singh et al. (1997) have also conducted work in this direction. While we concur with the view that the identification of categories and choices from informal specifications is challenging and cannot be fully automatic, we take the position that research on identification processes for informal specifications is a must, because this type of specification is more commonly accepted by the software industry.

[3] More specifically, a systematic and effective identification technique does not exist for various test suite generation methodologies including domain testing (Beizer, 1990), equivalence partitioning (Myers, 2004), in-parameter-order (Lei and Tai, 1998; Tai and Lei, 2002), CTM (Cain et al., 2004; Chen et al., 2000; Grochtmann and Grimm, 1993; Hierons et al., 2003; Singh et al., 1997), and CHOC'LATE (Chen et al., 2003; Poon et al., 2010). In this paper, we are interested in an effective technique to identify categories and choices that are free from problems and omissions, rather than an effective technique to organize or manipulate the identified categories and choices in order to generate test cases with a high failure detection capability, which is the target of most test suite generation methodologies. Furthermore, the occurrence of missing and problematic categories and choices should also affect the failure detection capability, although the study of such a complex correlation is beyond the scope of the present paper.

As a start, we conducted a literature review into the work on identifying categories and choices from specifications which are not written in a strict formal nature such as Z (Wordsworth, 1992) and Boolean predicates (Lau and Yu, 2005; Tai, 1996). Very little relevant work was found. De la Riva et al. (2006) developed a partition-based approach where categories and choices are identified by systematically examining both the XML schema and the XML query. Chen et al. (2005) and Hartmann et al. (2005) introduced their methods to identify categories and choices from UML activity diagrams, which are only one component in a specification. Thus, our review indicates that a systematic identification technique for the entire informal specification does not exist.

Thereafter, we have conducted some empirical studies to investigate the common mistakes made by testers involving the *entire* informal specifications under an ad hoc identification approach (Chen et al., 2004). We have formally defined these common mistakes under various types of problematic categories and choices and missing categories and choices. A major contribution of our empirical studies in Chen et al. (2004) is to help reduce the chance of repeating these mistakes by making them known to testers. As an interim solution, we have developed a checklist to serve as a simple guideline for detecting missing/problematic categories and choices (Chen et al., 2004).

4 Experimental settings

We conjecture that the types and amounts of mistakes made by testers in an ad hoc identification approach may vary with their past experience. Thus, we conduct the present studies to compare with our previous studies in (Chen et al., 2004) to verify the conjecture. Also, after discussing the mistakes with the subjects and providing them with the checklist, we will further determine how many mistakes the subjects can avoid in the next identification exercises. In this section, we describe and contrast the experimental settings of our present and previous studies.

4.1 The present studies

Our studies use three commercial specifications that are written primarily in an informal manner. These specifications are briefly described as follows.

The first specification $\mathbb{S}_{\mathsf{TRADE}}$ is related to the credit sales of goods by a wholesaler to retail customers. The main function of the system is to decide whether credit sales should be approved for individual retail customers. Such decision considers several issues, including the credit status and credit limit of the customer and the billing amount of the transactions.

The second specification $\mathbb{S}_{\mathsf{PURCHASE}}$ is related to the purchase of goods using credit cards issued by an international bank. Each credit card is associated with several attributes such as status (diamond, gold, or classic), type (corporate or personal), and credit limit (different card statuses will have different credit limits). The main functions of the system are to decide whether a purchase using a credit card should be approved, and to calculate the number of reward points to be granted for an approved purchase. The number of reward points further determines the type of benefit (such as free airline tickets and shopping vouchers) that the customer is entitled to.

The third specification $\mathbb{S}_{\mathsf{MOS}}$ is related to a meal ordering system (MOS), which is being used by an international company providing catering service for many different airlines. The main function of MOS is to help the catering company determine the types (such as normal,

child, and vegetarian) and numbers of meals to be prepared and loaded onto each flight served by the company.

In order to protect the identity of the three companies and to make $\mathbb{S}_{\mathsf{TRADE}}$, $\mathbb{S}_{\mathsf{PURCHASE}}$, and $\mathbb{S}_{\mathsf{MOS}}$ suitable for our studies, we have slightly amended the original specifications before commencing the studies. The majority of the contents of the original specifications, however, have remained unchanged.

We recruited 16 software practitioners as the subjects of our studies. They will be referred to as Subjects 1, 2, ..., 16. In general, their IT qualifications are undergraduate or postgraduate degrees in information technology, information systems, business computing, computer science, computing studies, and computer engineering. (Some of these subjects also have other non-IT academic qualifications such as MBA degrees.) In addition, the subjects have 8 to 20 years of commercial experience in software development and testing, with a mean of 11.9 years of experience. Thus, they are classified as *experienced* testers. On the other hand, the subjects in our previous studies in Chen et al. (2004) were undergraduates or postgraduates with little or no working experience in software development or testing. Hence, they are classified as *inexperienced* testers.

Before commencing our present studies, we prepared all the subjects by giving them a one-hour introduction of CHOC'LATE and CTM, supported by related literature, including Chen et al. (1998), Grochtmann and Grimm (1993), and Singh et al. (1997). The introduction was followed by a one-hour discussion in which some examples of CHOC'LATE and CTM (such as a program counting the number of times that an element occurs in a list (Grochtmann and Grimm, 1993)) were used to reinforce the subjects' understanding of these techniques.

4.2 Our previous studies

Prior to our present studies, we conducted similar studies (Chen et al., 2004) with subjects having less experience in software development and testing. Our previous studies involved 48 final-year undergraduates in the computer science and software engineering programs at The University of Melbourne, and a mix of 44 undergraduates and postgraduates in the computer science, software engineering, and information technology programs in Swinburne University of Technology. All the students had little or no working experience in software development and testing. Thus, when compared with the subjects in the present studies who have, on average, 11.9 years of relevant experience, the students can be considered as inexperienced testers.

The students were introduced to the concepts of the testing methods (such as CTM) in a one-hour lecture using the *same* set of literature. This was followed by a one-hour tutorial discussion using the *same* examples as in the present studies. Even the testing methods were taught by the *same* instructor in both our previous and present studies. They were then given the *same* set of specifications ($\mathbb{S}_{\mathsf{TRADE}}$, $\mathbb{S}_{\mathsf{PURCHASE}}$, and $\mathbb{S}_{\mathsf{MOS}}$) as in the present studies.

Thus, the preparation exercise in our previous studies was essentially the same as in the present studies, in terms of the instructor, teaching method, and teaching material. Furthermore, in both series of studies, the subjects were asked to identify categories and choices from the *same* set of specifications using a similar (ad hoc) identification approach (see Section 6.1 for more details). This arrangement allows us to compare the results between our previous and present studies in a meaningful way.

5 Terminology and definitions

As introduced in Section 1, categories are the major properties or characteristics of influencing factors of a software system. For every category Q proposed by the subjects, it may either be identified according to the definition, or incorrectly identified with something else in mind. In view of this situation, we will refer to any Q identified by the subjects as a *potential category*. Similarly, any Q_x identified by the subjects is called a *potential choice*.

Although readers are advised to refer to our previous paper (Chen et al., 2004) for the details of the different types of mistakes that may occur in an ad hoc identification approach, a basic understanding of such mistakes is needed for further discussions. We will therefore provide an overview of the mistakes below, without going into the detailed formal definitions in Chen et al. (2004).

Any potential category Q is said to be *relevant* if it is defined with respect to a factor that influences the observable results of a software system. Otherwise, it is said to be *irrelevant*. Only relevant categories are useful for test case generation. In the rest of the paper, relevant categories are simply referred to as categories unless otherwise stated. Given a set (denoted by *PC*) of potential categories and their associated potential choices, if there exists a (relevant) category Q such that $Q \notin PC$, then Q is a *missing category* with respect to *PC*.

Given a category Q, any potential choice Q_x is said to be *valid* if there exists some complete test frame B^c such that B^c contains Q_x. Otherwise, Q_x is said to be *invalid*. Obviously, only valid choices are useful for test case generation. In the rest of the paper, valid choices are simply referred to as choices unless otherwise stated. If a (valid) choice x is omitted in *PC*, then it is a *missing choice* with respect to *PC*. Any choice which is not properly identified can be classified into one or more of the following types:

One of the overlapping choices: Given a category Q, two distinct choices Q_x and Q_y are said to be *overlapping* if there exists a common element in both Q_x and Q_y (that is, the two sets of possible values are not disjoint).

One of the combinable choices: Given a category Q, two distinct choices Q_x and Q_y are said to be *combinable* if, for any complete test frames B_1^c and B_2^c containing Q_x and Q_y, respectively, such that $B_1^c \setminus \{Q_x\} = B_2^c \setminus \{Q_y\}$, they are associated with the same function rule in the specification. (The mapping between a given set of system inputs and the corresponding set of system 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 (Chen et al., 2004).) In this case, we should replace the individual Q_x and Q_y by a combined $Q_z = Q_x \cup Q_y$ so as to reduce the number of complete test frames and, hence, save testing effort.

A composite choice: Given a category Q, any choice Q_z is said to be *composite* if there exist valid, nonoverlapping, and noncombinable choices Q_x and Q_y such that $Q_x \cup Q_y \subseteq Q_z$. (Thus, we should replace Q_z by Q_x and Q_y in order to increase the comprehensiveness of the resulting set of complete test frames.)

A potential choice x is said to be *problematic* if at least one of the following criteria is satisfied:

- It is an invalid choice.
- It is one of the overlapping choices.
- It is one of the combinable choices.
- It is a composite choice.

Similarly, a potential category Q is said to be *problematic* if at least one of the following criteria is satisfied:

- It is an irrelevant category.
- It is a category with missing choices.
- It is a category with problematic choices.

Obviously, the occurrence of missing and problematic categories will affect the comprehensiveness of the test suite. To be more specific, non-problematic categories and their associated choices can be directly used for generating test cases, but problematic categories need to be refined or corrected in order that their non-problematic choices can be used in test case generation.

6 Study 1: Effect of tester experience

6.1 Objective and steps

The main objective of the first study is to investigate how the types and amounts of mistakes made in an ad hoc identification approach vary between inexperienced and experienced testers. The manual checking of missing and problematic categories is carried out by one of the authors of this paper, who has substantial experience in CHOC'LATE and CTM. This is also the case for study 2 to be described later. Since MOS contains numerous modules and is fairly complex in logic, we first decompose $\mathbb{S}_{\mathsf{MOS}}$ into several functional units. For instance, there is a functional unit $\mathbb{U}_{\mathsf{MEAL}}$ directly related to the generation of daily meal schedules and other units related to the maintenance of the airline codes and city codes. Such decomposition does not apply to $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$ because their corresponding systems are less complex and, hence, can be tested in their entirety. Thus, we treat $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$ as functional units denoted by $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, respectively. After the subjects have learned CHOC'LATE and CTM, we ask each of them to do the first round of identification exercises according to the following scheme:

(a) **Subjects 1 to 8:** For each of $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, identify from it a set of potential categories and their associated potential choices in an ad hoc manner. Furthermore, for each identified potential category and potential choice, the reason of its identification has to be stated. We have asked the eight subjects to work on $\mathbb{U}_{\mathsf{TRADE}}$ before $\mathbb{U}_{\mathsf{PURCHASE}}$.

(b) **Subjects 9 to 16:** Repeat (a) above for $\mathbb{U}_{\mathsf{MEAL}}$ instead of $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$. The primary reason for choosing $\mathbb{U}_{\mathsf{MEAL}}$ for the study (rather than other functional units of $\mathbb{S}_{\mathsf{MOS}}$) is because generating daily meal schedules is the most important core function of MOS.

Because of the above scheme, there were eight experienced subjects involved in each functional unit. The rationale of breaking the subjects into two groups was to reduce biases in the later investigation on how they performed in study 2. While the subjects were randomly assigned in groups, we have kept the average years of commercial experience in software development and testing of each group of subjects as similar as possible. Note that each subject was asked to identify one *PC* for each assigned functional unit. Thus, the number of *PC*s for each functional unit was equal to the number of subjects. This was also the case for our previous studies in Chen et al. (2004). (In the previous studies, the number of subjects for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$ were 48, 48, and 44, respectively.)

6.2 Findings and discussion

6.2.1 Potential categories and choices

Consider Table 1, which shows the statistics on potential categories and choices identified for each functional unit. Two sets of results are separated by slashes ("/"). The first set corresponds to our previous studies in Chen et al. (2004) involving inexperienced testers, while the second set corresponds to our present studies involving experienced testers. The data outside the brackets correspond to potential categories, while those in brackets correspond to potential choices. We have the following observations from the table:

Table 1 Statistics on potential categories and choices identified by inexperienced and experienced testers

		By inexperienced testers / experienced testers:			
Functional	**Numbers**	**Numbers of potential categories (choices)**			
unit	**of *PC*s** [a]	**Totals**	**Means** [b]	**Ranges**	**Standard deviations**
$\mathbb{U}_{\mathsf{TRADE}}$	48 / 8	265 (579) / 54 (124)	5.5 (12.1) / 6.8 (15.5)	5 (10) / 2 (8)	0.9 (1.5) / 0.7 (2.5)
$\mathbb{U}_{\mathsf{PURCHASE}}$	48 / 8	475 (1 138) / 101 (278)	9.9 (23.7) / 12.6 (34.8)	8 (20) / 4 (11)	2.0 (4.4) / 1.3 (3.1)
$\mathbb{U}_{\mathsf{MEAL}}$	44 / 8	615 (1 488) / 134 (299)	14.0 (33.8) / 16.8 (37.4)	36 (73) / 3 (10)	7.8 (16.7) / 1.5 (3.7)
Averages			**9.8 (23.2) / 12.0 (29.2)**	**16.3 (34.3) / 3.0 (9.7)**	**3.6 (7.5) / 1.2 (3.1)**

[a] *PC* = set of potential categories and choices
[b] For each subject

Observation 1: Complexity of the functional units and the mean numbers of potential categories and choices. The mean numbers of potential categories and choices identified by both inexperienced and experienced testers increase with the complexity of the functional units, with the minimum numbers attached to the least complex $\mathbb{U}_{\mathsf{TRADE}}$ and the maximum numbers attached to the most complex $\mathbb{U}_{\mathsf{MEAL}}$.

Interpretation: A natural reason for this phenomenon is that software systems associated with complex specifications often contain many aspects for testing, thus contributing to more potential categories and choices to be identified.

Observation 2: Variations in the numbers of potential categories and choices. The numbers of potential categories and choices vary substantially among the subjects, as evidenced by the large ranges and standard deviations. This, in turn, indicates that the quality of *PC*s, as identified by the subjects in an ad hoc manner, also varies significantly.

Interpretation: Suppose, among all the *PC*s identified by the subjects, one of them (denoted by PC_0) is a "good" set, in the sense that PC_0 contains all the relevant categories and has no missing or problematic categories. Now, given any set PC_1 with *more* potential categories and choices than PC_0, PC_1 may contain irrelevant categories, categories with invalid choices, or categories with combinable choices. On the other hand, given any set PC_2 with *fewer* potential categories and choices than PC_0, PC_2 may have missing categories or may contain categories with missing or composite choices. Thus, neither PC_1 nor PC_2 is of good quality.

In this case, the large ranges and standard deviations indicate that the quality of *PC*s (identified by the subjects in an ad hoc manner) varies significantly. Thus, the failure identification capability of the B^cs constructed from such *PC* is in doubt. Systematic identification techniques for identifying categories and choices from informal specifications are certainly needed, with a view to improving the quality of the *PC*.

Observation 3: Experience of the subjects and the mean numbers of potential categories and choices. The mean numbers of potential categories and choices identified by experienced testers are about 22% and 26% larger than those by inexperienced testers, respectively.

Interpretation: This observation should be interpreted with caution because, *by itself*, it does not necessarily indicate that the *PC*s identified by experienced testers are more comprehensive than those by inexperienced testers. The comprehensiveness of a *PC* depends on the number of *non-problematic* categories it contains. Even when a large number of potential categories are identified, some of them may be problematic and, hence, not useful for test case generation.

Observation 4: Experience of the subjects and the variations in the numbers of potential categories and choices. Among the three functional units, the ranges and standard deviations are generally much larger for inexperienced testers than for experienced testers. Thus, this observation suggests that the variation in the sizes of *PC*s is much larger for inexperienced testers than for experienced testers.

Interpretation: A plausible reason for the observation is that, by virtue of their experience, the experienced subjects are able to identify *PC*s with more consistent qualities. The observation shows that experience in software development and testing is vital to the identification process.

However, we remind readers to interpret this observation carefully. Our argument that experienced testers are able to identify *PC*s with more *consistent* qualities is put forward in a relative sense, when compared to inexperienced testers. It does not mean that the *PC*s identified by experienced testers are necessarily of *good* quality, as indicated by our later observations that even experienced testers have made numerous mistakes in the identification process. These later observations also suggest that, although practice and experience in software development and testing do contribute to the identification of categories and choices, such practice and experience cannot eliminate the need for systematic techniques.

In summary, the above observations show that:

- When the complexity of the functional units increases, the numbers of potential categories and choices identified by both groups of subjects also increase.
- There are large variations in the numbers of potential categories and choices identified by the subjects, and the variations are generally much larger for inexperienced testers than for experienced testers.
- Compared with inexperienced testers, experienced testers are able to identify more potential categories and choices.

6.2.2 Missing categories

We turn our attention to Table 2, which shows the data on missing categories for each functional unit. We observe the following from Table 2:

Table 2 Total numbers, mean numbers, and mean percentages of missing categories by inexperienced and experienced testers

	By inexperienced testers / experienced testers:		
Functional unit	**Total numbers of missing categories**	**Mean numbers of missing categories in each *PC* [a]**	**% of mean numbers of missing categories in each *PC* [a] in relation to mean numbers of potential categories in each *PC* [a]**
$\mathbb{U}_{\text{TRADE}}$	1 / 5	0.02 / 0.63	0.38% / 9.26%
$\mathbb{U}_{\text{PURCHASE}}$	33 / 5	0.69 / 0.63	6.95% / 4.95%
$\mathbb{U}_{\text{MEAL}}$	158 / 11	3.59 / 1.38	25.69% / 8.21%
Averages		**1.43 / 0.88**	**11.01% / 7.47%**

[a] *PC* = set of potential categories and choices

Observation 5: Complexity of the functional units and the mean numbers of missing categories. Similarly to observation 1, the mean numbers of missing categories in each *PC* generally increase with the complexity of the functional units for both groups of subjects.

Interpretation: A plausible reason for this phenomenon is that, in an ad hoc identification approach, the chance of omitting relevant categories is higher for more complex functional units.

Observation 6: Experience of the subjects and the mean numbers of missing categories. In addition, Table 2 shows that, when considering all three functional units together, the mean number of missing categories in each *PC* is significantly larger (about 63%) for inexperienced testers than for experienced testers.

Interpretation: The observation suggests that experience in software development and testing does help testers a great deal in avoiding the omission of relevant categories in the absence of a systematic identification technique.

In summary, the numbers of missing categories increase with the complexity of the functional units for both groups of subjects. In addition, by virtue of their experience, experienced testers have overlooked less relevant categories than inexperienced testers in the ad hoc identification approach.

6.2.3 Problematic and non-problematic categories

After examining the missing categories, we then analyze the problematic categories and choices identified by experienced testers for the three functional units. It turns out that all these categories and choices can be classified into the problematic types as defined in Chen et al. (2004). In other words, no new type of problematic category and choice is found. This suggests that the list of problematic categories and choices in Chen et al. (2004) is fairly comprehensive.

Table 3 shows the data on problematic and non-problematic categories for each functional unit. Let us first focus on problematic categories.

Observation 7: Complexity of the functional units and the mean numbers/percentages of problematic categories. Similarly to observations 1 and 5, the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories

Table 3 Total numbers, mean numbers, and mean percentages of problematic and non-problematic categories identified by inexperienced and experienced testers

	By inexperienced testers / experienced testers:					
	Problematic categories			Non-problematic categories		
Functional unit	**Total numbers**	**Mean numbers in each *PC*[a]**	**Mean % among all potential categories**	**Total numbers**	**Mean numbers in each *PC*[a]**	**Mean % among all potential categories**
$\mathbb{U}_{\mathsf{TRADE}}$	43 / 5	0.90 / 0.63	16.23% / 9.26%	222 / 49	4.63 / 6.13	83.77% / 90.74%
$\mathbb{U}_{\mathsf{PURCHASE}}$	79 / 12	1.65 / 1.50	16.63% / 11.88%	396 / 89	8.25 / 11.13	83.37% / 88.12%
$\mathbb{U}_{\mathsf{MEAL}}$	158 / 28	3.59 / 3.50	25.69% / 20.90%	457 / 106	10.39 / 13.25	74.31% / 79.10%
Averages		**2.04 / 1.88**	**19.52% / 14.01%**		**7.75 / 10.17**	**80.48% / 85.99%**

[a] *PC* = set of potential categories and choices

among all potential categories increase with the complexity of the functional units for both groups of subjects.

Interpretation: Recall that observation 1 states that the mean numbers of potential categories identified by each subject (that is, the mean numbers of potential categories in each *PC*) increase with the complexity of the functional units. In general, more potential categories to be identified would increase the chance of the occurrence of problematic categories.

Observation 8: Experience of the subjects and the mean numbers/percentages of problematic categories. Across the three functional units, the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories among all the potential categories identified by inexperienced testers are consistently larger (by an average of about 9% and 39%, respectively) than those by experienced testers.

Interpretation: Experience in software development and testing helps testers reduce the chances of identifying problematic categories.

Observation 9: Experience of the subjects and the reduction in the mean numbers of problematic categories. Consider the reduction in the mean numbers of problematic categories in each *PC* for a given functional unit from inexperienced to experienced testers. These reductions are 0.27, 0.15, and 0.09 for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$, respectively. We note that the reductions decrease with the complexity of the functional unit.

Interpretation: Although observation 8 finds that testing experience helps reduce the number of problematic categories, this advantage diminishes as the functional units become more complex. In other words, the need for a systematic identification technique for categories and choices is higher for more complex functional units.

We turn now to non-problematic categories.

Observation 10: Complexity of the functional units and the mean numbers/percentages of non-problematic categories. Table 3 shows that, for both inexperienced and experienced testers, the mean numbers of non-problematic categories increase with the complexity of the functional units (observation 10(a)). However, the table also

shows that the mean percentages of non-problematic categories among all the potential categories identified by both groups of subjects decrease with the complexity of the functional units (observation 10(b)).

Interpretation: Observation 10(a) is consistent with the trend in observation 1 for potential categories. A more complex functional unit has more aspects for testing and, hence, results in the identification of more non-problematic categories. On the other hand, observation 10(b) indicates that, given a potential category Q identified by either inexperienced or experienced testers, the chance of Q being a problematic category is higher for a more complex functional unit.

Observation 11: Experience of the subjects and the mean numbers/percentages of non-problematic categories. In Table 3, the mean number of non-problematic categories in each *PC* and the mean percentage of non-problematic categories among all the potential categories identified by experienced testers are consistently larger (by an average of about 31% and 7%, respectively) than those identified by inexperienced testers across the three functional units.

Interpretation: Once again, experience in software development and testing has a positive effect on the identification of non-problematic categories. In addition, this observation indicates that, given a potential category Q identified by experienced testers, the chance of Q being non-problematic (and, hence, useful for testing) should be higher than by inexperienced testers.

Following up on the above observations, Table 4 is produced to explore in greater detail the interrelationships among the complexity of the functional units, the level of experience of the subjects, and the performance of the subjects in the ad hoc identification exercises. This table shows the percentage increase/decrease in the mean numbers/percentages of potential, missing, problematic, and non-problematic categories identified by both groups of subjects when they work on the next (more complex) functional unit. (Readers may recall that $\mathbb{U}_{\text{TRADE}}$ is the least complex and $\mathbb{U}_{\text{MEAL}}$ is the most complex.) We have the following observations from Table 4:

Table 4 Percentage increase/decrease in mean numbers/percentages of potential, missing, problematic, and non-problematic categories identified by inexperienced and experienced testers

	By inexperienced testers / experienced testers:					
Functional unit	**% increase in mean numbers of *potential* categories in each *PC*** [a]	**% increase in mean numbers of *missing* categories in each *PC*** [a]	**% increase in mean numbers of *problematic* categories in each *PC*** [a]	**% increase in mean percentages of *problematic* categories among all potential categories**	**% increase in mean numbers of *non-problematic* categories in each *PC*** [a]	**% decrease in mean percentages of *non-problematic* categories among all potential categories**
From $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$	80% / 85%	3 350% / 0%	83% / 138%	2% / 28%	78% / 82%	0.5% / 2.9%
From $\mathbb{U}_{\text{PURCHASE}}$ to $\mathbb{U}_{\text{MEAL}}$	41% / 33%	420% / 119%	118% / 133%	54% / 76%	26% / 19%	10.9% / 10.2%

[a] *PC* = set of potential categories and choices

Observation 12: Experience of the subjects and the increase in the mean numbers of missing categories when the complexity of the functional unit increases. When a functional unit becomes more complex, the increase in the mean numbers of missing categories in each *PC* (the third column in Table 4) is much more significant for inexperienced testers than experienced testers.

Interpretation: This observation allows us to draw a conclusion similar to that in observation 6, that is, experience in software development and testing, to some extent, helps testers reduce the occurrence of missing categories in an ad hoc identification approach.

Observation 13: Experience of the subjects and the percentage increase in the mean numbers of potential and missing categories when the complexity of the functional unit increases. Let us compare the second and the third columns in Table 4 about the percentage increase in the mean numbers of potential and missing categories in each *PC*. Consider the data for inexperienced testers in both columns first. As the functional units become more complex, the percentage increase in the mean numbers of missing categories in each *PC* (3 350% from $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$ and 420% from $\mathbb{U}_{\text{PURCHASE}}$ to $\mathbb{U}_{\text{MEAL}}$) are much larger than the percentage increase in the mean numbers of potential categories in each *PC* (80% from $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$ and 41% from $\mathbb{U}_{\text{PURCHASE}}$ to $\mathbb{U}_{\text{MEAL}}$). The phenomenon of consistent percentage increase for missing categories, however, is not applicable to experienced testers when the functional units become more complex. Likewise, in the rightmost column of Table 2, the percentages of the mean numbers of missing categories in each *PC* (in relation to the mean numbers of potential categories in each *PC*) increase with the complexity of the functional units for inexperienced testers but not experienced ones.

Interpretation: When the functional unit becomes more complex, inexperienced testers are able to identify more potential categories, but at the same time they also make more mistakes in terms of the number of missing categories.

Observation 14: Experience of the subjects and the percentage increase in the mean numbers/percentages of problematic categories when the complexity of the functional unit increases. Observation 12 shows that the increase in the mean numbers of missing categories in each *PC* is much more significant for inexperienced testers than experienced testers when the functional units become more complex. In contrast to observation 12, the fourth and the fifth columns in Table 4 show that the percentage increase in the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories among all potential categories are larger for experienced testers than inexperienced testers when the functional units become more complex.

Interpretation: This observation provides further support to our argument in observation 9 that the contribution of testing experience to the reduction of problematic categories becomes less for more complex functional units.

Observation 15: Difference between the percentage increase in the mean numbers of problematic categories and that of non-problematic categories when the complexity of the functional unit increases. Let us compare the fourth and the sixth columns in Table 4 about the percentage increase in the mean numbers of problematic and non-problematic categories in each *PC*. For both groups of subjects, as the functional units become more complex, the percentage increase in the mean numbers of problematic

categories in each *PC* (such as 83% from $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$ for inexperienced testers) are larger than the percentage increase in the mean numbers of non-problematic categories in each *PC* (such as 78% from $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$ for inexperienced testers).

Interpretation: It appears, therefore, that although both inexperienced and experienced testers can identify more non-problematic categories when the functional units become more complex (see observation 10(a)), this advantage is not sufficient to offset the increase in problematic categories at the same time. This phenomenon provides an explanation to observation 10(b) discussed earlier.

We can draw further conclusions by considering related observations together:

Complexity of the functional units. Consider observations 1, 5, 7, 10, 13, and 15. When the complexity of a functional unit increases, there are more aspects to be tested. On one hand, more testing aspects normally leads to more categories and choices (in terms of the number of potential categories and choices (observation 1) and the number of non-problematic categories (observation 10)) to be identified. On the other hand, more testing aspects would also increase the chances of mistakes (in terms of the number of missing categories (observations 5 and 13) and the number of problematic categories (observations 7 and 15)).

Experience of the subjects. Consider observations 6, 8, 9, 11, 12, and 14. When compared with inexperienced testers, experienced testers have fewer missing categories (observations 6 and 12) and problematic categories (observation 8), but more non-problematic categories (observation 11). Hence, experience in software development and testing does help in the ad hoc identification exercises. It should be noted, however, that the contribution of experience to the performance of an ad hoc identification approach decreases with the complexity of the functional unit (observations 9 and 14). Thus, not only are systematic and more effective identification techniques generally needed, but such a demand will grow with the complexity of the specifications.

6.2.4 Types of problematic categories

Let us take a closer examination of the data on the different types of problematic categories, as summarized in Table 5. For each type of problematic category listed in columns 2 to 7, the data outside brackets show the total numbers of that type identified by inexperienced and experienced testers, respectively.[4] The data in brackets show the mean numbers of that type in each *PC*. Similarly, for each type of problematic category listed in columns 2 to 7 of Table 6, the data outside brackets show the percentages of that type among *potential* categories. The data in brackets show the percentages of that type among *problematic* categories.

Observation 16: Experience of the subjects and the mean numbers of different types of problematic categories. The data in brackets in Table 5 show that, with respect to the mean numbers of different types of problematic categories in each *PC*, experienced testers have identified *fewer* irrelevant categories, categories with missing choices, and categories with overlapping choices than their inexperienced counterparts, but *more*

[4] Readers may notice that, for a functional unit shown in any row of Table 5, the sum of the total numbers of different types of problematic categories may be greater than or equal to the corresponding total number of problematic categories shown in Table 3. This is because a category that is erroneously identified may belong to one or more types of problematic categories.

Table 5 Total numbers and mean numbers of different types of problematic categories identified by inexperienced and experienced testers

	By inexperienced testers / experienced testers:					
	Total numbers of different types of problematic categories (Mean numbers of different types of problematic categories in *each PC*[a])					
Functional unit	**Irrelevant categories**	**With missing choices**	**With invalid choices**	**With overlapping choices**	**With combinable choices**	**With composite choices**
$\mathbb{U}_{\text{TRADE}}$	0 / 0 (0.00 / 0.00)	3 / 0 (0.06 / 0.00)	0 / 0 (0.00 / 0.00)	6 / 0 (0.13 / 0.00)	0 / 0 (0.00 / 0.00)	34 / 5 (0.71 / 0.63)
$\mathbb{U}_{\text{PURCHASE}}$	0 / 1 (0.00 / 0.13)	9 / 1 (0.19 / 0.13)	2 / 1 (0.04 / 0.13)	26 / 2 (0.54 / 0.25)	0 / 1 (0.00 / 0.13)	42 / 8 (0.88 / 1.00)
$\mathbb{U}_{\text{MEAL}}$	123 / 14 (2.80 / 1.75)	12 / 2 (0.27 / 0.25)	14 / 4 (0.32 / 0.50)	4 / 1 (0.09 / 0.13)	5 / 2 (0.11 / 0.25)	4 / 7 (0.09 / 0.88)
Averages	**(0.93 / 0.63)**	**(0.17 / 0.13)**	**(0.12 / 0.21)**	**(0.25 / 0.13)**	**(0.04 / 0.13)**	**(0.56 / 0.83)**

[a] *PC* = set of potential categories and choices

Table 6 Percentages of different types of problematic categories identified by inexperienced and experienced testers

	By inexperienced testers / experienced testers:					
	% of different types of problematic categories among all *potential* (*problematic*) categories					
Functional unit	**Irrelevant categories**	**With missing choices**	**With invalid choices**	**With overlapping choices**	**With combinable choices**	**With composite choices**
$\mathbb{U}_{\text{TRADE}}$	0.0% / 0.0% (0.0% / 0.0%)	1.1% / 0.0% (7.0% / 0.0%)	0.0% / 0.0% (0.0% / 0.0%)	2.3% / 0.0% (14.0% / 0.0%)	0.0% / 0.0% (0.0% / 0.0%)	12.8% / 9.3% (79.1% / 100.0%)
$\mathbb{U}_{\text{PURCHASE}}$	0.0% / 1.0% (0.0% / 9.1%)	1.9% / 1.0% (11.4% / 9.1%)	0.4% / 1.0% (2.5% / 9.1%)	5.5% / 2.0% (32.9% / 18.2%)	0.0% / 1.0% (0.0% / 9.1%)	8.8% / 7.9% (53.2% / 66.7%)
$\mathbb{U}_{\text{MEAL}}$	20.0% / 10.4% (77.8% / 50.0%)	2.0% / 1.5% (7.6% / 7.1%)	2.3% / 3.0% (8.9% / 14.3%)	0.7% / 0.7% (2.5% / 3.8%)	0.8% / 1.5% (3.2% / 7.1%)	0.7% / 5.2% (2.5% / 25.0%)
Averages	**6.7% / 3.8% (25.9% / 19.7%)**	**1.7% / 0.8% (8.7% / 5.4%)**	**0.9% / 1.3% (3.8% / 7.8%)**	**2.8% / 0.9% (16.5% / 7.3%)**	**0.3% / 0.8% (1.1% / 5.4%)**	**7.4% / 7.5% (44.9% / 63.9%)**

categories with invalid choices, categories with combinable choices, and categories with composite choices.

Observation 17: Experience of the subjects and the percentages of different types of problematic categories. In Table 6, we observe a tendency similar to observation 16. With respect to the percentages of different types of problematic categories in all *PC*s, experienced testers have identified *fewer* irrelevant categories, categories with missing choices, and categories with overlapping choices than their inexperienced counterparts, but *more* categories with invalid choices, categories with combinable choices, and categories with composite choices.

Interpretation of observations 16 and 17 together: Both observations indicate that experienced testers are not necessarily better than inexperienced ones in every aspect.

Observation 18: The most and the least frequently occurring types of problematic categories. Given a problematic category Q identified by inexperienced or experienced testers, the chance that Q is a category with composite choices is the highest. On the other hand, the chance that it is a category with combinable choices is the smallest.

Interpretation: First, let us refer to Table 1 again. The mean numbers of choices in each category are 2.37 ($= \frac{23.2}{9.8}$) and 2.43 ($= \frac{29.2}{12.0}$) for inexperienced and experienced testers, respectively. The small number of choices per category suggests that all the subjects were inclined to reduce the total number of choices and, in turn, reduce the total number of complete test frames [5] with a view to saving testing effort. Obviously, fewer choices per category would also mean that the chance of having combinable choices is smaller.

Second, without the support of a systematic identification technique, it is difficult for the subjects to partition a given category into choices such that all the values in each choice are similar in their effects on the system's behavior or in the type of output they produce. This difficulty results in categories with composite choices.

Here, we summarize the above observations related to the different types of problematic categories:

- Experienced testers are not necessarily better than inexperienced ones in every aspect. Experienced testers have identified fewer irrelevant categories, categories with missing choices, and categories with overlapping choices than inexperienced testers, but more categories with invalid choices, categories with combinable choices, and categories with composite choices.
- Among the different types of problematic categories, categories with composite choices occur the most, while categories with combinable choices occur the least.

7 Study 2: Effectiveness of checklist guideline

7.1 Objective and steps

In one of our previous papers (Chen et al., 2004), we provided a checklist as a simple guideline for detecting missing/problematic categories and choices despite an ad hoc identification approach. The objective of the second study here is to evaluate the effectiveness of our checklist, in terms of its ability to help testers reduce the occurrence of missing and problematic categories in *PC*s.

When commencing study 2, we first discussed with the 16 subjects the missing and problematic categories involved in their first study, and how the checklist could be used to help detect and remove these mistakes. We then asked each subject to perform a second round of identification exercises according to the following scheme:

(a) **Subjects 1 to 8:** Identify from $\mathbb{U}_{\mathsf{MEAL}}$ a set *PC* of potential categories and their associated potential choices in an ad hoc manner. Then use the checklist as a simple guideline for detecting and removing any mistake from the *PC*, and to refine any

[5] Recall that a complete test frame is a set of choices such that a test case will be formed whenever a single value is selected from each choice.

potential categories and potential choices in the *PC* if necessary. Finally, for every potential category or potential choice that remains in the *PC*, state the reason why it should be identified.

(b) **Subjects 9 to 16:** Repeat (a) above for $\mathbb{U}_{\mathsf{TRADE}}$ followed by $\mathbb{U}_{\mathsf{PURCHASE}}$, instead of $\mathbb{U}_{\mathsf{MEAL}}$.

Note that, in study 1 in which our checklist was not used, Subjects 1 to 8 performed the identification exercises for $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$ only (but not $\mathbb{U}_{\mathsf{MEAL}}$), whereas Subjects 9 to 16 performed the exercise for $\mathbb{U}_{\mathsf{MEAL}}$ only (but not $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$). This arrangement prevents the experienced subjects from building up their knowledge of the same functional unit from study 1 and, hence, allows us to measure the effectiveness of our checklist in an objective manner.

7.2 Findings and discussions

We first consider missing categories:

Observation 19: Reducing the numbers of missing categories. Table 7 shows the total numbers of missing categories we have detected from the *PC*s identified by experienced testers before and after using the checklist. From there, we see that the checklist helps reduce the numbers of missing categories for all three functional units. Considering all the functional units together, the total number of missing categories is reduced by 67%, which is significant.

Table 7 Total numbers of missing categories with and without checklist

	Without checklist / with checklist:
Functional unit	**Total numbers of missing categories**
$\mathbb{U}_{\mathsf{TRADE}}$	5 / 1
$\mathbb{U}_{\mathsf{PURCHASE}}$	5 / 2
$\mathbb{U}_{\mathsf{MEAL}}$	11 / 4
Totals	**21 / 7**

The following reports our observations related to problematic categories:

Observation 20: Reducing the occurrence of different types of problematic categories. Table 8 shows the total numbers of different types of problematic categories identified by experienced testers with and without the use of the checklist. Except for the irrelevant categories identified from $\mathbb{U}_{\mathsf{TRADE}}$, the total numbers for each type of problematic category across the three functional units using the checklist do not exceed those totals when the checklist was not used. Considering all three functional units together, the checklist is able to reduce the numbers of irrelevant categories, categories with missing choices, categories with invalid choices, categories with overlapping choices, categories with combinable choices, and categories with composite choices by about 27%, 100%, 60%, 100%, 100%, and 60%, respectively. Thus, the checklist is most effective in preventing the occurrence of categories with missing/overlapping/combinable choices.

Interpretation of observations 19 and 20 together: Both observations serve as strong evidence that the checklist is fairly effective in reducing missing and problematic categories. These two observations, however, also reconfirm the need for a systematic identification technique because missing and problematic categories still exist even with the use of the checklist.

Table 8 Total numbers of different types of problematic categories with and without checklist

	Without checklist / with checklist:					
	Total numbers of different types of problematic categories					
Functional unit	**Irrelevant categories**	**With missing choices**	**With invalid choices**	**With overlapping choices**	**With combinable choices**	**With composite choices**
$\mathbb{U}_{\text{TRADE}}$	0 / 1	0 / 0	0 / 0	0 / 0	0 / 0	5 / 2
$\mathbb{U}_{\text{PURCHASE}}$	1 / 1	1 / 0	1 / 1	2 / 0	1 / 0	8 / 3
$\mathbb{U}_{\text{MEAL}}$	14 / 9	2 / 0	4 / 1	1 / 0	2 / 0	7 / 3
Totals	**15 / 11**	**3 / 0**	**5 / 2**	**3 / 0**	**3 / 0**	**20 / 8**

Observation 21: Reducing the occurrence of irrelevant categories. Table 8 also shows that, among different types of problematic categories, the checklist is least effective in reducing the occurrence of irrelevant categories.

Interpretation: A plausible reason is that, to avoid the identification of irrelevant categories, testers must determine the influencing factors of the software system under test. This determination task is non-trivial and may sometimes be intangible at the specification stage.

To summarize, among the various types of problematic categories, the checklist is most effective in preventing the occurrence of categories with missing/overlapping/combinable choices, and least effective in reducing the occurrence of irrelevant categories.

8 Threats to validity

There are four threats to validity in our present empirical studies owing to various settings. First, the present studies involve only 16 experienced subjects, compared with 44 to 48 undergraduates and postgraduates in our previous studies (Chen et al., 2004). It would certainly be better if more experienced subjects participated. However, it was not easy to find a large group of experienced software testers who were willing to participate in empirical studies (with or without remuneration). Second, only three specifications were used in the studies. Nevertheless, we believe that even with 16 experienced subjects and three specifications, our findings still provide an inspiring insight into the effect of tester experience on category and choice identification. This is because our empirical studies are largely exploratory in nature ("to find out what is happening", "to seek new insights", and "to generate ideas and hypotheses for future research" (Robson, 2002)) rather than attempts to identify causal relationships among various factors through statistical hypothesis testing.

Third, as stated in Section 6.1, the checking of missing and problematic categories was carried out by one of the authors. Although the author is knowledgeable in CHOC'LATE and CTM, he has a stake in the outcome of the studies as well as prior knowledge of the hypotheses. This may be a potential source of bias.

Fourth, one may argue that the subjects may gain in experience after doing one case (such as $\mathbb{U}_{\mathsf{TRADE}}$). We believe that this effect should be minimal in study 1 because, in this study, Subjects 1 to 8 were advised of their errors only after they have completed all the identification tasks for their assigned functional unit(s). Furthermore, Subjects 9 to 16 were only involved with the functional unit $\mathbb{U}_{\mathsf{MEAL}}$. The experience effect, however, might exist in study 2 because the same groups of subjects were asked to perform the identification exercises for the purpose of applying CHOC'LATE and/or CTM *without* the checklist (in study 1) and then *with* the checklist (in study 2). Their performance in the identification exercises was found to improve, which might indicate that the subjects had benefited from the first study. This was indeed plausible. Although the subjects were experienced, they were not necessarily experienced in CHOC'LATE and/or CTM and had received only one hour of training in the methods. In addition, after the first study, the subjects were given feedback regarding the mistakes they had made, and might have therefore learned from this.

9 Summary and conclusion

We have described our comparative studies using three commercial specifications and involving inexperienced and experienced software testers. In general, experienced testers identified more potential categories and choices. They also had fewer missing categories and problematic categories. At the same time, they identified more non-problematic categories. These observations thus provide evidence that experience in software development and testing does help improve the quality of the identified *PC* despite an ad hoc identification approach. *We must, however, point out that the contribution of experience to the reduction of mistakes decreases with the complexity of the functional units. We find from the empirical results that, although experienced testers can identify more non-problematic categories when the functional units become more complex, this advantage is not sufficient to offset the increase in problematic categories at the same time.* (This phenomenon also occurs for inexperienced testers.) Thus, software development experience cannot replace the demand for a systematic identification methodology.

Regarding the types of mistakes committed, experienced testers are not necessarily better than inexperienced ones in every aspect. First, on one hand, the increase in missing categories in each *PC* is larger for inexperienced testers than experienced testers when the functional units become more complex; on the other hand, the increase in problematic categories in each *PC* is smaller for inexperienced testers than experienced testers as the functional units become more complex. Second, with respect to all potential/problematic categories, experienced testers have identified fewer irrelevant categories and categories with missing/overlapping choices, but more categories with invalid/combinable/composite choices. Because neither the experienced nor the inexperienced testers performed better in all aspects in the identification exercises, it makes sense to involve both groups of testers in the identification process in the real industrial settings.

Moreover, we observe that the use of the checklist helps software testers reduce the occurrence of missing categories and problematic categories of all types. (There may be a threat to validity due to the gain in experience by the subjects after doing the exercises in study 1. Readers may refer to Section 8.) Among the different types of problematic

categories, the checklist is more effective for reducing the occurrence of categories with missing/overlapping/combinable choices, but least effective in reducing the occurrence of irrelevant categories. Our studies also show that, with the use of the checklist, even software practitioners with substantial years of commercial experience in software development and testing still make a number of mistakes.

We end this paper with two final reminders. First, our study results are not restricted to CHOC'LATE and CTM only. As mentioned in Section 1, the identification of categories and choices (or their equivalents) is also needed in domain testing (Beizer, 1990), equivalence partitioning (Myers, 2004), and in-parameter-order (Lei and Tai, 1998; Tai and Lei, 2002). Second, in line with the thoughts of the software community (Briand, 2007; Carver et al., 2008; Porter and Johnson, 1997; Tichy, 1998), observation of human performance is an essential element of software engineering. In this regard, our results should play a part in the contributions to software engineering research.

Acknowledgements We are grateful to the 16 anonymous software practitioners for their invaluable time and effort in participating in the studies. We are also grateful to the associate editor and the two reviewers for their constructive comments of the paper.

References

Amla, N., Ammann, P. (1992). Using Z specifications in category partition testing. In *Systems Integrity, Software Safety, and Process Security: Building the Right System Right: Proceedings of the 7th Annual IEEE Conference on Computer Assurance* (*COMPASS 1992*), (pp. 3–10). Los Alamitos, CA: IEEE Computer Society Press.

Bache, R., & Müllerburg, M. (1990). Measures of testability as a basis for quality assurance. *Software Engineering Journal*, *5*(2), 86–92.

Beizer, B. (1990). *Software Testing Techniques*. New York, NY: Van Nostrand Reinhold.

Boehm, B. W., & Basili, V. R. (2001). Software defect reduction top 10 list. *IEEE Computer*, *34*(1), 135–137.

Briand, L. C. (2007). A critical analysis of empirical research in software testing. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement* (*ESEM 2007*), (pp. 1–8). Los Alamitos, CA: IEEE Computer Society Press.

Briand, L. C., Labiche, Y., Bawar, Z., & Spido, N. T. (2009). Using machine learning to refine category-partition test specifications and test suites. *Information and Software Technology*, *51*(11), 1551–1564.

Cain, A., Chen, T. Y., Grant, D. D., Poon, P.-L., Tang, S.-F., & Tse, T. H. (2004). An automatic test data generation system based on the integrated classification-tree methodology. In C. V. Ramamoorthy, R. Y. Lee, & K. W. Lee (Eds.), *Software Engineering Research and Applications*, Lecture Notes in Computer Science, (vol. 3026, pp. 225–238). Berlin, Germany: Springer.

Carver, J. C., Nagappan, N., & Page, A. (2008). The impact of educational background on the effectiveness of requirements inspections: an empirical study. *IEEE Transactions on Software Engineering*, *34*(6), 800–812.

Chen, T. Y., Poon, P.-L., & Tang, S.-F. (1998). A systematic method for auditing user acceptance tests. *IS Audit and Control Journal*, *5*, 31–36.

Chen, T. Y., Poon, P.-L., Tang, S.-F., & Tse, T. H. (2004). On the identification of categories and choices for specification-based test case generation. *Information and Software Technology*, *46*(13), 887–898.

Chen, T. Y., Poon, P.-L., Tang, S.-F., & Tse, T. H. (2005). Identification of categories and choices in activity diagrams. In *Proceedings of the 5th International Conference on Quality Software* (*QSIC 2005*), (pp. 55–63). Los Alamitos, CA: IEEE Computer Society Press.

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

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

de la Riva, C., Garcia-Fanjul, J., & Tuya, J. (2006). A partition-based approach for XPath testing. In *Proceedings of the International Conference on Software Engineering Advances* (*ICSEA 2006*). Los Alamitos, CA: IEEE Computer Society Press.

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

Grottke, M., & Trivedi, K. S. (2007). Fighting bugs: remove, retry, replicate, and rejuvenate. *IEEE Computer*, *40*(2), 107–109.

Hartmann, J., Vieira, M., Foster, H., & Ruder, A. (2005). A UML-based approach to system testing. *Innovations in Systems and Software Engineering*, *1*(1), 12–24.

Hierons, R. M., Harman, M., & Singh, H. (2003). Automatically generating information from a Z specification to support the classification tree method. In *Proceedings of the 3rd International Conference of B and Z Users*, Lecture Notes in Computer Science, (vol. 2651, pp. 388–407). Berlin, Germany: Springer.

Lau, M. F., & Yu, Y. T. (2005). An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology*, *14*(3), 247–276.

Lei, Y., & Tai, K.-C. (1998). In-parameter-order: a test generation strategy for pairwise testing. In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium* (*HASE 1998*), (pp. 254–261). Los Alamitos, CA: IEEE Computer Society Press.

Miller, K. W., Morell, L. J., Noonan, R. E., Park, S. K., Nicol, D. M., Murrill, B. W., & Voas, J. M. (1992). Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering*, *18*(1), 33–43.

Myers, G. J. (2004). *The Art of Software Testing*. Hoboken, NJ: Wiley.

National Research Council (1991). *Computers at Risk: Safe Computing in the Information Age*. Washington, DC: National Academies Press.

Neumann, P. G. (1991). The computer-related risk of the year: weak links and correlated events. In *Systems Integrity, Software Safety, and Process Security: Proceedings of the 6th Annual Conference on Computer Assurance* (*COMPASS 1991*), (pp. 5–8). Los Alamitos, CA: IEEE Computer Society Press.

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

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (Eds.) (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley.

Perry, W. E. (2006). *Effective Methods for Software Testing*. Indianapolis, IN: Wiley.

Poon, P.-L., Tang, S.-F., Tse, T. H., & Chen, T. Y. (2010). CHOC'LATE: a framework for specification-based testing. *Communications of the ACM*, *53*(4), 113–118.

Porter, A. A., & Johnson, P. M. (1997). Assessing software review meetings: results of a comparative analysis of two experimental studies. *IEEE Transactions on Software*

Engineering, *23*(3), 129–145.

Ramesh, V., Glass, R. L., & Vessey, I. (2004). Research in computer science: an empirical study. *Journal of Systems and Software*, *70*(1–2), 165–176.

Robson, C. (2002). *Real World Research: a Resource for Social Scientists and Practitioner-Researchers*. Oxford, UK: Blackwell.

Sanders, J. W., & Curran, E. (1994). *Software Quality: a Framework for Success in Software Development and Support*. Wokingham, UK: Addison-Wesley.

Shepard, T., Lamb, M., & Kelly, D. (2001). More testing should be taught. *Communications of the ACM*, *44*(6), 103–108.

Singh, H., Conrad, M., & Sadeghipour, S. (1997). 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*), (pp. 81–90). Los Alamitos, CA: IEEE Computer Society Press.

Tai, K.-C. (1996). Theory of fault-based predicate testing for computer programs. *IEEE Transactions on Software Engineering*, *22*(8), 552–562.

Tai, K.-C., & Lei, Y. (2002). A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, *28*(1), 109–111.

Tichy, W. F. (1998). Should computer scientists experiment more? *IEEE Computer*, *31*(5), 32–40.

Wordsworth, J. B. (1992). *Software Development with Z: a Practical Approach to Formal Methods in Software Engineering*. International Computer Science Series. Wokingham, UK: Addison-Wesley.

Yu, Y. T., Tang, S.-F., Poon, P.-L., & Chen, T. Y. (2001). A study on a path-based strategy for selecting black-box generated test cases. *International Journal of Software Engineering and Knowledge Engineering*, *11*(2), 113–138.