To appear in the *Proceeedings of the 1st ACIS International Conference on Software Engineering Research and Applications* (*SERA 2003*), International Association for Computer and Information Science Mt. Pleasant, Michigan (2003)

ADDICT: A Prototype System for Automated Test Data Generation Using the Integrated Classification-Tree Methodology*

A. Cain, T. Y. Chen, D. Grant
School of Information Technology
Swinburne University of Technology
Hawthorn 3122, Australia
{acain,tchen,dgrant}@it.swin.edu.au

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

Sau-Fun Tang‡
School of Information Technology
Swinburne University of Technology
Hawthorn 3122, Australia
sftang@hkbu.edu.hk

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

Abstract

The classification-tree method (CTM) is a black box technique developed by Grochtmann and Grimm, which helps software testers construct test cases systematically from functional specifications. Later, some extension work has been done on CTM by Chen et al., with a view to improving the effectiveness of the method. This extension results in their integrated classification-tree methodology (ICTM). In this paper, we describe and discuss a prototype system ADDICT that is based on ICTM.

1. Introduction

Software testing has two important purposes. First, it is commonly used to reveal the presence of faults in software. Second, even if testing does not reveal any fault, it still provides increased justification for confidence in the correctness of the software [1]. Without doubt, these two purposes can be effectively achieved only when testing is performed in a systematic and thorough manner. Because of this, Ostrand and Balcer [2] have developed the *category-partition method* (CPM) so that test cases can be generated from functional specifications (hereafter simply referred to as specifications) via the notion of formal test specifications. Following up on their work, several studies [3, 4] on CPM have been conducted. Recently, CPM has been enhanced by Chen et al. by means of their *choice relation framework* [5].

Based on CPM, Grochtmann and Grimm [6] have developed a similar but different method — the *classification-tree method* (CTM). The major concept of CTM is to construct test cases via the construction of *classification trees* (denoted by $\mathcal{T}$s). Although the concept of CTM is promising, this method has a major weakness — the absence of a systematic tree construction algorithm. As a result, users of this method are left with a loosely defined task of constructing $\mathcal{T}$s. For complicated specifications, this construction task could be difficult and hence prone to human errors. If a $\mathcal{T}$ is wrongly constructed, the quality of the resultant test cases generated from it will be adversely affected.

The problem of the absence of a tree construction algorithm is alleviated by Chen et al. via their integrated classification-tree methodology (ICTM) [7]. With this methodology, software testers can construct $\mathcal{T}$s by using a systematic tree construction algorithm. In this paper, we discuss the functionality of a prototype system ADDICT (which stands for **A**utomate**D** test **D**ata generation using the **I**ntegrated **C**lassification-**T**ree methodology) built upon ICTM.

The rest of the paper is organized as follows. Section 2 outlines the major concept of ICTM [7]. Section 3 describes

*This work is supported in part by a grant of the Research Grants Council of Hong Kong (Project No. HKU 7029/01E), a research and conference grant of The University of Hong Kong, and a grant from the Australian Research Council (ARC Discovery Project: DP 0450914).

†**Contact Author.** Tel: (852) 2766 7072; fax: (852) 2356 9550; e-mail: acplpoon@inet.polyu.edu.hk

‡Also with the Department of Finance and Decision Sciences, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong.

in detail the various system features of ADDICT. Finally, Section 4 summarizes and concludes the paper.

2. Overview of the integrated classification-tree methodology (ICTM)

Basically, ICTM [7] helps testers construct test cases from specifications via the construction of $\mathcal{T}$s. This construction task is supported by a predefined algorithm. ICTM consists of the following steps:

(1) Decompose the specification into several *functional units* $\mathcal{U}$s that can be tested independently. For each $\mathcal{U}$ selected for testing, repeat steps (2) to (7) below.

(2) Identify *classifications* (different criteria for partitioning the input domain of the selected $\mathcal{U}$) and their associated *classes* (disjoint subsets of values for each classification) for the selected $\mathcal{U}$. For every classification $[X]$, its associated classes should partition the possible values of $[X]$ completely.[1] The grouping of certain values in a single class $|X : x|$ indicates the belief that a test case with any value in $|X : x|$ is essentially as good as one with any other value in that class [2].

(3) Construct a *classification-hierarchy table* $\mathcal{H}_{\mathcal{U}}$ for $\mathcal{U}$, which captures the hierarchical relation for each pair of classifications.

(4) Construct a classification tree $\mathcal{T}_{\mathcal{U}}$ from $\mathcal{H}_{\mathcal{U}}$.

(5) Construct a *combination table* from $\mathcal{T}_{\mathcal{U}}$. Various combinations of classes can then be selected from the table according to a set of predefined selection rules. Each of these combinations of classes is called a *potential test frame*.

(6) Check all the potential test frames against $\mathcal{U}$, with a view to identifying those frames whose combinations of classes contradict $\mathcal{U}$. These potential test frames contradicting $\mathcal{U}$ are known as *illegitimate test frames*; they are not useful to testing and hence should be discarded. After removing all the illegitimate test frames, all the remaining potential test frames are called *complete test frames*. Basically, a complete test frame F is a set of classes such that, when we select one element from every class in F, sufficient information can be derived to execute $\mathcal{U}$.

(7) From each F, construct a test case by selecting one element from each class in F.

Readers may refer to [7] for details. Nevertheless, when we describe the various system features of ADDICT in Section 3, we will elaborate on the above steps with examples.

[1] In this paper, classifications are enclosed by square brackets [ ] while classes are enclosed by vertical bars | |. Furthermore, the notation $|X : x|$ denotes class x in classification X.

3. ADDICT: a prototype system for automated test case generation

We use a commercial-like specification, denoted as PURCHASE (part of it is given in the Appendix), to explain each step of ICTM mentioned in Section 2 and to describe the various features of ADDICT. Note that the current version of ADDICT supports steps (2) to (5) of ICTM.

Step (1) of ICTM (Decomposition of Specification):
The first step is to decompose PURCHASE into a number of independent functional units $\mathcal{U}$s. In our case, because of the simplicity of PURCHASE, no decomposition is needed. In other words, the entire specification can be treated as one functional unit denoted by $\mathcal{U}_{\texttt{PURCHASE}}$.

Step (2) of ICTM (Identification of Classifications and Classes):
From $\mathcal{U}_{\texttt{PURCHASE}}$, the tester identifies 9 classifications and 22 classes. The number of classes contained in a classification ranges from 2 to 5. The following lists four examples of these classifications together with their associated classes:

(*a*) Classification [Class of Credit Card], with |Class of Credit Card: Gold| and |Class of Credit Card: Classic| as its two associated classes.

(*b*) Classification [Credit Limit of Gold Card], with |Credit Limit of Gold Card: \$5,000| and |Credit Limit of Gold Card: \$6,000| as its two associated classes.

(*c*) Classification [Credit Limit of Classic Card], with |Credit Limit of Classic Card: \$2,000| and |Credit Limit of Classic Card: \$3,000| as its two associated classes.

(*d*) Classification [Current Purchase Amount (*PA*)], with |Current Purchase Amount: $PA \leq \$2\,000.00$|, |Current Purchase Amount: $\$2\,000.00 < PA \leq \$3\,000.00$|, |Current Purchase Amount: $\$3\,000.00 < PA \leq \$5\,000.00$|, |Current Purchase Amount: $\$5\,000.00 < PA \leq \$6\,000.00$|, and |Current Purchase Amount: $PA > \$6\,000.00$| as its five associated classes.

Consider Figure 1 which depicts an input screen provided by ADDICT for entering the full and short names of classifications and classes. In this figure, the tester has defined classification [Credit Limit of Gold Card] in the upper-left box, and class |Credit Limit of Gold Card: \$5 000| in the bottom-right box. Additionally, the tester is adding class |Credit Limit of Gold Card: \$6 000| through the upper-right box.

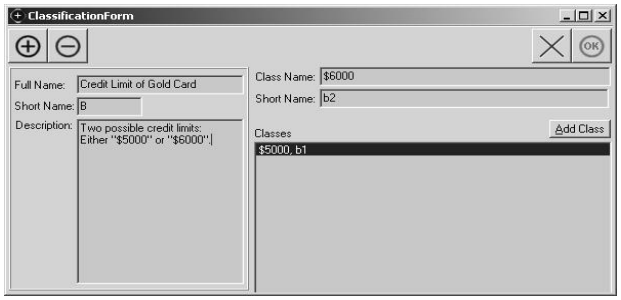

Figure 1. Input screen for classifications and classes

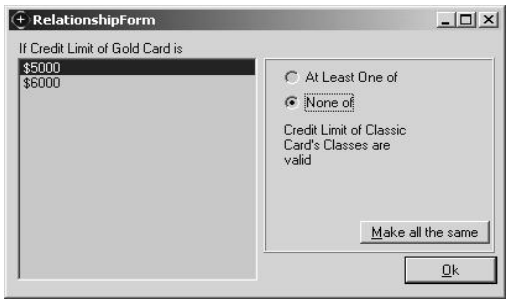

Figure 2. Input screen for constraints between a pair of classifications

Step (3) of ICTM (Construction of Classification-Hierarchy Table):

After entering all the classifications and classes into ADDICT, the next step is to construct the classification-hierarchy table $\mathcal{H}_{\texttt{PURCHASE}}$ for PURCHASE. Here, the tester defines the hierarchical relation for each pair of classifications $[X]$ and $[Y]$ (denoted by $[X] \rightarrow [Y]$). There are four possible types of hierarchical relation as follows:

- $[X]$ is a ***loose ancestor*** of $[Y]$ (denoted by $[X] \Leftrightarrow [Y]$),
- $[X]$ is a ***strict ancestor*** of $[Y]$ (denoted by $[X] \Rightarrow [Y]$),
- $[X]$ is ***incompatible with*** $[Y]$ (denoted by $[X] \sim [Y]$), and
- $[X]$ has ***other relations*** with $[Y]$ (denoted by $[X] \otimes [Y]$).

In the above, the symbols "$\Leftrightarrow$", "$\Rightarrow$", "$\sim$", and "$\otimes$" are known as *hierarchical operators*. Readers may refer to [7] for details, especially the conditions in determining the correct hierarchical relation for $[X] \rightarrow [Y]$. Note that the conditions associated with each of the above hierarchical relations are mutually exclusive and exhaustive, and hence $[X] \rightarrow [Y]$ is well defined. These hierarchical relations will determine the relative position of $[X]$ and $[Y]$ in $\mathcal{T}$. For example, $[X] \Rightarrow [Y]$ corresponds to the situation where $[X]$ will appear as either a parent or an ancestor of $[Y]$ in $\mathcal{T}$.[2]

Figure 2 depicts an input screen to capture the constraints of [Credit Limit of Gold Card] on [Credit Limit of Classic Card]. These captured constraints will be used by ADDICT to determine the appropriate hierarchical operator for [Credit Limit of Gold Card] $\rightarrow$ [Credit Limit of Classic Card]. In the input screen, the tester indicates that |Credit Limit of Gold Card: \$5 000| cannot be combined with any class (that is, |Credit Limit of Classic Card: \$2 000| and |Credit Limit of Classic Card: \$3 000|) in [Credit Limit of Classic Card] to form part of any complete test frame. After entering some other constraints in Figure 2, ADDICT will automatically assign the hierarchical operator "$\sim$" to [Credit Limit of Gold Card] $\rightarrow$ [Credit Limit of Classic Card]. In short, ADDICT will determine and assign the appropriate hierarchical operator to $[X] \rightarrow [Y]$, based on the captured constraints of $[X]$ on $[Y]$.

Figure 3 depicts the completed $\mathcal{H}_{\texttt{PURCHASE}}$ with every element in it contains a hierarchical operator and corresponds to the hierarchical relation between a pair of classifications.[3] We use t_{ij} to denote the element in the ith row and the jth column of $\mathcal{H}_{\mathcal{U}}$. Consider, for example, t_{23} in $\mathcal{H}_{\texttt{PURCHASE}}$. It contains the hierarchical operator "$\sim$", and corresponds to [Credit Limit of Gold Card] $\sim$ [Credit Limit of Classic Card]. Note that the background color of all unassigned elements in $\mathcal{H}_{\texttt{PURCHASE}}$ is initially set to "blue". Once the constraints corresponding to an element t_{ij} have been entered and a hierarchical operator has been assigned to it, the background color of that element will change to "white".

With regard to the construction of $\mathcal{H}_{\mathcal{U}}$, the following features provided by ADDICT are worth mentioning:

(a) A constraint of ICTM is that the parent-child or ancestor-descendant hierarchical relation must be *anti-symmetric* for any pair of classifications. Otherwise a $\mathcal{T}$ cannot be constructed. In other words, $[X] \Rightarrow [Y]$ must imply $[X] \not\Rightarrow [Y]$. Software testers may need to redefine the original set of classifications and classes in order to meet this constraint while preserving the requirements of the target system (see [7] for details).

[2] For the *parent-child* relation, a classification is "directly" placed under one or more classes of another classification. For the *ancestor-descendant* relation, a classification is "indirectly" placed under one or more classes of another classification.

[3] Note that, short names instead of full names for the classifications (both names are entered via the input screen as depicted in Figure 1) are displayed as row and column headings in $\mathcal{H}_{\texttt{PURCHASE}}$. The idea is to fit the entire $\mathcal{H}_{\texttt{PURCHASE}}$ into the screen. In the situation where $\mathcal{H}_{\texttt{PURCHASE}}$ is too large (because of too many classifications) that exceeds the size of the screen, then vertical and horizontal scroll bars can be used to view different parts of $\mathcal{H}_{\texttt{PURCHASE}}$.

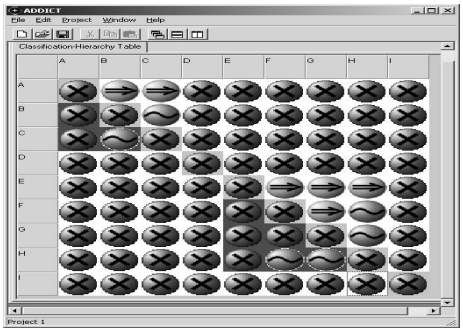

Figure 3. Classification-hierarchy table $\mathcal{H}_{\texttt{PURCHASE}}$ for PURCHASE

Regarding this issue, ICTM helps testers identify such unwarranted situations by means of the hierarchical operator "$\Leftrightarrow$". Whenever $[X] \Leftrightarrow [Y]$ is being defined, we know that a symmetric parent-child or ancestor-descendant hierarchical relation occurs between $[X]$ and $[Y]$. In this case, testers will be alerted to redefine $[X]$ and $[Y]$ (and their associated classes) so as to prevent a loop in $\mathcal{T}$.

Consider t_{12} and t_{21} in $\mathcal{H}_{\texttt{PURCHASE}}$ of Figure 3. They correspond to ([Class of Credit Card] $\Rightarrow$ [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card] $\otimes$ [Class of Credit Card]), respectively. Suppose, during the process of entering the constraints between these two classifications, the tester has made a mistake and eventually caused ADDICT to assign the hierarchical operator "$\Leftrightarrow$" to both t_{12} and t_{21}. Accordingly, the background color of t_{12} and t_{21} will change from "white" to "red", thus alerts the tester that symmetric parent-child or ancestor-descendant hierarchical relations occur. Note that, in this case, the unwarranted situation happens to occur because of an input error; symmetric parent-child or ancestor-descendant hierarchical relations in fact do not exist in $\mathcal{U}_{\texttt{PURCHASE}}$. In some other cases, however, this occurrence may result from correct inputs because symmetric parent-child or ancestor-descendant hierarchical relations do exist between some pairs of classifications identified from $\mathcal{U}$.

(*b*) In [7], Chen et al. have identified three properties of the hierarchical relations as follows:

Property 1: If $[X] \Rightarrow [Y]$, then $[Y] \otimes [X]$.
Property 2: If $[X] \sim [Y]$, then $[Y] \sim [X]$.
Property 3: If $[X] \otimes [Y]$, then $[Y] \Rightarrow [X]$ or $[Y] \otimes [X]$.

Using these properties, ADDICT provides a certain degree of automatic deduction and consistency check during the construction of $\mathcal{H}_{\mathcal{U}}$. Examples are given as below:

Automatic deduction: Consider Figure 2 again. This input screen is used to enter the constraints of each class in [Credit Limit of Gold Card] on [Credit Limit of Classic Card]. The entered constraints cause ADDICT to assign the hierarchical operator "$\sim$" to [Credit Limit of Gold Card] $\rightarrow$ [Credit Limit of Classic Card]. Later, without automatic deduction, the tester is required to enter the constraints of each class in [Credit Limit of Classic Card] on [Credit Limit of Gold Card] via another input screen similar to Figure 2, if such constraints have not yet been entered. Now, by using Property 2, this requirement no longer exists because ADDICT will automatically deduce the hierarchical operator for [Credit Limit of Classic Card] $\rightarrow$ [Credit Limit of Gold Card] to be "$\sim$". Accordingly, the background color for t_{32} (which corresponds to [Credit Limit of Classic Card] $\sim$ [Credit Limit of Gold Card]) will change from "blue" to "green" to inform the tester that the hierarchical operator for t_{32} is automatically deduced (note that the background color for all the table elements whose hierarchical operators are manually defined is "white"). Besides Property 2, ADDICT will also provide automatic deduction based on Property 1. In fact, with the feature of automatic deduction, only about three-quarters of the hierarchical relations in $\mathcal{H}_{\texttt{PURCHASE}}$ have to be manually defined.

Consistency Checking: Consider t_{12} and t_{21} in $\mathcal{H}_{\texttt{PURCHASE}}$ in Figure 3 again, which correspond to ([Class of Credit Card] $\Rightarrow$ [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card] $\otimes$ [Class of Credit Card]), respectively. Suppose,

- The constraints for t_{21} are entered before that for t_{12}.
- The constraints for t_{21} are entered correctly, causing ADDICT to assign the hierarchical operator "$\otimes$" to t_{21}.
- Thereafter, the tester has made a mistake during the entry of the constraints for t_{12}, causing ADDICT to incorrectly assign the hierarchical operator "$\sim$" to t_{12}.

This mistake is undesirable because incorrect hierarchical relations will eventually result in the generation of illegitimate test frames, or the omission of some complete test frames. Regarding this problem, ADDICT provides a consistency check for the defined hierarchical relations. In fact, the incorrect hierarchical operator "$\sim$" for t_{12} will be detected as an inconsistency by ADDICT with reference to Properties (2) and (3) mentioned above. This is

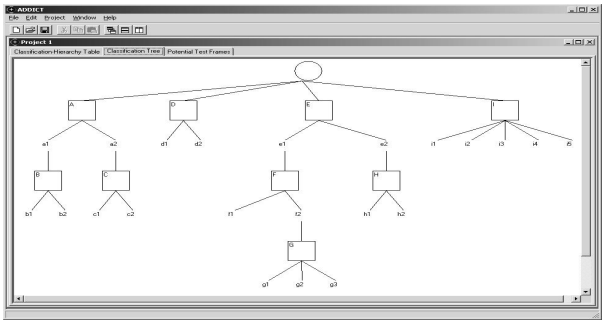

Figure 4. Classification tree $\mathcal{T}_{\texttt{PURCHASE}}$ for PURCHASE

because the combination of ([Class of Credit Card] $\sim$ [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card] $\otimes$ [Class of Credit Card]) contradicts these two properties. Accordingly, the background of t_{12} and t_{21} in $\mathcal{H}_{\texttt{PURCHASE}}$ will change from "white" to "red" to alert the tester to take correction actions. An alert message box will also be displayed automatically by ADDICT to inform the tester about the inconsistency.

Step (4) of ICTM (Construction of Classification Tree):

Based on the completed $\mathcal{H}_{\texttt{PURCHASE}}$ in Figure 3, ADDICT will automatically construct the corresponding classification tree $\mathcal{T}_{\texttt{PURCHASE}}$ (see Figure 4), based on a predefined tree construction algorithm provided in [7]. Similarly to $\mathcal{H}_{\texttt{PURCHASE}}$, short names are used for the classifications and classes in displaying $\mathcal{T}_{\texttt{PURCHASE}}$, and vertical and horizontal scroll bars can be used to view different parts of $\mathcal{T}_{\texttt{PURCHASE}}$ if the tree is too large to fit into the screen.

In step (5) of ICTM described in Section 2, potential test frames are generated by selecting combinations of classes from the combination table of $\mathcal{T}$, based on certain selection rules. Thereafter, the combination of classes in every potential test frame *P* has to be checked against $\mathcal{U}$, with a view to classifying *P* as either a complete test frame or an illegitimate test frame. The reason for checking is because, occasionally, a $\mathcal{T}$ may not be able to capture all the constraints among classifications identified from $\mathcal{U}$. This problem results in the selection of some illegitimate test frames from the combination table of $\mathcal{T}$.

Let N_c and N_p denote the total number of complete test frames and potential test frames, respectively, selected from the combination table of $\mathcal{T}$. In [7], Chen et al. define an effectiveness metric $E_{\mathcal{T}}$ for any $\mathcal{T}$ as $E_{\mathcal{T}} = \frac{N_c}{N_p}$. $E_{\mathcal{T}}$ is defined as such based on the argument that $\mathcal{T}$ is merely a means to construct complete test frames for testing. The ideal situation is that all potential test frames are complete (in this case, $N_c = N_p$), and hence $E_{\mathcal{T}} = 1$. Obviously, a small value of $E_{\mathcal{T}}$ is undesirable since more effort is required to identify all the illegitimate test frames. Furthermore, this manual identification process is prone to human errors, especially when N_p is large. If some complete test frames are somehow mistakenly classified as illegitimate and hence not being used, the comprehensiveness of testing will be adversely affected.

Chen et al. [7] observe that a major reason for a small value of $E_{\mathcal{T}}$ is the occurrence of duplicated subtrees in $\mathcal{T}$. To deal with this problem, they develop a classification tree restructuring technique to suppress the occurrence of duplicated subtrees in $\mathcal{T}$. This restructuring technique is part of their integrated classification tree construction algorithm. Two important properties of this restructuring technique are: (*i*) to reduce the value of N_p by pruning some duplicated subtrees from $\mathcal{T}$, and (*ii*) to retain all the complete test frames and hence N_c remains unchanged. Because of these two properties, the value of the effectiveness metric $E_{\mathcal{T}}$ can be increased. Readers may refer to [7] for details.

In ADDICT, the construction of the resultant $\mathcal{T}$ is performed on an incremental basis — classifications and classes are firstly assembled together to form subtrees, which in turn are joined together to form the resultant $\mathcal{T}$. During the tree construction process, ADDICT will automatically detect the occurrence of duplicated subtrees. If duplicated subtrees do exist, ADDICT will apply the tree restructuring technique by Chen et al., in order to increase the value of $E_{\mathcal{T}}$ of the resultant $\mathcal{T}$. Note that the tree construction process, that incorporates the restructuring technique, is performed by ADDICT in a fully automatic manner without human intervention.

Step (5) of ICTM (Construction of Combination Table and Selection of Potential Test Frames):

With $\mathcal{T}_{\texttt{PURCHASE}}$ in Figure 4, the next step is to construct the corresponding combination table, from which potential test frames can be selected. This step is rather straightforward by following some selection rules given in [7], which will not be repeated here. Same as $\mathcal{T}_{\texttt{PURCHASE}}$, the construction of the combination table and the selection of potential test frames are done by ADDICT automatically. In our case, a total of 240 potential test frames will be selected from the combination table of $\mathcal{T}_{\texttt{PURCHASE}}$ by ADDICT.

Step (6) of ICTM (Differentiation between Complete and Illegitimate Test Frames):

As discussed in step (4) above, the tester has to check all the potential test frames with $\mathcal{U}_{\texttt{PURCHASE}}$ to see whether any of them is illegitimate. In our case, no illegitimate test frame exists, and hence all the 240 potential test frames are also complete.

Step (7) of ICTM (Construction of Test Cases):

For each of the 240 complete test frames Fs, the tester selects an element from each class contained in F to form a test case. Consider, for example, $F_1 = \{|$Class of Credit Card: Gold$|$, $|$Credit Limit of Gold Card: \$6 000$|$, $|$Current Purchase Amount (PA): \$5 000.00 $< PA \leq$ \$6 000.00$|$, ... $\}$, which is a complete test frame generated by ADDICT for $\mathcal{U}_{\texttt{PURCHASE}}$. A possible test case for F_1 is (Class of Credit Card = Gold, Credit Limit of Gold Card = \$6 000, Current Purchase Amount = \$5 123.40, ...). Obviously, a total of 240 test cases will be constructed in this step for testing $\mathcal{U}_{\texttt{PURCHASE}}$.

4. Summary and conclusion

In this paper, we have outlined the various steps of ICTM and discussed the various system features of ADDICT. We believe that ICTM is a viable method for generating test cases from specifications, especially with the support of appropriate automated tools such as ADDICT. CTM has successfully been applied to an airfield lighting system and an adaptive cruise control system [8]. Since ICTM improves on CTM with respect to the construction of $\mathcal{T}$s, and since the sets of complete test frames generated from both methods are the same, the encouraging results when CTM is used to test the above systems should also be applicable to ICTM.

We note that CTM has been used for teaching software testing to undergraduate students in computer science and software engineering programs at two universities in Australia [9] and a university in Hong Kong [10]. The reported results of using CTM for the teaching of software testing are very encouraging. Students of these universities have indicated that CTM is systematic and easy to understand and learn. Since ICTM is an extension of CTM, we believe that the observations should remain unchanged if ICTM is used instead of CTM.

References

[1] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas, "Estimating the Probability of Failure When Testing Reveals no Failures", *IEEE Transactions on Software Engineering*, vol. 18, no. 1, pp. 33–43, January 1992.

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

[3] N. Amla, and P. Ammann, "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 '92*), Gaithersburg, MD, June 1992, pp. 3–10.

[4] P. Ammann, and J. Offutt, "Using Formal Methods to Derive Test Frames in Category-Partition Testing", in *Safety, Reliability, Fault Tolerance, Concurrency, and Real Time Security: Proceedings of the 9th Annual IEEE Conference on Computer Assurance* (*COMPASS '94*), Gaithersburg, MD, June/July 1994, pp. 69–79.

[5] T. Y. Chen, P.-L. Poon, and T. H. Tse, "A Choice Relation Framework for Supporting Category-Partition Test Case Generation", *IEEE Transactions on Software Engineering*, vol. 29, no. 6, June 2003.

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

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

[8] H. Singh, M. Conrad, and S. Sadeghipour, "Test Case Design Based on Z and the Classification-Tree Method", in *Proceedings of the 1st IEEE International Conference on Formal Engineering Methods* (*ICFEM '97*), Hiroshima, Japan, November 1997, pp. 81–90,

[9] T. Y. Chen, and P.-L. Poon, "Experience with Teaching Black-Box Testing in a Computer Science/Software Engineering Curriculum", *IEEE Transactions on Education* (accepted for publication).

[10] Y. T. Yu, S. P. Ng, P.-L. Poon, and T. Y. Chen, "On the Use of the Classification-Tree Method by Beginning Software Testers", in *Proceedings of the 18th Annual ACM Symposium on Applied Computing* (*SAC 2003*), Melbourne, FL, March 2003, pp. 1123–1127.

Appendix (Part of the Specification PURCHASE for the Program $\mathcal{P}_{\texttt{PURCHASE}}$)

XYZ is an international bank that issues credit cards to approved customers. ... For each purchase, $\mathcal{P}_{\texttt{PURCHASE}}$ shall accept the transaction details together with the various information of the credit card. Thereafter, validation of these details is performed in order to determine whether this purchase should be approved. The following are the various inputs to $\mathcal{P}_{\texttt{PURCHASE}}$:

(*a*) Details of Credit Cards:

- **Class of Credit Card:** Either "Gold" or "Classic".
- **Credit Limit of Credit Card:** For gold credit cards, the credit limit is either \$5 000 or \$6 000. For classic credit cards, the credit limit is either \$2 000 or \$3 000.
- ...

(*b*) Details of Purchase:

- **Current Purchase Amount:** It can be any amount greater than \$0.00.
- ...