Proceedings of the IASTED International Conference
SOFTWARE ENGINEERING AND APPLICATIONS
November 6-9, 2000, Las Vegas, Nevada, USA

IMPROVING THE EFFECTIVENESS OF THE CLASSIFICATION-TREE METHODOLOGY IN SOFTWARE TESTING *

T. Y. CHEN
Swinburne University of Technology
Australia

W. H. KWOK and T. H. TSE †
The University of Hong Kong
Hong Kong

Abstract

This paper reports on the experience in the applications of the classification-tree methodology to a real-life system. As a result of the study, we found the need to supplement the original method by introducing a new relation operator and a new technique for constructing classification trees. Our supplement helps to enforce the completeness of test cases and improve on the effectiveness of classification trees.

Keywords: Black-box testing, classification trees, partition testing, software testing, specification-based testing, test case selection.

1 Introduction

As a black-box testing technique, the classification-tree method [1, 2, 3] was developed by Grochtmann and Grimm as an extension of the category-partition method [4, 5] for identifying classifications and their associated classes from a specification. They define *classifications* as the different criteria for partitioning the input domain of the program to be tested, and *classes* as the disjoint subsets of values for each classification. The classifications of the specification are arranged hierarchically to form a classification tree. Test cases for the specification are determined by joining the terminal classes of the tree in a systematic manner.[1]

However, Grochtmann and Grimm assume that software testers can construct a classification tree from a specification according to their personal experience and expertise. No recommendation has been given on the actual process. Chen and Poon [6, 8] enhanced Grochtmann and Grimm's approach by proposing a methodology for building classification trees systematically. They introduced the concept of a *classification-hierarchy table* and a tree construction algorithm to handle the problems of ad hoc examination of the relations among classifications and joining classifications for building a classification tree. They defined four operators ($\otimes$, $\sim$, $\Rightarrow$ and $\Leftrightarrow$) for categorizing the hierarchical relation of a pair of classifications. A summary of the operators is shown in Appendix A.

The algorithm of constructing a classification tree consists of (i) the construction of sub-trees associated with ancestor relations; (ii) the integration of the sub-trees by putting them under a root node; and (iii) the restructuring of duplicated classifications to improve the effectiveness [7, 9, 10] of the classification tree.

It is possible to have test cases generated by a classification tree with incorrect class combinations. Each *legitimate* test case is formed by a set of classifications such that each classification only contributes *one and only one* of its classes to the test case. If a test case involves two mutually exclusive classes within the same classification, it will be *illegitimate.*

Let N_l be the number of legitimate test cases determined from a classification tree and N_t be the total number of test cases from the tree. The *effectiveness* of a classification tree is defined as the ratio of N_l to N_t. In general, $N_t \geq N_l$, and hence the effectiveness of a classification tree is less than or equal to 1.

To differentiate classifications and classes from normal text, we shall use the following two symbols throughout this paper: $\Delta(X)$ indicates a classification X and $\nabla(x)$ indicates a class x.[2]

2 Review of Original Construction Algorithm

2.1 Problem observed

We have studied the construction algorithm proposed by Chen and Poon by applying it to build a classification

*This research is supported in part by grants HKU 7029/97E and CityU 1118/99E of the Hong Kong Research Grants Council and a CRCG grant of the University of Hong Kong.

†All correspondence should be addressed to Dr. T. H. Tse, Department of Computer Science and Information Systems, The University of Hong Kong, Pokfulam Road, Hong Kong. Email: tse@csis.hku.hk

[1] By combining a terminal class originated from every top-level classification at a time, a class combination table is formed. Each row of the table is a test case.

[2] In the classification-tree method, a classification with all its classes is represented by a one-level sub-tree, which looks like a triangle. Hence we use the symbol Δ to represent a classification. On the other hand, a class within a classification is conceptually the inverse, and hence we use an upside-down triangle ∇ to represent a class.

Classification Details	
	Associated classes
Exam Status	*RG,AT*
Cancel Reason	*specified, absent*
User Reply	*go, stop*
Link Type	*PS, PL, SU*
Report Status	*en, ed, pe, del*
Capture Mode	*pending,done*
User Dialogue	*yes, no*

Classification-Hierarchy Table							
	Exam Status	*Cancel Reason*	*User Reply*	*Link Type*	*Report Status*	*Capture Mode*	*User Dialogue*
Exam Status	⊗	⊙	⊙	⇒	⇒	⇒	⇒
Cancel Reason	⊙	⊗	⊙	⊗	⊗	⊙	⊗
User Reply	⊙	⊙	⊗	⊗	⊗	⊙	⊗
Link Type	⊗	⇒	⇒	⊗	⊙	⊙	⇒
Report Status	⊗	⇒	⇒	⊙	⊗	⊙	⇒
Capture Mode	⊗	⊙	⊙	⊙	⊙	⊗	⇒
User Dialogue	⊗	⇒	⇒	⊗	⊗	⊗	⊗

Table 1: **Classification-Hierarchy Table of the Specification for *Examination Cancellation***

tree for an examination cancellation module of a health-care information system. The system is a multi-user database application and has been put into production since 1994. The specification of the module was prepared with data-oriented techniques. It is shown in Appendix B. The classification-hierarchy table for the seven classifications identified from the specification is shown in Table 1.

The classification tree constructed according to the Chen and Poon methodology is shown in Figure 1(i). Let T_1 denote this classification tree. It generates 57 test cases, 26 of which are legitimate. If we check the completeness of the legitimate test cases, we will find that some valid class combinations are not included in the classification tree. The missing class combinations lead to *eight* legitimate test cases that cannot be generated from the classification tree. The classification-hierarchy relations involved with the missing class combinations are tabulated in Table 2. Four of the missing legitimate test cases are formed by combining $\nabla(RG)$ of $\Delta(Exam\ Status)$ with the classes of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$. The other four missing cases are formed by combining $\nabla(pending)$ of $\Delta(Capture\ Mode)$ with the classes of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$.

We note from this table that all the problematic relations involve the ⊙ operator (referred to generally as "other relations" in the Chen and Poon method). On the other hand, the classification tree is derived only from relations involved with the ⇒ operator in the classification-hierarchy table (referred to as "ancestor relations"). The cause of the missing legitimate test cases is the exclusion of a special case of "other relations" when the classification tree is being constructed. We need to refine the "other relations" to tackle this problem.

	Cancel Reason	*User Reply*
Exam Status	⊙	⊙
Capture Mode	⊙	⊙

Table 2: **Relations Excluded by the Construction Algorithm**

2.2 Introduction of a coexistence operator

Intuitively, if any class of a classification can coexist with any class of another classification, we say that the two classifications are independent of each other. We formally introduce a new operator to indicate this type of relation for a pair of classifications in a classification-hierarchy table.

> We define $\Delta(X)$ to be **independent** of $\Delta(Y)$, denoted by $\Delta(X) \odot \Delta(Y)$, if and only if any class $\nabla(x)$ in $\Delta(X)$ can be part of some legitimate input irrespective of whether any class $\nabla(y)$ in $\Delta(Y)$ is part of the same input.

A characteristic of the relation is that given $\Delta(X) \odot \Delta(Y)$, we have $\Delta(Y) \odot \Delta(X)$.

Because of the introduction of the ⊙ operator, we need to modify the definition of the ⊗ operator as well. Thus, we define $\Delta(A)$ to have **other relations** with $\Delta(B)$, denoted by "$\Delta(A) \otimes \Delta(B)$", if and only if $\Delta(A)$ is neither a loose ancestor nor a strict ancestor nor independent of $\Delta(B)$, and is not incompatible with it.

2.3 Problem resolution

To construct a classification tree covering all the valid class combinations according to Table 1, we should not only consider ancestor relations for grouping classifications into sub-trees. We must also group classifications with coexistence relations. In order that classifications described by coexistence relations will not be missed from a classification tree, we draw them as top-level classifications in the classification tree.

In this way, we have $\Delta(Exam\ Status)$, $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ as the top-level classifications of the new classification tree. The

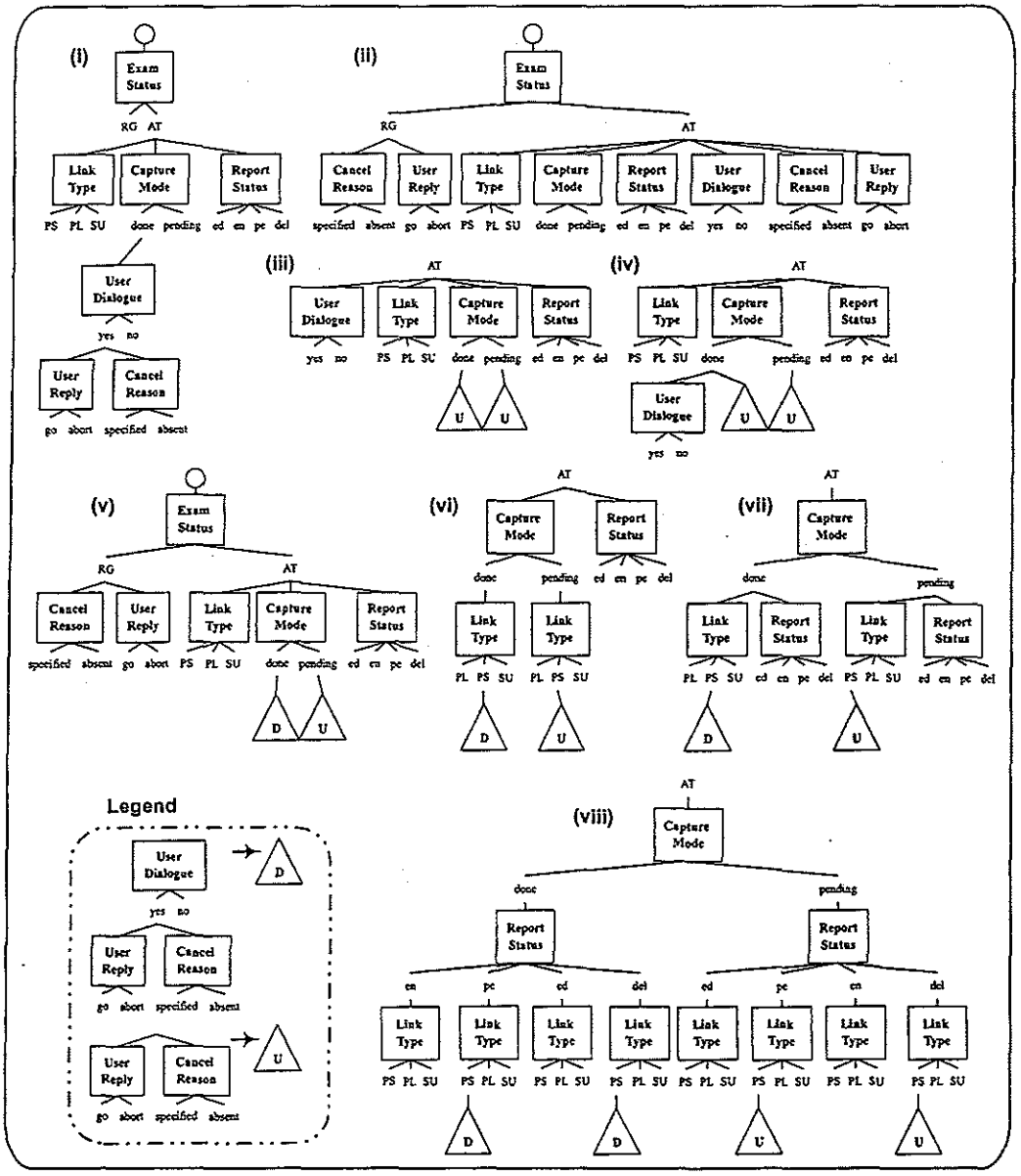

Figure 1: **Steps for Constructing a *Catch-All Sub-Tree***

coexistence relations in Table 2 suggest that we can arrange $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ as the children of every class of $\Delta(Exam\ Status)$. This new arrangement leads to the classification tree of Figure 1(ii). Using the new classification tree, we have resolved the problem of missing legitimate test cases involving $\nabla(RG)$ of $\Delta(Exam\ Status)$ and the classes of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$.

As for the classifications under $\nabla(AT)$ of $\Delta(Exam\ Status)$, Table 2 reminds us to apply a similar arrangement to $\Delta(Capture\ Mode)$. The resulting classifications are shown in Figure 1(iii).

Among the four classifications under $\nabla(AT)$, we have "$\Delta(Capture\ Mode) \Rightarrow \Delta(User\ Dialogue)$". We have thus arranged $\Delta(User\ Dialogue)$ as shown in Figure 1(iv). $\Delta(User\ Dialogue)$ is an ancestor of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ from Table 1. The resulting classifications are shown in Figure 1(v). Let T_1' denote the classification tree in Figure 1(v). The problem of missing legitimate test cases involving $\nabla(pending)$ of $\Delta(Capture\ Mode)$ and the classes of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ have been fixed by T_1'.

Through the construction of T_1', we have illustrated the usefulness of the coexistence relation when constructing a classification tree. The legitimate test cases associated with the relations in Table 2 can be reflected in T_1'.

Furthermore, for this example, we need not apply the restructuring algorithm [7, 9, 10] to improve the effectiveness of T_1' because the sub-tree $S_{\Delta(User\ Dialogue)}$ appears only once in T_1'. As we illustrate below, restructuring is by no means a trivial process.

Referring to Figure 1(i), the arrangement of $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ under $\Delta(User\ Dialogue)$ is the structure of $S_{\Delta(User\ Dialogue)}$. Before applying the restructuring algorithm to T_1, the sub-tree $S_{\Delta(User\ Dialogue)}$ appeared under $\nabla(pe)/\nabla(del)$ of $\Delta(Report\ Status)$, $\nabla(PS)$ of $\Delta(Link\ Type)$, and $\nabla(done)$ of $\Delta(Capture\ Mode)$. Based on the criterion of returning the minimum number of test cases after restructuring, $S_{\Delta(User\ Dialogue)}$ under $\nabla(PS)$ is retained.

For the manipulation of the classifications under $\nabla(AT)$, the only way to retain this sub-tree while maintaining the classes $\Delta(Cancel\ Reason)$ and $\Delta(User\ Reply)$ with $\nabla(pending)$ of $\Delta(Capture\ Mode)$ in T_1' is to make use of the ancestor relation between $\Delta(Capture\ Mode)$ and $\Delta(User\ Dialogue)$ for establishing the hierarchy of these classifications.

3 Effectiveness Improvement with Catch-All Sub-Trees

In the previous section, we have demonstrated the use of both the coexistence relations and ancestor relations to construct a classification tree in a situation where the problems have already been identified. In general, we need to establish a systematic way for constructing a classification tree from a classification-hierarchy table based on the new concept. With this idea in mind, we recommend the following revised procedure:

(a) Construction of Catch-All Sub-Trees

Referring to Table 1 again, we note that top-level classifications cannot be descendants of other classifications. Furthermore, top-level classifications cannot be incompatible with other classifications either. However, any other classifications may be top-level classifications.

Based on these observations, we select the following as potential top-level classifications: classifications that can coexist with other classifications but are not descendents of any other classifications. We construct a sub-tree under each potential top-level classification.

In our earlier example, we have selected $\Delta(Exam\ Status)$ as the only top-level classification of T_1'. In general, we recommend determining it in a systematic manner from the classification-hierarchy table. We consider every row of the classification table. If the classification corresponding to some row is a potential top-level classification, then construct a sub-tree with this classification as the root. In this case, construct other elements in the sub-tree based on every entry in the same row. We will refer to such a sub-tree as a *catch-all sub-tree*, since it captures all types of relations under a potential top-level classification, with the exception of "incompatible" relations. For example, T_1' is the catch-all sub-tree of the row in Table 1 corresponding to the potential top-level classification $\Delta(Exam\ Status)$.

(b) Streamlining the Hierarchy of Catch-All Sub-Trees

Because of the coexistence relations among potential top-level classifications, we can establish a hierarchical relationship among the potential top-level classifications. This hierarchical relationship can be streamlined in a step-by-step fashion, as illustrated below. The streamlining steps enable us to ultimately select one of the potential top-level classifications as the only top-level classification of the final tree.

With Figure 1(v) as the starting point in determining the hierarchy of classifications under $\nabla(AT)$, we put $\Delta(Link\ Type)$ as a child of every class of $\Delta(Capture\ Mode)$. This is illustrated in Figure 1(vi). According to Table 1, $\Delta(Link\ Type)$ is an ancestor of $\Delta(User\ Dialogue)$, $\Delta(Cancel\ Reason)$

Classification Tree	E_T	N_t	N_l
T_1	0.1754	57	10
T_1'	0.1964	112	22
T_1''	1	42	42

Table 3: **Effectiveness Comparison of the Classification Trees**

and Δ(*User Reply*). This explains why we have the hierarchy in Figure 1(vi) after the arrangement.

Following the same principle, we put Δ(*Report Status*) as a child of every class of Δ(*Capture Mode*), as shown in Figure 1(vii). Since

- there is a coexistence relation between Δ(*Link Type*) and Δ(*Report Status*), and
- Δ(*Report Status*) is an ancestor of Δ(*User Dialogue*), Δ(*Cancel Reason*) and Δ(*User Reply*) according to Table 1,

we further put Δ(*Link Type*) as a child of every class of Δ(*Report Status*), as shown in Figure 1(viii). Let T_1'' denote the classification tree in Figure 1(viii). The effectiveness of T_1, T_1' and T_1'' is compared in Table 3. It shows that T_1'' not only generates as many as valid class combinations according to Table 1, but also offers the highest effectiveness of test case generation.

4 Conclusion

In this paper, the construction of a classification tree based on the concept of catch-all sub-trees has been illustrated by a real-life example. A new operator $\odot$ has been introduced in classification-hierarchy tables to highlight coexistence relations that are an important concept in the construction of classification trees.

Using the catch-all sub-tree construction approach, we have resolved the problem of missing valid class combinations caused by solely considering ancestor relations in the original construction algorithm. By applying the new approach to the same real-life example, we have also demonstrated a marked improvement in effectiveness of the resulting classification tree.

References

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

[2] M. Grochtmann, J. Wegener, and K. Grimm, Test case design using classification trees and the classification-tree editor CTE, in *Proceedings of the 8th International Software Quality Week (QW '95)* (San Francisco, California: Software Research Institute, 1995).

[3] H. Singh, M. Conrad, and S. Sadeghipour, Test case design based on Z and the classification-tree method, in *Proceedings of the 1st IEEE International Conference on Formal Engineering Methods (ICFEM '97)* (Los Alamitos, California: IEEE Computer Society, 1997), pp. 81–90.

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

[5] M. J. Balcer, W. M. Hasling, and T. J. Ostrand, Automatic generation of test scripts from formal test specifications, in *Proceedings of the 3rd ACM Annual Symposium on Software Testing, Analysis, and Verification (TAV '89)* (ACM Press, New York, 1989), pp. 210-218.

[6] T. Y. Chen and P. L. Poon, Classification-hierarchy table: a methodology for constructing the classification tree, in *Proceedings of 1996 Australian Software Engineering Conference (ASWEC '96)* (Los Alamitos, California: IEEE Computer Society, 1996), pp. 93–104.

[7] T. Y. Chen and P. L. Poon, Improving the quality of classification trees via restructuring, in *Proceedings of 3rd Asia-Pacific Software Engineering Conference (APSEC '96)* (Los Alamitos, California: IEEE Computer Society, 1996), pp. 83–92.

[8] T. Y. Chen and P. L. Poon, Construction of classification trees via the classification-hierarchy table, *Information and Software Technology*, 39(13), 1997, 889–896.

[9] T. Y. Chen and P. L. Poon, On the characteristics of a quality metric for classification trees, in *Proceedings of 1997 Australian Software Quality Conference* (New South Wales, Australia: Software Quality Association, 1997), pp. 69–82.

[10] T. Y. Chen and P. L. Poon, On the effectiveness of classification trees for test case construction, *Information and Software Technology*, 40(13), 1998, 765–775.

Appendixes

A Original Definitions Chen and Poon's Hierarchical Operators

(1) We define X to be a **loose ancestor** of Y, denoted by $X \Leftrightarrow Y$, if and only if the following conditions are satisfied:

(a) There exist some $x \in X$ and $y \in Y$ such that $class(X) = x$ and $class(Y) = y$ are part of a legitimate input.

(b) There exists some $x' \in X$ such that, for any $y' \in Y$, we cannot have $class(X) = x'$ and $class(Y) = y'$ in any legitimate input.

(c) There exists some $y'' \in Y$ such that, for any $x'' \in X$, we cannot have $class(X) = x''$ and $class(Y) = y''$ in any legitimate input.

(2) We define X to be a **strict ancestor** (or simply an **ancestor**) of Y, denoted by $X \Rightarrow Y$, if and only if the following conditions are satisfied:

(a) There exist some $x \in X$ and $y \in Y$ such that $class(X) = x$ and $class(Y) = y$ are part of a legitimate input.

(b) There exists some $x' \in X$ such that, for any $y' \in Y$, we cannot have $class(X) = x'$ and $class(Y) = y'$ in any legitimate input.

(c) There does not exist any $y'' \in Y$ such that, for any $x'' \in X$, we cannot have $class(X) = x''$ and $class(Y) = y''$ in any legitimate input.

(3) We define X to be **incompatible** with Y, denoted by $X \sim Y$, if and only if for any $x \in X$ and $y \in Y$, we cannot have $class(X) = x$ and $class(Y) = y$ in any legitimate input.

(4) We define X to have **other relations** with Y, denoted by $X \otimes Y$, if and only if X is neither a loose ancestor nor a strict ancestor of Y, and is not incompatible with it.

B Specification of *Examination Cancellation*

An active examination is classified into the types *RG* and *AT*. The cancellation reason for an active examination is captured in a pop-up window. The cancellation action is committed by clicking the "OK" button of the window. The "Cancel" button of the window aborts the action. The cancellation reason is mandatory for committing the action.

(a) Examinations of *RG* Type

The captioned pop-up window is shown as above.

(b) Examinations of *AT* Type

Before the captioned pop-up window is shown, the following steps of checking are went through

(b_1) If the link type of an *AT* examination is *PL* or *SU*, the examination cannot be cancelled. The possible link types for an *AT* examination are *PS*, *PL* or *SU*.

(b_2) If the report status of an *AT* examination is *pe* or *del*, the examination can be cancelled. The possible report status for an *AT* examination is *ed* (Edited), *en* (Endorsed), *pe* (Pending) and *del* (Deleted).

(b_3) Whether or not the data-capture work of an *AT* examination is completed, the examination can be cancelled. For *AT* examinations with completed data-capture work, a dialogue window is prompted to remind the status. Two options provided by the reminder are "Go" and "Stop". The cancellation action is halted after clicking the "Stop" button.

After going through the series of checking and satisfying the criteria as mentioned above, the cancellation reason window is shown.