Hitex White Paper

The Classification Tree Method

By Frank Büchner

hitex
DEVELOPMENT TOOLS

March 2002 - 001

Embedding Software Quality

Hitex GmbH
Greschbachstr. 12
D-76229 Karlsruhe
Germany

Tel: +49 721 9628 - 0
Fax: +49 721 9628 - 149

E-mail: Sales@hitex.de
Support@hitex.de

Internet: http://www.hitex.de

March 2002 - 001

Embedding Software Quality

Contents

Introduction

From problem definition to test case specification using the Classification Tree Method

Testing is a compulsory step in the software development process. However, the planning of such testing often raises the same questions:

- **How many tests should be run?**
- **What test data should be used?**
- **How can error sensitive tests be created?**
- **How can redundant tests be avoided?**
- **Have any test cases been overlooked?**
- **When is it safe to end testing?**

Anyone who has been confronted with such issues will be glad to know that the Classification Tree Method offers a systematic procedure to create test case specifications based on a problem definition.

1. The Classification Tree Method

1.1. The Problem Definition

The Classification Tree Method is applied to the definition of a (functional) problem.

Informally expressed, the solution to such problems requires a function to be executed, so that one can determine if the function yields the expected result or not.

Data processing software normally solves functional problems, since input data is processed according to an algorithm (i.e. the function) to become output data (i.e. the solution).

Here's an example of a functional problem definition:

> An initial value and a length define a range of values. Determine if a given value is within the defined range or not. Only integer numbers are to be considered.

1.2. Test Relevant Aspects

The first step in using the Classification Tree Method is to consider the problem's range of input data (i.e. all possible test cases) and all relevant test aspects. These aspects are then classified, i.e. the set of all possible values of an aspect are divided (completely and disjunctively) into classes.

So in our example, the input data range consists of all possible ranges of values that can be formed from integer numbers, combined with all possible test values, i.e. with all integer numbers.

The initial value and the length can be regarded as test relevant aspects. This is convenient since according to the problem definition, a range of values is defined by an initial value and a length.

It's convenient to classify if the test value is within the range of values or not using a "position" aspect.

So the three aspects to be used for classification are initial value, length and position and they thus form the basis of the so-called Classification Tree.

A particular problem definition can have completely different classifications, that are each relevant and usable.

2. Tool Support

To design the Classification Tree and for further use of the Classification Tree Method, a more reasonable approach is to use a tool that supports drawing the Classification Tree and the specification of test cases. The Classification Tree Editor CTE) has been especially created for this purpose.

The CTE comes complete with its own graphical editor that is intended specifically for drawing Classification Trees. Fig. 1 shows a simple Classification Tree for the problem definition described above. Three branches emerge from the root (i.e. the node "is_val_in_range"), which lead to the three classifications (rectangular nodes) "range_start", "range_length" and "position". These classifications are considered as base classifications for the given problem.

2.1. Forming Classes

Classes are now formed for the base classifications, where all possible values of an aspect are classified both completely and disjunctively. Since the initial value covers all integer numbers, it would be reasonable to form a class for positive values, a class for negative values and another class for the value zero. This class formation is useful since it ensures that a test case with a negative initial value is not overlooked. Classes are shown in the Classification Tree as frameless nodes. The branches, which represent the classes, emerge from the classification nodes (in this case: "range_start").

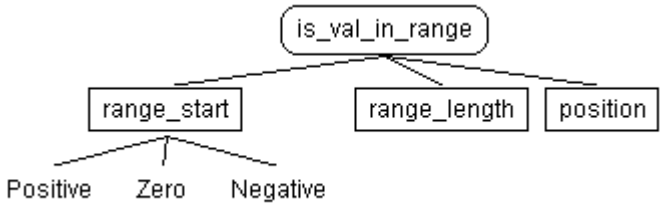

Fig. 1a A simple Classification Tree

2.2. A Systematic Approach

The same classes that were formed for the "range_start" classification can also be formed for "range_length." This means that a class for negative values is also implemented for the length. This is reasonable since the problem definition does not prohibit lengths from being negative values, even if negative lengths themselves are practically not necessary or even senseless. So, the systematic approach to specify test cases has revealed that the problem definition is insufficient. A clarification of the problem definition is probably the best way to remedy this issue. Note, a test case that includes a negative length would have most likely been overlooked if this systematic approach with the Classification Tree Method had not been adopted. Since negative values can occur in both the initial value as well as the length, a negative initial value combined with a negative length would be a valid input. A test case for such a combination would most likely be missing in a set of spontaneously selected test cases. Note, the problem definition described is a relatively simple one. Imagine how many important test cases could be overlooked if problems with dozens of aspects or input parameters are to be tested.

Two classes are formed for test values, so that in terms of “position” they are categorized as being either “inside“ or “outside“ the range (figure 1b).

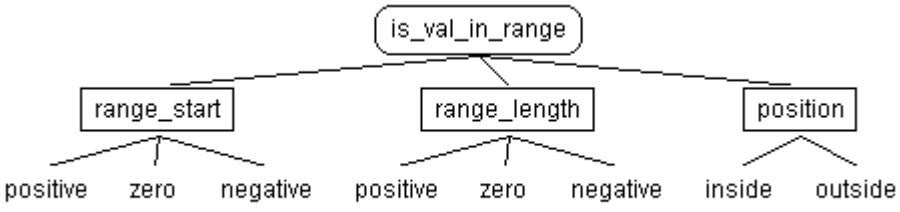

Fig. 1b The Classification Tree - further developed

2.3. Critical Values

It is generally recognized that during testing, critical values (also known as boundary values) of input parameters promise the most success if the objective is to generate malfunctions. This fact should also be taken into account in the Classification Tree. Hence in our example, a further classification can be introduced for the size (“size”) of initial values that are positive. The classes and quantity of classes introduced depends on the problem definition and last but not least on the rating of the person who creates the classification tree . The criterion “where is a problem assumed to exist?“ can serve as a guide here.

The problem definition itself usually provides clues for critical values. If the problem definition already mentions case distinctions (e.g. if the pressure exceeds 10 units, open the safety valve), then this would point to obvious critical values.

The current problem definition yields no clues whatsoever on critical values, therefore the "black box" approach cannot be used any further, since if a value of 5 for range_start produces the correct result, it cannot be assumed that using a value of 6 would produce incorrect results.

2.4. Error Sensitive Test Cases

Beginning with the assumed implementation (i.e. using the "white box" approach), an interesting question arises if we consider an initial value that is very large. What happens if a range begins with an initial value that is the largest possible positive value and furthermore, the range has a positive length? Would some kind of "wrap around" then take place and would a very small test value then be incorrectly considered as being “in range?“ Or, does the program simply crash? Based on these considerations, positive initial values can be either placed in a class with the largest possible positive value or in another class for all other values.

A third possible class would be a class with the smallest possible positive integer value, i.e. the number one. In our example, the smallest positive whole number would be classified as a “normal value“ and there is no valid reason for it to have its own class. This is an arbitrary decision made by the creator of the classification tree. The example described explains the idea behind the Classification Tree Method, which is: detailed consideration of the problem definition and a systematic approach leads to the determination of error sensitive test case specifications and avoids the specification of redundant test cases.

2.5. Risk Analysis

A risk analysis of the problem would usually also help to find test relevant aspects and therefore form test case specifications. If a malfunction has particularly grave consequences when it occurs under certain conditions, then is it important that tests are carried out under these same conditions.

The main objective here is not for these test cases to expose malfunctions with a high probability of occurrence, but rather to ensure that no malfunctions occur under these particular conditions.

2.6. Classes for Test values

As already mentioned, two classes are created to categorize test values: position "inside" the range and position “outside” the range. If a test value lies outside the range, it is useful for the test if we then further classify if the test value is “below” or “above” the range. The consideration of critical values leads to a further classifcation of test values that are located in the immediate vicinity of the range limits.

The complete classification tree is shown in figure 2.

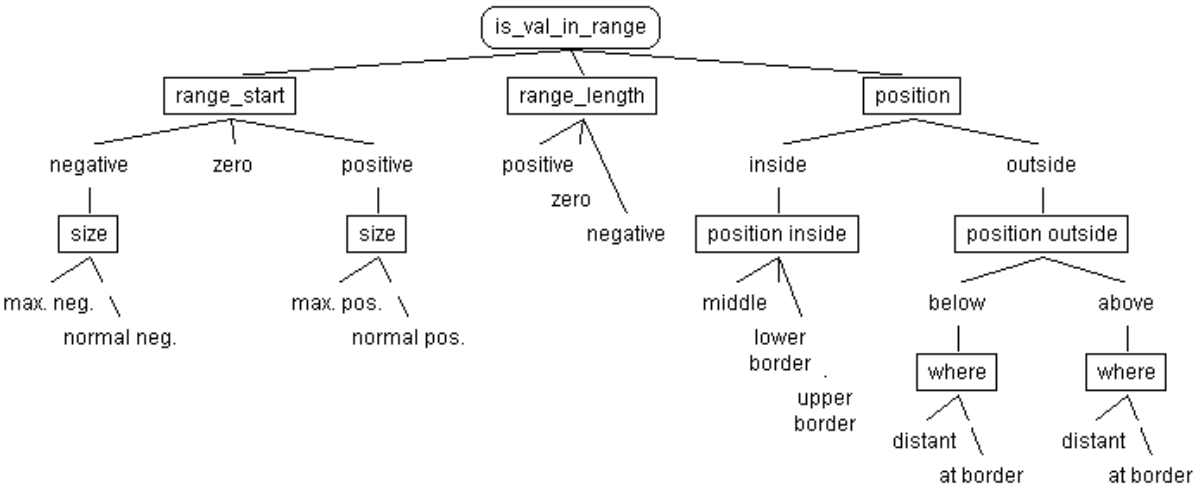

Fig. 2 The complete classification tree

2.7. Separating Specification from Data

A Classification Tree specifies test cases, but it does not specify test data. Although we have determined that a test case with a “normal“ positive initial value can exist, we have not determined which concrete value is to be actually used later on in the test.

Expressed in another way, the creation of test case specifications using the Classification Tree Method does not include a means of arriving at concrete test values from the terms in the Classification Tree ("normal" and "positive"). This may appear strange in the given case, since “zero“ has an obvious concrete value. However, for general problem definitions, concrete values are not so obvious. For example, if a problem definition leads to a class for “green triangles,“ what concrete value should be used for “green?“ However, technical issues are not the deciding factor.

The abstraction of classes from concrete test data is a deliberate methodical means of making test ideas explicit. Thus the implementation of a test case specification into concrete test data is a separate procedure that can for example be performed by the Test Data Editor of the test tool Tessy [1]. Due to the separation of test case specification and test data selection, it is not absolutely necessary for the developer of the software to create the test case specifications. This is not only desirable due to the higher probability of finding errors, but it also allows both tasks to be performed in parallel, leading to earlier completion.

2.8. Test Case Specification

If the Classification Tree has been created, the next task is to specify suitable test cases. A test case specification results from the combination of classes, depicted as leaves in the classification tree.

Classes that belong to a classification are of course non combinable, since by definition, classes are disjunctive and no representative can be found which would belong to several classes of one classification. In our example, it's not possible for a test value to be simultaneously inside and outside the range. Expressed in another way: a leaf class is selected for each of the initial test relevant aspects (i.e. for each base classification).

In the CTE, a test case specification comprises of a line that is made up of test case descriptions and markers in the combination table. The combination table is located in the CTE, beneath the Classification Tree. The desired classes are selected by simply setting the markers in the combination table. The CTE supports this selection process by automatically ensuring that only combinable classes are selected for a test case.

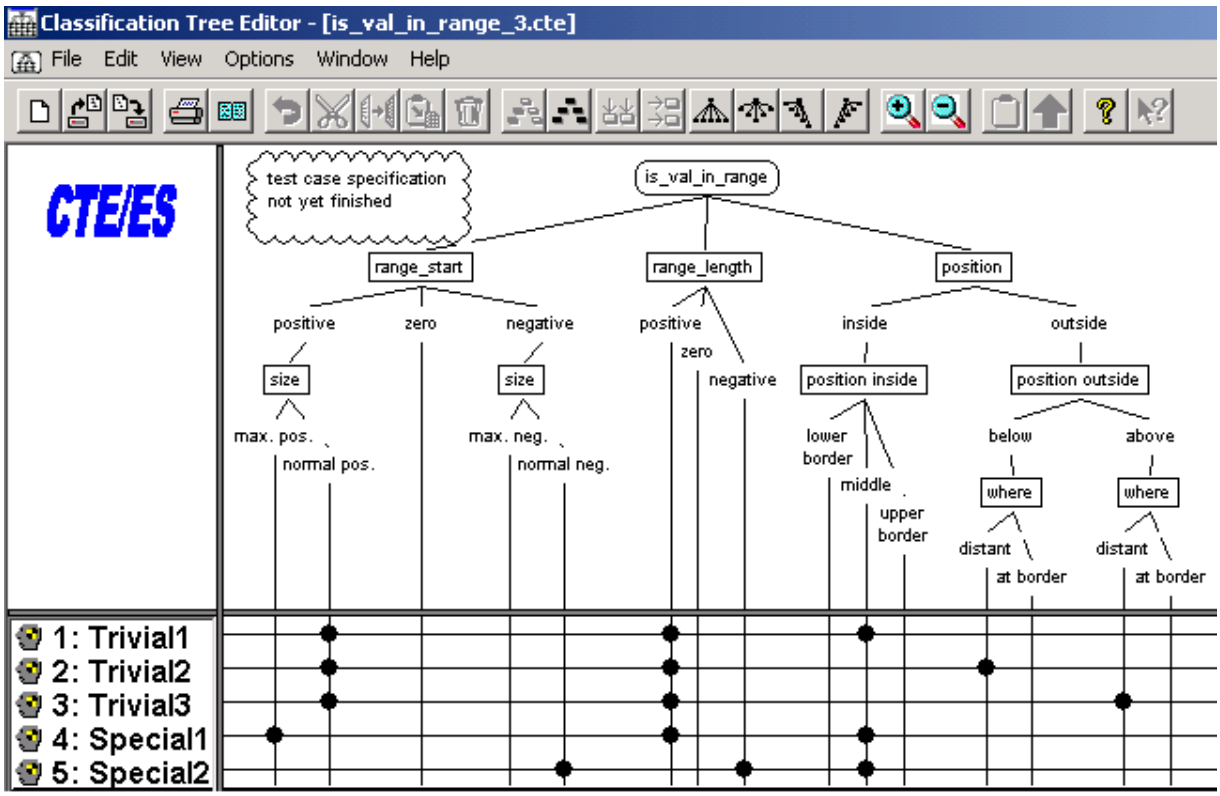

Fig. 3 Classification Tree and some test case specifications

2.9. Test Coverage

The number of test case specifications and thus the scope of a test remain in principle for the user to decide. However, based on the Classification Tree, it's possible for some values to be determined that provide clues to the number of test cases required.

The first value is the number of test cases, if each leaf class is included at least once in a test case specification. This number is known as the minimum criterion. Since leaf classes of the same base classification cannot be combined, the minimum criterion is the largest number of leaf classes that belong to a base classification. In our example, the largest amount of leaf classes, namely seven, belong to the base classification "position." Seven is thus the value of the minimum criterion.

The maximum criterion is the number of test cases that results when all permitted combinations of leaf classes are considered. In our example, the maximum criterion amounts to 105 (i.e. 5 * 3 * 7).

A reasonable number of test case specifications obviously lies somewhere between the minimum and maximum criterion. As a rule of thumb, the total number of leaf classes gives an estimate for the number of test cases required to get sufficient test coverage.

The objective of the Classification Tree Method is to determine a sufficient but minimum number of test case specifications. So generally speaking, it is not necessary to specify a test case for each possible combination. In fact, the Classification Tree Method should enable the user to use well-designed specifications, thus reducing the number of tests. The Classification Tree provides the necessary overview for this. In practical applications, this reduction of test cases is essential, since the maximum criteria can easily run into very high numbers. Furthermore, the required expenditure for running automatic tests to their full extent in comparison to the benefits they provide is excessive.

The wish of some users to run tests not only with one representative of a class, but rather with all possible representatives (in combination with all possible representatives of all other combinable classes), fails due to the resulting astronomical number of test cases – this becomes apparent even using our very basic example.

2.10. More About the CTE

The main objective of the CTE is to comfortably support use of the Classification Tree Method. This includes on the one hand drawing and editing a Classification Tree. Here sub-trees can give an improved overview, descriptions and commentary can be added to help to improve documentation, automatic layout of the tree always results in a clearer representation after modifications, elements of the tree can be copied and repositioned and parts of the classification tree can be stored in libraries and be reused later in other classification trees. On the other hand, the CTE also aids the management of test case specifications. Test case specifications can be provided with commentary and can be combined into test sequences, which is necessary for the description of dynamic processes.

The CTE can also verify the Classification Tree and test case specifications. This would for example reveal incomplete tree sections or unused classes. Furthermore, it's possible to compile a statistical evaluation, perhaps of the number of different tree elements. This leads to an estimation of the necessary test expenditure.

The compiled information can be exported in various file formats, which aids the transfer of test case specifications to other tools as well as documentation.

All in all, the CTE provides all the functionality required to make efficient use of the Classification Tree Method.

A concrete example of an extensive application of the Classification Tree Method and the CTE is the test case specification for the conformance testing of operating systems according to the OSEK/VDX specification [2].

The CTE is an integral part of Tessy [1], which is a tool to automate the testing of embedded software. Of course, it's possible to export test case specifications from the CTE to Tessy. Since the CTE's application area is not limited to the testing of embedded software, it is also available as a separate product.

CTE and Tessy both originate from DaimlerChrysler's software technology research laboratory.

3. The Classification Tree Method in the Development Process

The creation of test case specifications according to the Classification Tree Method using the CTE can, and should, be created independently from the implementation. This would ideally take place before the implementation stage and should be performed by someone other than the software developer.

The Classification Tree and test case specifications remain easy to understand thanks to graphical representation and commentary and they can also be subject to review procedures.

Information generated by the CTE according to the Classification Tree Method documents the test coverage and thus contributes to the conformity of development processes according to various quality standards (Bootstrap, Spice, CMM).

Due to the systematic approach and the compulsion to consider all test aspects when applying the Classification Tree Method, there is a high probability that the problem definition will be correctly represented in test cases. However, because of human participation in this transfer process, there is no complete assurance that this will be the case.

The size of a Classification Tree can be a measure of a problem's complexity. The number of test cases deemed essential by the Classification Tree Method forms on the one hand a good measure of the required test expenditure and can on the other hand also serve as an estimation of the required implementation expenditure.

However, COCOMO and function points are methods that are better suited to expenditure estimation. Note, a large number of test cases does not automatically guarantee sufficient test coverage. This depends more on being able to produce test cases that are error sensitive and being able to avoid those that are redundant.

4. Conclusion

The Classification Tree Method has even in the case of our simple example allowed us to derive test cases that would have more than likely been overlooked if test case specification was performed spontaneously.

The CTE provides an overview of specified test cases and thus allows redundant test cases to come to light and the presence of error sensitive test cases to be verified. Furthermore, the documentation of specified test cases aids quality in the software development process.

Glossary

Classification Tree Method

A systematic procedure used to determine a set of test case specifications.

Test relevant aspect

A test relevant criterion according to which classification takes place.

Leaf Class

A class that is not further divided into sub-classes.

Test case specification

A selection of combinable leaf classes of the Classification Tree together with a description.

Test case

A test case specification with concrete test data.

Classification Tree

A diagram showing the iterative classification of test relevant aspects.

Classification Tree Editor (CTE)

A tool that supports applying the Classification Tree Method.

Black box Test

A test based on a problem definition, without consideration of the implementation.

White box Test

A test based on an implementation.

References

[1] http://www.hitex.de/perm/tessy.htm

[2] http://www.osek-vdx.org/mod_20/ostestplan20.pdf