hitex
DEVELOPMENT TOOLS

Hitex Germany
– Head Quarters –
Greschbachstr. 12
76229 Karlsruhe
Germany

☎ +049-721-9628-0
Fax +049-721-9628-149
E-mail: Sales@hitex.de
WEB: www.hitex.de

Hitex UK
Warwick University
Science Park
Coventry CV47EZ
United Kingdom

☎ +44-24-7669-2066
Fax +44-24-7669-2131
E-mail: Info@hitex.co.uk
WEB: www.hitex.co.uk

Hitex USA
2062 Business Center Drive
Suite 230
Irvine, CA 92612
U.S.A.

☎ 800-45-HITEX (US only)
☎ +1-949-863-0320
Fax +1-949-863-0331
E-mail: Info@hitex.com
WEB: www.hitex.com

Embedding Software Quality

White Paper

Test Case Design Using the Classification Tree Method

The aim of the Classification Tree Method is to derive a set of test case specifications starting from a functional problem specification.

Product:	TESSY
Author:	Frank Buechner
Revision:	02/2009 – 003

Preface

In order to keep you up-to-date with the latest developments on our products, we provide White Papers containing additional topics, special hints, examples and detailed procedures etc.

For more information on the current software and hardware revisions as well as our update service, please visit www.hitex.de, www.hitex.co.uk or www.hitex.com.

Contents

1 Introduction 4

2 The Classification Tree Method 5
2.1 Objective 5
2.2 A Human Being Applies It 5
2.3 General Method 5
2.4 Origin of the Method 5

3 Applying the Classification Tree Method 6
3.1 Overview: Steps to Take 6
3.2 Step 1: Drawing the Tree 6
3.2.1 Starting Point: Functional Problem Definition 6
3.2.2 Determining the Test-Relevant Aspects 6
3.2.3 Classifying the Values of a Test-Relevant Aspect 7
3.2.4 Repeating Equivalence Partitioning 8
3.2.5 Result: Classification Tree 9
3.2.6 Using Boundary Values 10
3.2.7 Testing a Hysteresis 10
3.3 Step 2: Specifying Test Cases 11

4 Example "is_value_in_range" 12
4.1 Problem 12
4.2 Test-Relevant Aspects 12
4.3 Forming Classes 13
4.4 Moving On Systematically 13
4.5 A First Range Specification 14
4.6 A Second Range Specification 14
4.7 Extending the Tree by a Boundary Class 15
4.8 Another Interesting Test Case Specification 16
4.9 The Completed Classification Tree 17
4.10 The Completed Test Case Specification 19
4.11 Another Test Case Specification 20

5 Tool Support 22
5.1 Classification Tree Editor CTE 22
5.1.1 The Drawing Area 22

5.1.2 The Combination Table 23
5.1.3 General 24
5.2 Tessy 25

6 More on the Classification Tree Method 26
6.1 Composition 26
6.2 Sequences of Test Case Specifications 27
6.3 Coping with Big Trees 28
6.3.1 Refinements 28
6.3.2 Simplifying Specifications 28
6.4 Separating Specification from Data 29
6.4.1 The Difference between Abstract and Parameterized 29
6.4.2 Parameterization 30

7 Example "Ice Warning Indication" --- Continued 32
7.1 Where We Left Off 32
7.2 Hysteresis Using a State 33
7.3 Hysteresis Using a Sequence 34
7.4 Hysteresis Using a Sequence and Boundary Values 35

8 Advantages of the Classification Tree Method 36
8.1 Visualizes Testing Ideas 36
8.2 Gives Confidence 36
8.3 Reduces Complexity 36
8.4 Checks Problem Specification 36
8.5 Estimates Testing Effort 36

9 Literature 37

1 Introduction

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 (CTM) offers a systematic procedure to create test case specifications based on a problem definition.

2 The Classification Tree Method

2.1 Objective

The objective of the Classification Tree Method (CTM) is to transform a (functional) definition of a problem systematically into a set of error-sensitive, low redundancy set of test case specifications.

The systematic approach yields a high probability that the resulting set of test specifications is complete, i.e. that no relevant tests are overlooked. Naturally, correct usage of the method and an appropriate integration in the development process are prerequisites.

Having a complete set of tests gives evidence when it is safe to end testing.

2.2 A Human Being Applies It

The Classification Tree Method (CTM) is applied by a human being. Therefore, the outcome of the method depends on the experiences, reflections, and appraisals of this human being, i.e. of the user of the CTM. Most probably two different users will come out with a different set of test case specifications for the same functional problem.

However, both sets could be considered to be "correct", because there is no absolute correctness. (It should be clear that there are set of test cases that are definitively wrong or incomplete). Because of the human user, errors cannot be avoided. One remedy is the systematic inherent in the method. This systematic guides the user and stimulates his creativity. The user shall specify test cases with a high probability to detect a fault in the test object. Such test cases are called "error-sensitive" test cases. On the other hand, the user shall avoid that too many test cases are specified, that are superfluous, i.e. do not increase test intensiveness or test relevance. Such test cases are called "redundant" test cases.

It is advantageous, if the user is familiar with the field of application the method is applied in.

2.3 General Method

The Classification Tree Method is a general method, i.e. it can not only be applied to module/unit testing of embedded software, but to software testing in general and also to functional testing of problems, that are not software related. The prerequisite to apply the method is to have available a functional specification of the behaviour of the test object.

The CTM incorporates several well-known approaches for test case specification, e.g. equivalent partitioning, and boundary value analysis.

2.4 Origin of the Method

The Classification Tree Method stems from the former software research laboratory of Daimler in Berlin, Germany.

3 Applying the Classification Tree Method

3.1 Overview: Steps to Take

Step 1: Drawing the classification tree by determination of the test relevant aspects and the values they can take.

Step 2: Specifying test cases by selection of leaf classes of the classification tree.

Sometimes there is an (optional) intermediate step: Assigning values to classes. This step is only possible when Tessy is involved. See section Parameterization on p. 30.

3.2 Step 1: Drawing the Tree

3.2.1 Starting Point: Functional Problem Definition

The Classification Tree Method is applied to the (functional) specification of a test object. This can be thought of as a description of the expected behaviour of the test object, hence functional. "If the button is pushed, the light will go on; if the button is released, the light will go off" is a very simple example of a functional specification for the behaviour of a test object.

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).

3.2.2 Determining the Test-Relevant Aspects

The CTM starts by analysing the functional specification. This means, the human user of the method thinks about this specification with the objective to figure out the so-called test-relevant aspects of the specification at hand. An aspect is considered relevant if the user expects that aspect to influence the behaviour of the test object during the test. In other words, an aspect is considered relevant if the user wants to use different values for this aspect during testing. To draw the tree, these aspects are worked on separately. This reduces the complexity of the original problem considerably, what is one of the advantages of the CTM.

3.2.2.1 Example for an Test-Relevant Aspect

Consider systems that measures distances in a range of some meters, e.g. the distance to a wall in a room. Those systems usually send out signals and measure the time until they receive the reflected signal. Those systems can base on two different physical effects: One can use sonar to determine the distance, whereas the other can use radar.

The question is now: Is the temperature of the air in the room a test relevant aspect for the test of these measurement systems? Quite astonishingly at first glance, the answer is "yes" for one system and "no" for the other. However, you soon reckon that the speed of sound in air (sonar) is dependent on the temperature of the air. Therefore, to get exact results, the sonar system (hopefully) takes this temperature into account during the calculation of the distance. To test if this is working correct, you have to do some tests at different temperatures. Therefore, the temperature is a test-relevant aspect for the sonar system. On the other hand, we all know that the speed of a radar signal, that travels at

the speed of light, is independent from the temperature of the air it travels in (it didn't even need air to travel). Therefore, the temperature of the air is not a test-relevant aspect for the testing of the radar system, i.e. it would be superfluous to do testing at different temperatures.

This example shows that it needs careful thinking to figure out (all) test relevant aspects. I.e. it would lead to poor testing if someone simply takes the test cases for the radar system and applies them to the sonar system without adding some temperature-related test cases. Additionally, this example illustrates that it is advantageous to have some familiarity with the problem field at hand when designing test cases.

3.2.3 Classifying the Values of a Test-Relevant Aspect

After all test relevant aspects are determined, the values that each aspect may take are considered. The values are divided into classes according to the equivalence partitioning method. Values are assigned to the same class, if the values are considered equivalent for the test. Equivalent for the test means that if one value out of a certain class causes a test case to fail and hence reveals an error, every other value out of this class will also cause the same test to fail and will reveal the same error.

In other words: It is not relevant for testing which value out of a class is used for testing, because they all are considered to be equivalent. Therefore, you may take an arbitrary value out of a class for testing, even the same value for all tests, without decreasing the value of the tests. However, the prerequisite for this is that the equivalence partitioning was done correctly, what is in the responsibility of the (human) user of the CTM.

Please note:

- Equivalent for the test does not necessarily mean that the result of the test (e.g. a calculated value) is the same for all values in a class.
- Equivalence partitioning must be *complete* in mathematical sense: Every possible value of a test relevant aspect must be assigned to a class.
- Equivalence partitioning must be unique in mathematical sense: A value of a test relevant aspect must be assigned to a single class, and not to several classes.

3.2.3.1 Example for Equivalence Partitioning: Ice Warning

As example, we take the following functional problem specification:

> An ice warning indication in the dashboard of a car shall be tested. This ice warning indication depends on the temperature reported by a temperature sensor at the outside of the car. This sensor can report temperatures from -60 °C to +80 °C. At temperatures above 3 °C the ice warning shall be off, at lower temperatures it shall be on.

It is obvious that the temperature is the only test-relevant aspect. To have an reasonable testing effort, we do not want to have a test case for every possible temperature value. Therefore, all possible temperature values need to be classified according to the equivalence partitioning method. It is an important "best practise" to find out if invalid values may be possible, when applying equivalence partitioning. In our case a short circuit or an interruption of the cable could result in an invalid value. Therefore, we should divide the temperature values in valid and invalid values first. The invalid values can relate to temperatures that are too high (higher than 80 °C) and to temperatures that are too low (lower than -60 °C). Further it is tempting to form two classes out of the valid temperatures: The first class shall contain all the values that result in the ice warning display being on (i.e. from -60 °C to 3 °C) and the other class shall contain all values that result in the ice warning display being off (i.e. from 3 °C to 80 °C). This is depicted in the picture below.

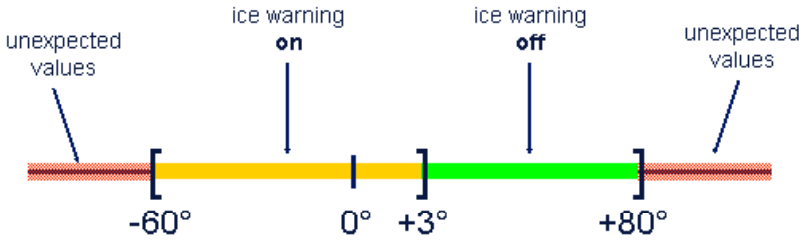

Fig. 1 Initial equivalence partitioning for "ice warning"

The equivalence partitioning in the figure above leads to at least four test cases, because we need to take a value out of each class for the tests.

3.2.4 Repeating Equivalence Partitioning

An equivalence class can be sub-divided according to additional aspects. This equivalence partitioning on several levels reduces the complexity of equivalence partitioning, because you can consider each class isolated from the other classes and decide, if (and how) it needs to be sub-divided or not. Furthermore, this equivalence partitioning on several levels documents the thoughts resp. stages of work until the final equivalence partition. This serves understandability and traceability of the result. Also it allows easily reverting steps if the final equivalence partition has become too fine granulated.

3.2.4.1 Example for Repeated Equivalence Partitioning

For the example "ice warning", the classification of the valid values is not detailed enough (in my opinion), because according to the equivalence partitioning method, it would be sufficient to use a single, arbitrary value out of a class for all the tests. This could be for instance the value +2 °C out of the class of temperatures, for which the ice warning display is on. In consequence, no test with a minus temperature would check if the ice warning display is on. To avoid this consequence, you could divide this class further according to the sign of the temperature. This reflection could result in a classification tree according to figure below.

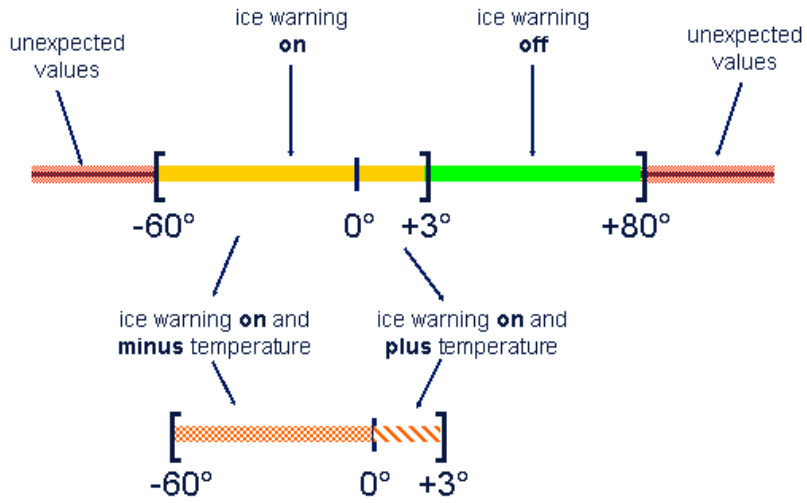

Fig. 2 Repeated equivalence partitioning for "ice warning"

3.2.5 Result: Classification Tree

Using the CTM, the result of the repetition of equivalence partitioning (for all test relevant aspects) is depicted in the so-called classification tree. The root of the tree is at the top, it grows downwards. The root represents the functional problem; the test relevant aspects depart from the root. Test relevant aspects (also called classifications) are drawn in nodes depicted by rectangles. Classes are shown in the classification tree as frameless nodes. The branches, which represent the classes, emerge from the classification nodes.

3.2.5.1 Classification Tree for the Example "Ice Warning"

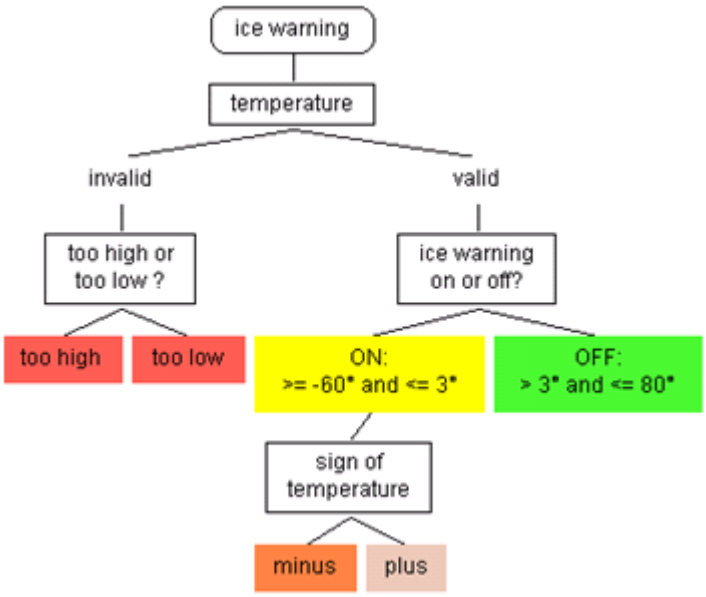

Fig. 3 A possible classification tree for "ice warning"

3.2.6 Using Boundary Values

The idea behind using boundary values for test cases is that values at the borders of a range of values are better suited to form error-sensitive test cases that values in the middle.

The idea behind boundary values analysis is contrary to equivalence partitioning, because one method takes a set of values as equivalent and the other method prefers special values in such a set.

Despite the fact that the idea behind boundary values analysis is exactly the opposite of equivalence partitioning, both approaches can be expressed in the CTM. An example for it is given in section Extending the Tree by a Boundary Class (p. 15) and in section The Completed Classification Tree (p. 17) (see p. 18 and also section Hysteresis Using a Sequence and Boundary Values on p. 35).

3.2.7 Testing a Hysteresis

The current problem specification of the "ice warning" example does not mention hysteresis. However, it may be tempting to extend the current problem specification in that fast changes in the state of the ice warning display shall be avoided. For instance, the ice warning display shall be switched off only after the temperature has risen to more than 4 °C. This could be realized by a hysteresis function.

The necessary test cases for such a hysteresis function can be specified by the CTM. An example for it is given in section Hysteresis Using a State (p. 33) et seqq.

3.3 Step 2: Specifying Test Cases

Using the CTM, test cases are specified in the so-called combination table below the classification tree. The leaf classes of the classification tree form the head of the combination table. A line in the combination table depicts a test case. The test case is specified by selecting leaf classes, from which values for the test case at hand shall be used. This is done by the human user of the method, by setting markers in the line of the respective test cases in the combination table.

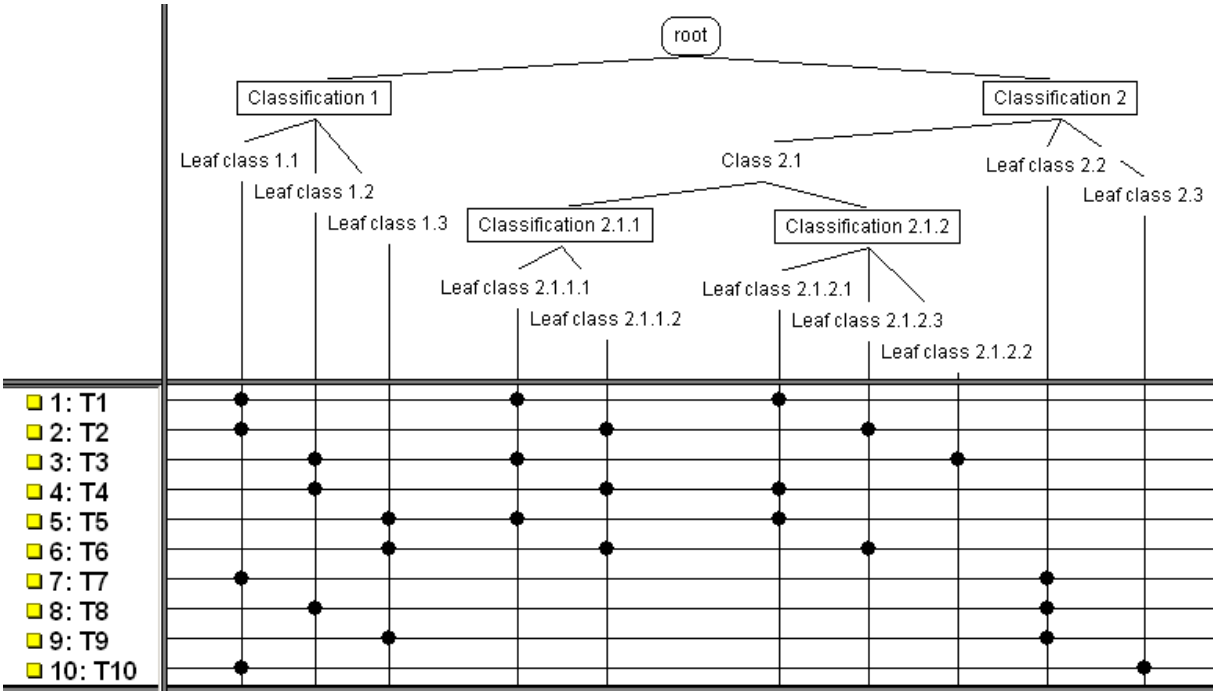

Fig. 4 Result of the CTM: Classification tree (above) with combination table (below) (from [4])

It may be tempting to combine every class with every other class during the specification of the test cases. Besides the fact, that not every combination might be possible for logical reasons, it is not the intention of the CTM to do so, despite the fact that it could be done automatically by a tool. This would lead to many test cases, with the disadvantages of (a) loss of overview and (b) too much effort for executing the test cases.

The objective of the CTM is to find a minimal, non-redundant (but sufficient) set of test cases by trying to cover several aspects in a single test case, whenever possible. Similar to the drawing of the tree, it depends on the appraisal and experience of the (human) user of the method, how many (and which) test cases are specified. Obviously the size of the tree influences the number of test cases needed: A tree with more leaf classes naturally results in more test cases than a tree with less leaf classes. The number of leaf classes needed at least for a given tree is called the "minimum criterion". It can be calculated from the consideration that each leaf class should be marked in at least one test case, and that some leaf classes cannot be combined in a single test case, because the classes exclude each other. Similar a "maximum criterion" can be calculated, which gives the maximal number of test cases for a given classification tree. A rule of thumb states that the number of leaf classes of the tree gives the order of magnitude for the number of test cases required for a reasonable coverage of the given tree.

4 Example "is_value_in_range"

4.1 Problem

Here is a very basic example of a functional problem definition:

> A start 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.

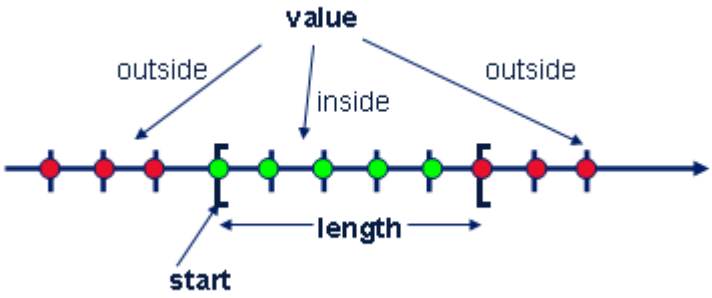

Fig. 5 The problem "is_value_in_range" depicted graphically

It is obvious, that exhaustive testing is practically impossible, because we get 65536 * 65536 * 65536 = 281.474.976.710.656 test cases, even if we assume only 16 bit integers. If we would assume 32 bit integers … well, we better do not.

4.2 Test-Relevant Aspects

The start of the range 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 a start value and a length. Furthermore, it reflects the intention to use different values for the start and the length during testing, what sounds reasonable.

Furthermore, we should have some test cases, which result in "inside", and other test cases which result in "outside". We call the corresponding aspect "position", because the position of the value under test with respect to the range determines the result.

So the three test-relevant aspects to be used for classifications are initial value, length and position and they thus form the basis of the classification tree.

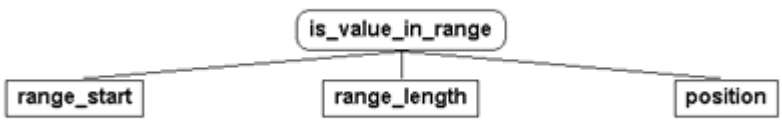

Fig. 6 The initial classification tree with three test-relevant aspects

The initial classification tree features three test-relvant aspects, which form the three initial classifications. They are depicted in the rectangular nodes emerging from the root node ("is_value_in_range").

4.3 Forming Classes

Now classes are formed for the base classifications according to the equivalence partitioning method. Usually, the problem specification gives us hints how to form the classes. E.g. if the problem specification would state "if the start value is greater than 20, the length value doubles", we should form a class for start values greater than 20 and a class for start values smaller than resp. equal to 20.

Unfortunately, the problem specification at hand is too simple to give us similar hints. However, since the start value can take on 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. (It would also be reasonable to form just two classes, e.g. one class for positive start values including zero and the other class for negative start values. This depends on ones emphasis having zero as value for the start of the range in a test case or not.)

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").

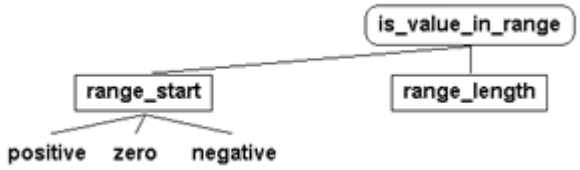

Fig. 7 The classification tree for "is_value_in_range" - further developed

In the above tree the classification "position" is omitted. It will be elaborated later.

4.4 Moving On Systematically

Because of the systematic inherent in the CTM, and because "range_length" is an integer as well as "range_start", it is stringent to use for "range_length" the same classes as for "range_length". This results in the following tree.

Fig. 8 The classification tree for "is_value_in_range" - further developed

Because we have omitted "position" in the above tree, currently the bottom of the tree consists of leaf classes only. This allows us specifying a first test case, or better a range to be used in the first test case.

4.5 A First Range Specification

To specify a first range (to be used In the first test case), we have to insert a line in the combination table and to set markers on that line.

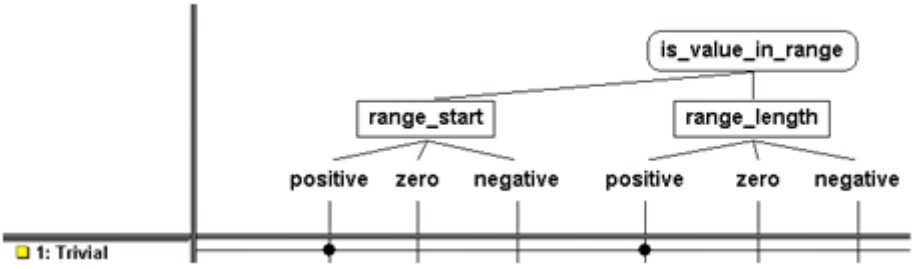

Fig. 9 A first specification for the range in the combination table

Two markers are set on the line for the first specification. One marker selects the class "positive" for the start of the range; the other marker selects the class "positive" for the length of the range. Hence a range with the start value of, say, 5 and a length of, say, 2 would be according to this specification. This is not remarkable; such a value pair most probably would have been used in any case when test cases for the problem at hand are thought of. Therefore, this first specification was named "Trivial".

However, with the same tree as above, a much more interesting test case range can be specified.

4.6 A Second Range Specification

We can insert a second line in the combination table and specify a much more interesting tests case.

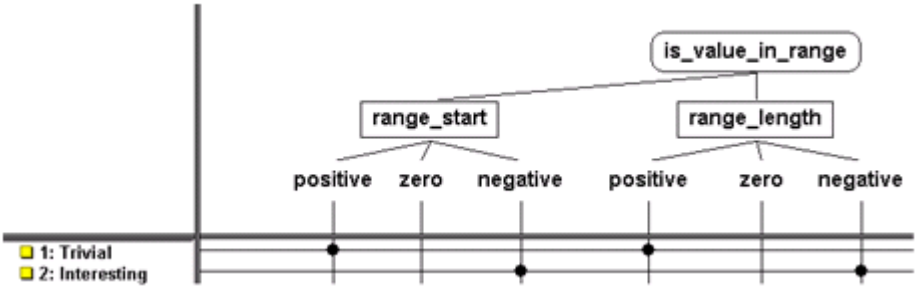

Fig. 10 A second specification for the range in the combination table

For the second specification, again two markers are set. They specify that a negative value shall be used both for the start of the range and for the end of the range. Hence a range with the start value of -5 and a length of -2 would be according to the second specification. But this value pair raises some questions: Shall the value -6 lie inside the range? Or shall the value -4 lie inside the range? Or shall no value at all lie inside the range, if the length of the range is negative? Each opinion has its supporters and it is hard to decide what is to be considered "correct". Actually, at this point it is out of our competence to decide what is correct. We have found a problem of the specification!

It is important to note that it is a valuable result to find a problem (omission or contradiction) in the functional problem specification; and that it was achieved in the case at hand during test case specification for the functional problem; and it is generally more likely to detect a problem in the functional specification if the test case specification is systematic; and that the CTM is a systematic method for test case specification. Hence, the CTM provides good means to detect problems in the functional problem specification.

Probably a test case using a negative length would not have been used if the test case specification would have been done spontaneous and non-systematic. But a negative length is completely legal for the functional problem specification that was given above. If you consider that the problem specification at hand was a very simple one, you may imagine how likely it is to overlook a problem in a more comprehensive and complicated problem specification.

4.7 Extending the Tree by a Boundary Class

In case we are not satisfied with the fact that a fixed single positive value, e.g. 5, may serve as value for the start of the range in all test cases, we can sub-divide the class "positive" according to a suitable classification. In our example, we classify according to the size. The idea behind this is to have a class containing only a single value, in our case the highest positive value existing in the given integer range. We use this value because it is an extreme value, and as we know, using extreme values (or boundary values) in test cases is well-suited to produce error-sensitive (or interesting) test cases.

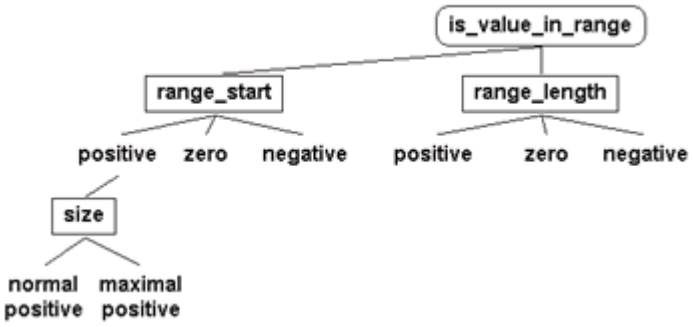

Fig. 11 A class for an extreme value for the start of the range is introduced

In the figure above, the positive values for the start of the range are subdivided according to their size. This results in the two classes "normal positive" and "maximal positive". The class "maximal positive" holds the highest possible positive value (i.e. MAX_INT), and the class "normal positive" holds all other positive values. This satisfies mathematical completeness.

Remark 1

Another possibility to classify the positive start values would have been for instance to classify in odd and even values. This would have been completely legal. This would have been probably also sensible for e.g. a problem of number theory, but not target-oriented for the problem at hand.

Remark 2

Please note that for the moment we do not know and we need not to know the size (in bits) of the integers used in the problem at hand. We simply specify "the highest positive value in the given integer range". This keeps our test case specification abstract! I.e. our test case specification is appropriate for any integer size. As soon as we assume we use, e.g. 16 bit integers, and therefore parameterize our test case by specifying 32767 as value in the class "maximal positive", we loose this abstraction. I.e. if we port the parameterized test case to a, say, 32 bit integer system, the test case looses its sense. This is not the case if we port the abstract test case specification.

4.8 Another Interesting Test Case Specification

With the classification tree extended according to the figure above, we can insert an additional line in the combination table and specify again an interesting range for a third test case.

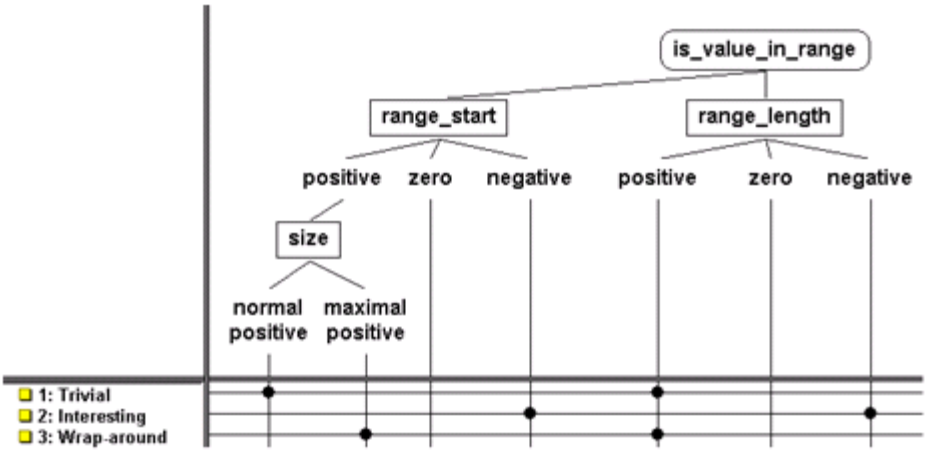

Fig. 12 The third range specification provokes a wrap-around

The third range specification in the figure above combines the highest positive number for the start value of the range with a positive length, i.e. the range exceeds the given integer range.

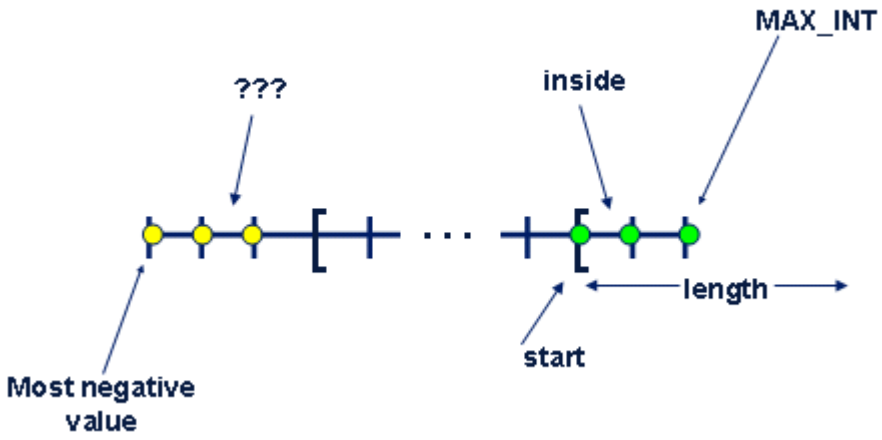

Fig. 13 The third range exceeds the given integer range

The situation with the third range specification is similar to the situation depicted in the figure above. The situation raises some questions: Will the situation be handled sensible and gracfully by the test object? Or will it crash due to the overflow? Will the negative values on the left hand side in the figure above (depicted in yellow) be accounted to lie inside the range or not? And what is correct with respect to the last question? The problem specification above does not give an answer to the latter question, i.e. again we have found a weak point in the problem specification.

To sum up, designing test cases according to the classification tree method has revealed two problems of the problem specification and has lead to interesting test cases so far.

4.9 The Completed Classification Tree

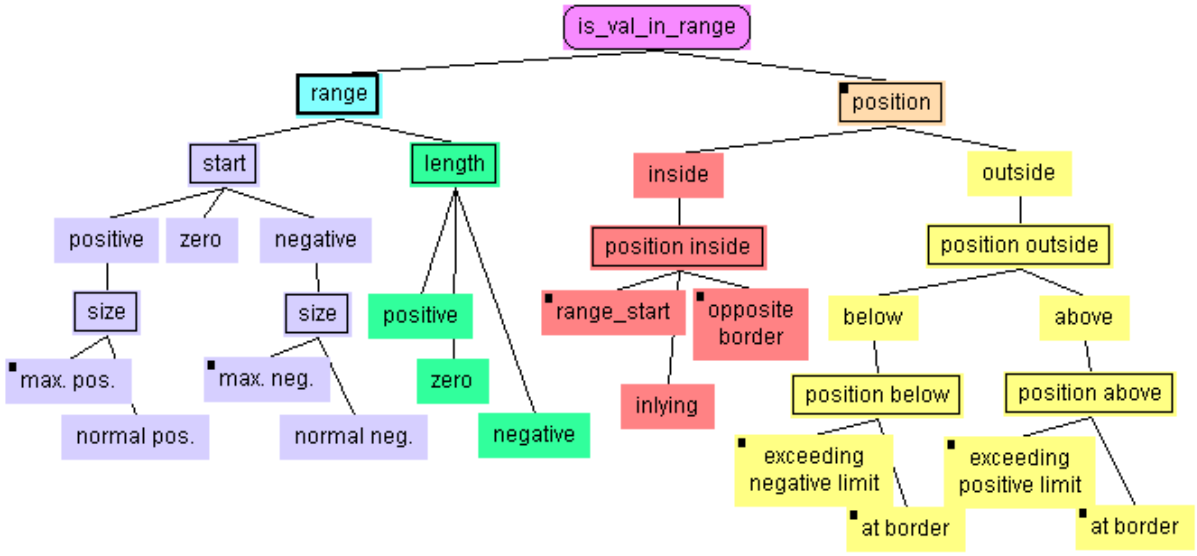

Fig. 14 The completed classification tree

In the figure above, one possible completed classification tree is depicted. This tree is discussed in the following.

- Analogous to having a class "maximal positive" for the start value of the range, a class "maximal negative" is introduced for the negative start values of the range. The idea behind this class is to combine the maximal negative start value with a negative length of the range, what shall provoke an underflow or negative wrap-around. This idea comes from the systematic in the CTM: If a positive wrap-around is seen as an interesting test case, also a negative wrap-around should be exercised.
- An example for a composition is given by "range". A composition may be used for a relation "consists of". In our case, the range consists of a start value and a length. Compositions are depicted by rectangles with thicker borders than rectangles for classifications. For more on compositions: See section Composition on p. 26.
- The final tree features still the three initial classes "positive", "zero", and "negative" for the length of the range. It is important to note that the tree reveals at a glance that nothing like "maximal positive length" or similar is considered to be useful for the testing problem at hand.
- Now we catch up the discussion of the classification "position" introduced in Fig. 6 (p. 12), but not elaborated until now. It is obvious that a position can either be inside or outside the range, hence this classification suggests itself. Furthermore, it is obvious that there are two different areas outside the range: below the range and above the range. This is reflected in the classification "position outside". (If the tree would miss such a classification, it may well be considered "incorrect").
- The class "inside" of the classification "position" could well be a leaf class of the classification tree. However, in the classification tree in the figure above, this class is subdivided further in the sub-classes "range_start", "opposite_border", and "inlying". This is done to force the use of boundary values in the test cases. If a test case specification selects the class "range_start", the value that shall be checked if it is inside the range or not shall take the value of the start of the range, i.e. the lowest value that is considered to be inside the range, i.e. a boundary value. The class "opposite_border" is intended to create an analogous test case specification, but using the highest value that is considered to be inside the range. The class "range_start" and the class "opposite_border" both contain only a single value. All other values inside the range are collected in the class "inlying"; this class exists mainly because of the requirement for completeness of equivalence partitioning. A similar approach to use boundary values is visible in the classes "at border" for positions outside the range.

4.10 The Completed Test Case Specification

In Fig. 15, the same classification tree as in the figure above is depicted, but with a completed combination table, what results in a complete test case specification for the functional problem at hand.

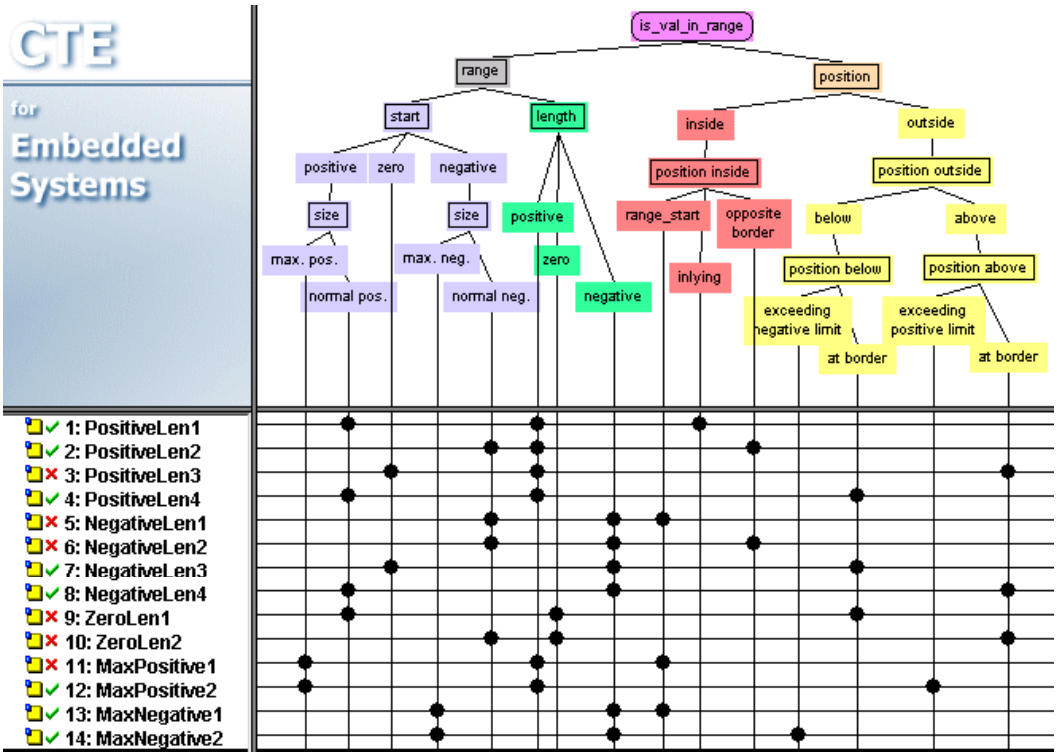

Fig. 15 The completed test case specification

the test case specification above lists 14 test cases. Please note that these are specified by the human user, i.e. they depend on its appraisal. Thus the scope of the test remains 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. 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). The maximum criterion takes into account that it is not possible to select e.g. a negative length and a positive length for the same test case specification, because this is impossible by the construction of the tree. However, the maximum criterion takes not into account that it is not possible to select e.g. a zero length and "inlying", because this is not impossible by the construction of the tree, but by the semantics of the function problem at hand.

A reasonable number of test case specifications obviously lies somewhere between the minimum and the 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. In the test case specification at hand, the classification tree has 15 leaf classes, what fits well to 14 test cases.

By the test case specification in the figure above, you can deduct how the functional problem specification was extended with respect to the questions raised in sections A Second Range Specification (p. 14) and Another Interesting Test Case Specification (p. 16).

- Question from section 4.6: If the length of the range is negative, are there values that can be inside the range? The answer is obviously "yes", because in test case specification no. 5 and no. 6 a negative length shall be used and the position of the value shall be inside the range.
- Question from section 4.8: If the length of the range exceeds the given integer range, shall negative values be inside the range? Test case specification no. 12 clarifies, that this should not be the case.

The leaf class "inlying" is selected for only one test case specification, namely test case specification no. 1. This reflects the fact that this class exists only because of the requirement for mathematical completeness of equivalence partitioning, and not because the inlying values are considered to produce error-sensitive test cases.

4.11 Another Test Case Specification

In Fig. 16, an alternative test case specification to the functional problem specification at hand is depicted.

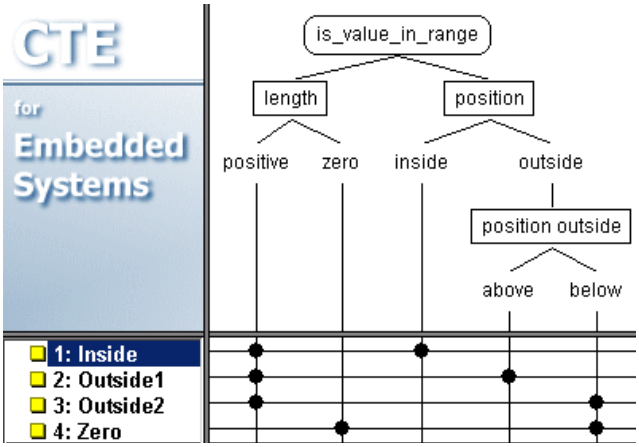

Fig. 16 An alternative test case specification

What are the differences to the more elaborated test case specification in the section above?

- The start value of the range is not mentioned in the classification tree. This means, the start value is not considered to be a test-relevant aspect by the user of the CTM. In consequence, any arbitrary value can be used as start value in the four test cases. This value can be fix for all test cases, but does not have to be.
- The problem of a negative length is completely neglected. For the problem specification from section Problem on p. 12 which specifies a length to be an integer and hence also the length to be negative, this is a serious flaw.
- The problem of wrap-around is neglected. This may be considered to be an esoteric problem, and therefore it could be accepted that it is not mentioned in the alternative test case specification.
- The usage of boundary values is not forced by the alternative test case specification. This is questionable, because boundary values produce error-sensitive test cases. The alternative test case specification minimizes testing effort (by specifying only four test cases), but this is at the cost of thoroughly testing.

But the point is not which test case specification is better. The main point is:

Test case specification according to the Classification Tree Method visualizes testing ideas!

This is illustrated by the discussion of the two solutions above.

5 Tool Support

Applying the CTM is supported by the Classification Tree Editor CTE. The test tool Tessy can import test case specifications from CTE.

The description of CTE refers to CTE/ES V2.4.

Both CTE and Tessy originate from the former software research laboratory of Daimler in Berlin.

5.1 Classification Tree Editor CTE

The Classification Tree Editor CTE supports drawing classification trees and specifying test cases in the combination table.

5.1.1 The Drawing Area

The CTE features a graphics editor especially made for drawing classification trees. For instance, the editor helps to create the required sequence of classifications and classes, i.e. it suggests only the allowed node types as sub-nodes for a given node.

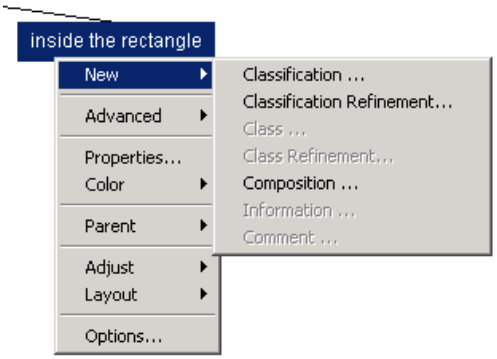

Fig. 17 Creating new nodes is context-sensitive

In the figure above, a sub-node for a class shall be created. The CTE does not allow creating e.g. a class node as sub-node for a class and thus avoids illegal sequences of nodes.

To manage bigger trees, sub-trees may be used. Sub-trees are called "refinements" of classes or classifications. See section Refinements (p. 28). Sub-trees may be stored separately and may be used as building blocks for other trees, similar to a library.

For better overview in bigger trees, the CTE features a navigator window. It allows easy and comfortable navigation to any location of the tree.

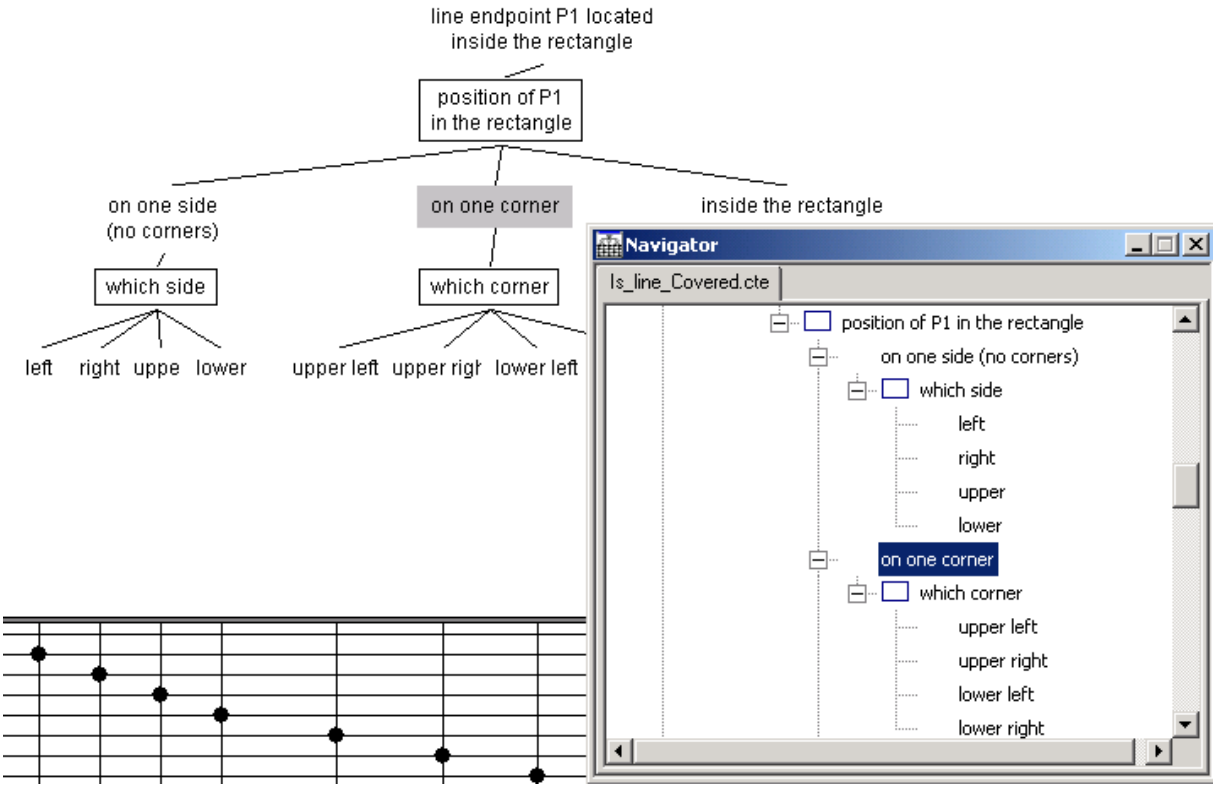

Fig. 18 The navigator window allows moving easily to any tree location

Descriptions and comments may be inserted in the tree or attached to tree nodes. This serves for documentation purposes.

The layout of a tree can be done automatically according to different styles. This provides for good overview and for efficient use of the drawing area.

Elements of the tree (i.e. single nodes or sub-trees) can be cut or copied and pasted to other locations in the tree.

5.1.2 The Combination Table

The CTE also helps managing test case specifications in the combination table.

Test cases specifications can be defined as sequences. This can be used to describe dynamic tests, See section Sequences of Test Case Specifications on p. 27.

Comments/descriptions can be attached to test case specifications. This may be used to link a test case to the requirement it tests. Also the expected result of a test can be specified. After a test was executed, the actual result (passed/failed) and the actual reaction can be documented. When the CTE is used in conjunction with Tessy (see below), the passed/failed result is transferred automatically from Tessy to the CTE.

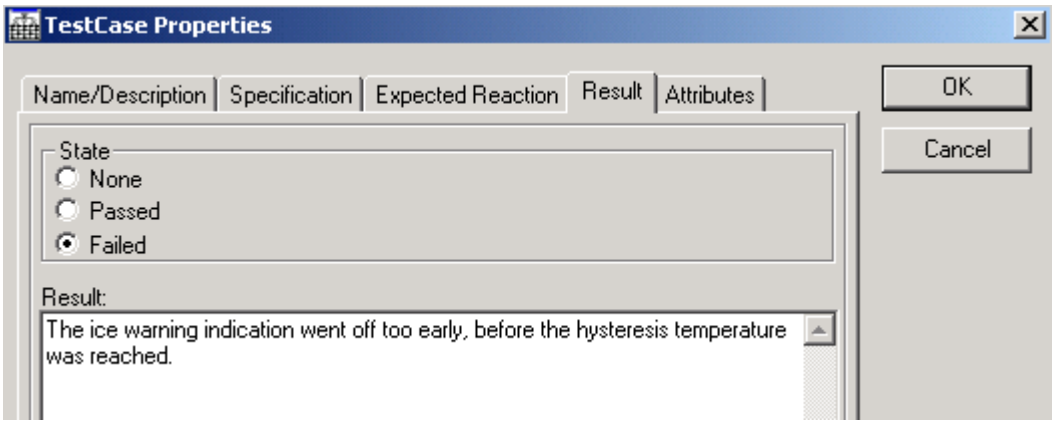

Fig. 19 Properties of a test case specification in the combination table

5.1.3 General

The CTE can check a classification tree and its combination tables. This reveals unused classes, empty test case specifications, incomplete trees, and the like.

The CTE can calculate some metrics on the trees, e.g. the minimum and maximum criteria (see end of chapter 3.3 on p. 11 and chapter 4.10 on p. 19). This allows an estimate of the effort required for testing.

Test case specifications in the CTE can be exported/saved in various formats (*.xml, *.txt, *.html, *.wmf), and also to Word, Excel, or directly to a printer. This can be used to import the test case specifications into other tools, e.g. test management tools or test execution tools.

To Tessy (see below), test cases specifications can be exported directly.

5.2 Tessy

Tessy [1] is a tool to automate module/unit testing of embedded software. Tessy manages test cases, executes the test, evaluates test results and creates test reports. Also code coverage is determined by Tessy automatically.

The CTE can export test case specifications to Tessy; Tessy can export test results (passed/failed) back to CTE.

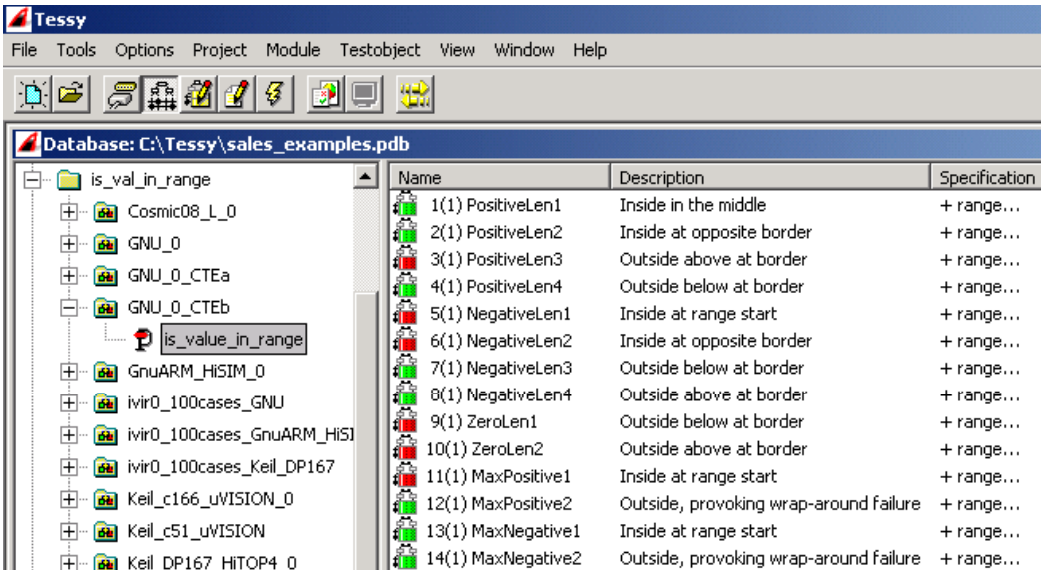

Fig. 20 Test case specifications from the CTE were executed in Tessy

In Fig. 20, test case specifications for the example "is_value_in_range" were exported form the CTE to Tessy and then executed by Tessy. The green/red icons specify passed/failed test results. These test results are exported from Tessy back to CTE.

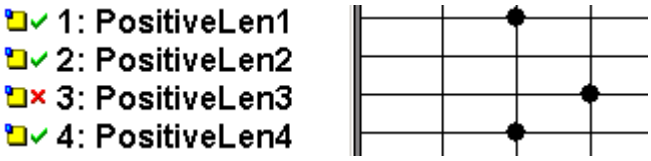

Fig. 21 Green tick marks and red crosses indicate the passed/failed result of a test

In Fig. 21, the passed/failed results of a test were transferred from Tessy to CTE automatically, but this can also be done manually in the test case properties of the CTE.

6 More on the Classification Tree Method

6.1 Composition

Usually classifications and classes alternate in the classification tree. The relation between a classification and its classes is "can take the value of". It is important to distinguish this relation from "consists of"

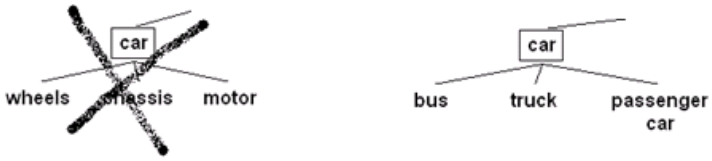

Fig. 22 Classification "car"

On the left hand side in the figure above, the relation between classification (car) and classes (wheels, chassis, motor) is "consists of", i.e. a car consists of wheels, motor, and chassis. This is not the intended semantic between classification and class in a classification tree. A correct semantic is depicted on the right hand side of the figure above: A car can take the value of a bus, or a truck, or a passenger car.

If you need to express "consists of", you may use a "composition" in a classification tree. A composition is depicted by a rectangular node like classifications, but with thicker borders than classifications. Compositions are "transparent" for the classification / class alteration. Values from classes related to classifications combined by a composition may be combined in a test case specification, i.e they do not exclude each other.

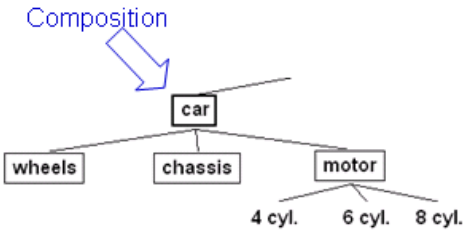

Fig. 23 Composition "car"

In the classification tree for "is_value_in_range" (refer to Fig. 14 on p. 17) also uses a composition.

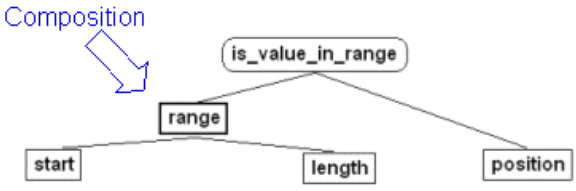

Fig. 24 Composition "range"

6.2 Sequences of Test Case Specifications

The CTM also can be used to specify test cases that consist of sequences of test case.

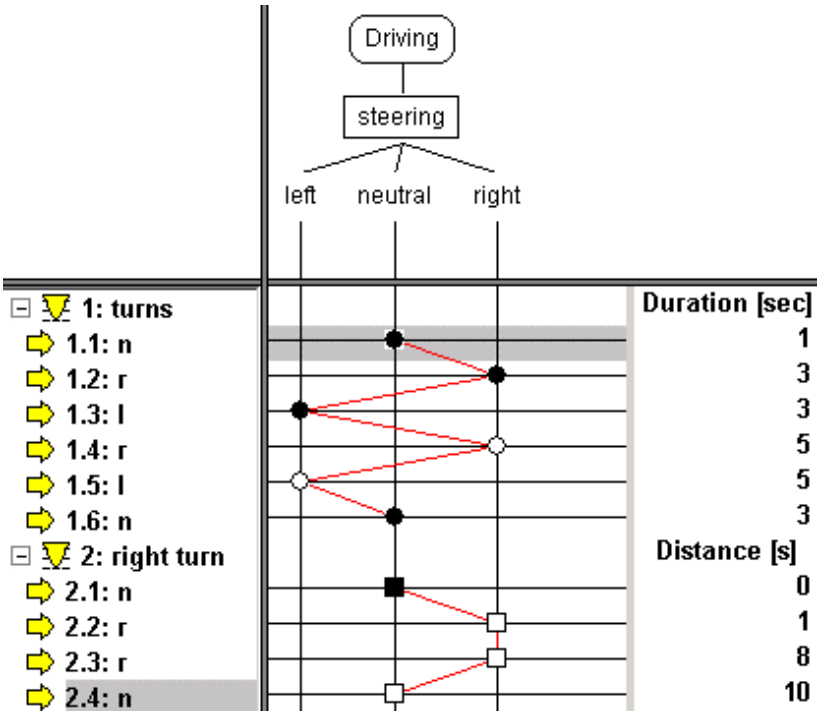

Fig. 25 A sequence of steering angles

In the figure above, two sequences of steering angles are specified. The part of such a sequence is called "test step". I.e. sequence no. 2 consists of test steps 2.1, 2.2, 2.3, 2.4.

On the right hand side, the time specification for the sequence is given. There are two types of time specifications available: Duration, which specifies the time a certain sub-sequence shall take; and distance, which specifies the starting point of a test step on an absolute time line.

The first sequence starts in the neutral position, which shall be held for 1 second. Then the steering wheel shall be turned to the right; in this position is shall be held for 3 seconds; and so on.

The second sequence also starts in the neutral position, which shall be held also for 1 second. Then the steering wheel also shall be turned to the right; but in this position is shall be held for 7 seconds; and so on.

Several graphical symbols may be used to specify the test cases. The markers could be circles, triangles, squares, and they can be solid or not. The transitions between the sub-sequences can have different shapes: solid lines, dashed lines, etc. This is for better overview only; it has no predefined built-in meaning whatsoever.

6.3 Coping with Big Trees

Sometimes trees grow too big. There are several approaches to cope with this.

6.3.1 Refinements

Bigger trees can be allocated to several drawing pads. For this, the tree is spilt into sub-trees, which form a hierarchy of trees. The transition points between the different drawing pads are called refinements. Refinements may be used both for classes and for classifications. Refinements for classes are depicted by an underlined class name. Refinements for classifications are depicted by a double line at the side of the classification rectangle.

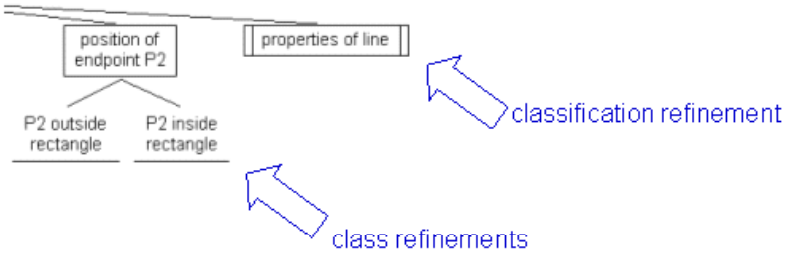

Fig. 26 Refinement for classes and classifications

6.3.2 Simplifying Specifications

Sometimes it is possible to reduce the number of classes in a tree by abandoning explicit test case specification, and use descriptive texts to specify your testing intentions. This results in smaller trees and hence improves overview. It may lead to less test case specifications required for that tree; but this is not the case in any situation.

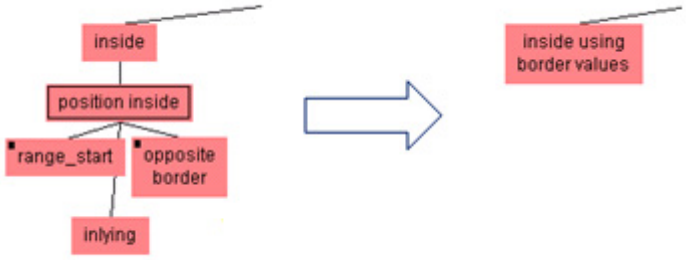

Fig. 27 Abandon explicit test case specification

In the figure above, a part of the classification tree from Fig. 14 on p. 17 was taken and simplified. It is now no longer guaranteed (by the syntax of the tree) that a test case with the value "range_start" will be present in the set of test cases. This objective is transferred to the process of specifying parameter values (i.e. data) for test case specifications. There parameter values shall be varied in a way that boundary values are used. This abandons the idea behind equivalence partitioning.

Please note: If the tree from Fig. 14 on p. 17 is simplified as described in the figure above, the number of required test case specifications (i.e. the minimum criterion) actually decreases. This is because the sub-tree deleted is relevant for the minimum criterion. I.e. the tree according to Fig. 14, the minimum criterion is 7. If the tree is simplified according to the figure above, the minimum criterion is decreased to 5. However, not all simplifications of the tree decrease the minimum criterion. E.g. if you use only two classes for the length of the range instead of three classes as depicted in Fig. 14, the minimum criterion will not be decreased.

6.4 Separating Specification from Data

6.4.1 The Difference between Abstract and Parameterized

When using the CTM, one should use abstract test case specifications. Care shall be taken that a test case is not parameterized too early. Ideally, test case specification is done without knowing about the implementation of the test object. This may prevent to parameterize too early, because the necessary information may not be available during test case specification.

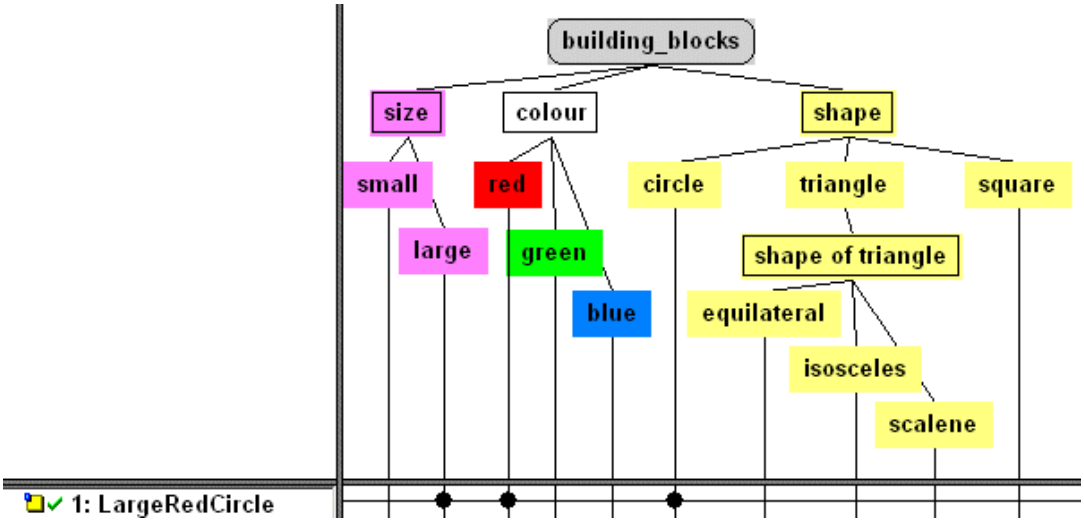

Fig. 28 An abstract test case specification

In the figure above, a "red" building block shall be the test object. However, without knowledge of the realisation/implementation, it is practically impossible to provide a parameter value (i.e. data) for "red". (Is the colour coded as an enumeration type? And if yes, what is the name of the component related to "red"? Or is the colour coded as RGB value? And if yes, how many bits make up this RGB value? And how is "red" coded?)

By abstract test case specification, test case specification is kept independent of the implementation. Therefore, abstract test case specifications may be (re-) used for different implementations. Also abstract test case specification keeps the semantic of a test case specification better than a parameterized one. For instance, a class "prime number" could be parameterized by using the value "13", what actually is a prime number. But having only the plain value "13" of the parameterized test case, it is no longer possible to deduct the original intention. The value 13 could also be a representative of the class "two digit number" or of the class "value greater than 10". The knowledge that it is acceptable to replace 13 by 17, but not by 15, will be lost.

6.4.2 Parameterization

When used with Tessy, the CTE allows specifying test data for a test case specification, i.e. parameterization of a test case. It is very tempting to do so, because it saves a lot of effort. However, as described in the section above, abstraction is lost.

Parameterization is done in two steps:

(1) A variable of the interface of the test object is assigned to a classification of the classification tree. The set of interface variables is determined by Tessy and exported to the CTE.

(2) A value is assigned to a class emerging from this classification.

Every test case specification, for which a marker in the combination table selects a class with a value causes that value to be exported to Tessy when the respective test case specification is exported to Tessy.

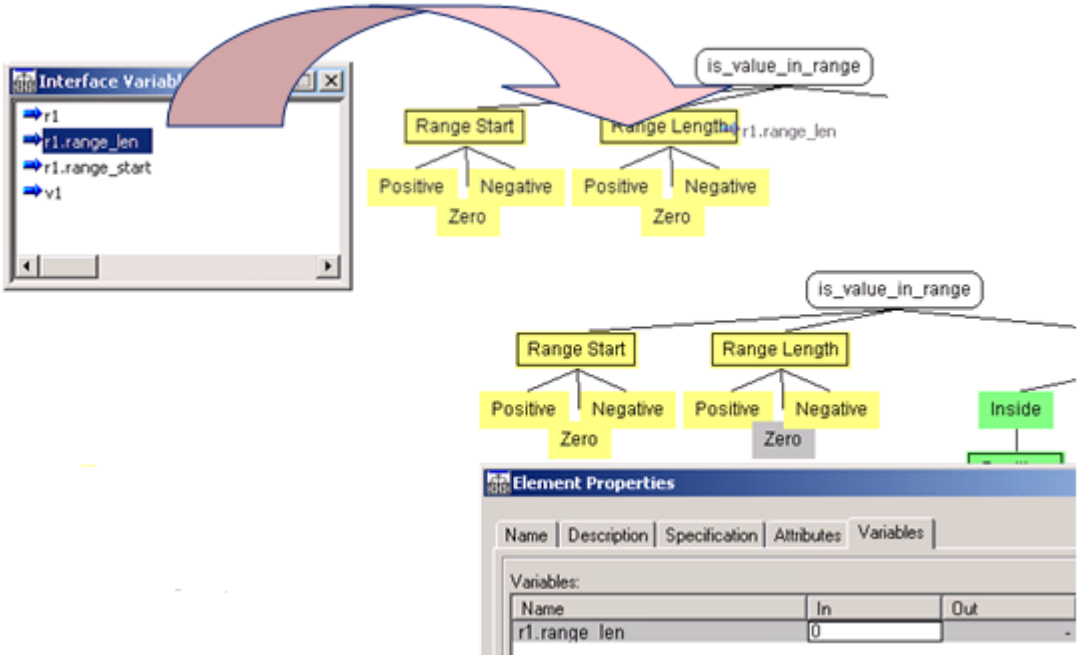

Fig. 29 Assigning a value to a class

In the upper part of the figure above, the interface variable "r1.range_len" is assigned to the classification "Range Length". In the lower part of the figure above, a value is assigned to the class "zero", which emerges from the classification "Range Length". The value assigned to the class "zero" is 0. For every test case specification, in which the class "zero" is marked, the value 0 is used to parameterize this test case. The value in question is automatically transferred to Tessy when the test cases specifications are exported to Tessy.

Since 0 for "zero" is natural and obvious, it is not to criticize to parameterize in that way. But the same way a value could also be assigned to e.g. the class "positive" of the same classification. However, which value shall now be selected? Shall it be 5, or 10, or 20? Or 100, 1000, 10000? This is not easy to answer and thus gives insight to the problem of too early parameterization.

There is a trade-off between saving a lot of effort by parameterization and loosing to much abstraction (i.e. re-usability).

Be aware of it!

7 Example "Ice Warning Indication" --- Continued

7.1 Where We Left Off

In section Classification Tree for the Example "Ice Warning" on p. 10 we have designed an initial classification tree for the example "ice warning". If we add the combination table and set some markers, we get an initial test case specification.

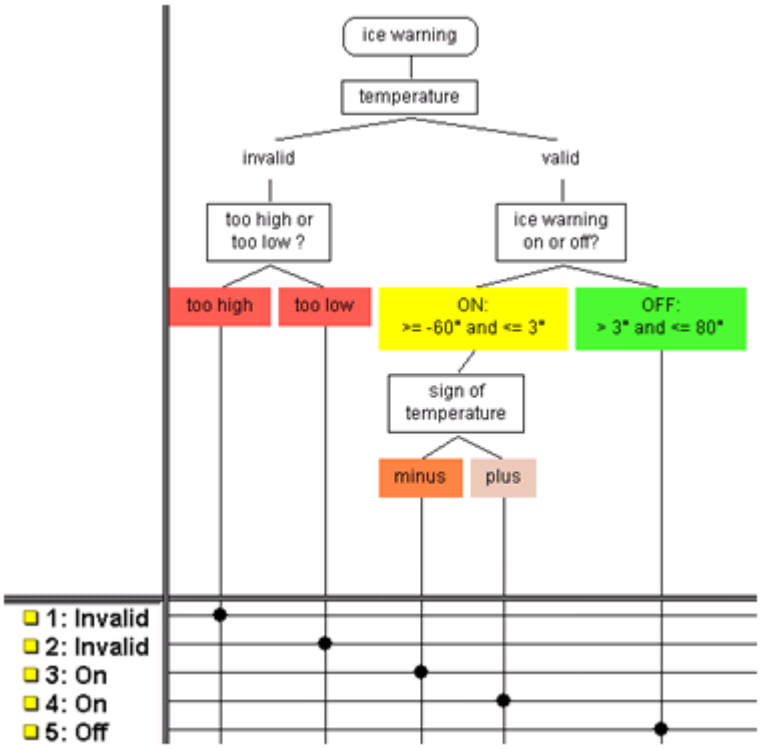

Fig. 30 Initial test case specification for "ice warning"

The combination table in the figure above comprises of five test cases, which are formed very simple: Each leaf class of the classification tree is assigned to a test case. Each test case uses a value from the respective class. The names of the test cases indicate the expected result. (The expected result could also be defined directly as a property of a test case, together with other properties like description, actual result, and the like.)

7.2 Hysteresis Using a State

Now fast changes in the state of the ice warning display shall be avoided. This could be realized by a hysteresis function. The ice warning display shall be switched off only after the temperature has risen to more than 4 °C. (Still the ice warning display shall be switched on if the temperature falls to 3 °C or below). Therefore, the result of a test with a temperature between 3 °C and 4 °C depends on the current state (on/off) of the ice warning display. In the classification tree this can be reflected by introducing an additional classification, which represents the current state of the ice warning display. The classes of this classification are combined with values of the temperature range between 3 °C and 4 °C. This yields two new test cases (see figure below).

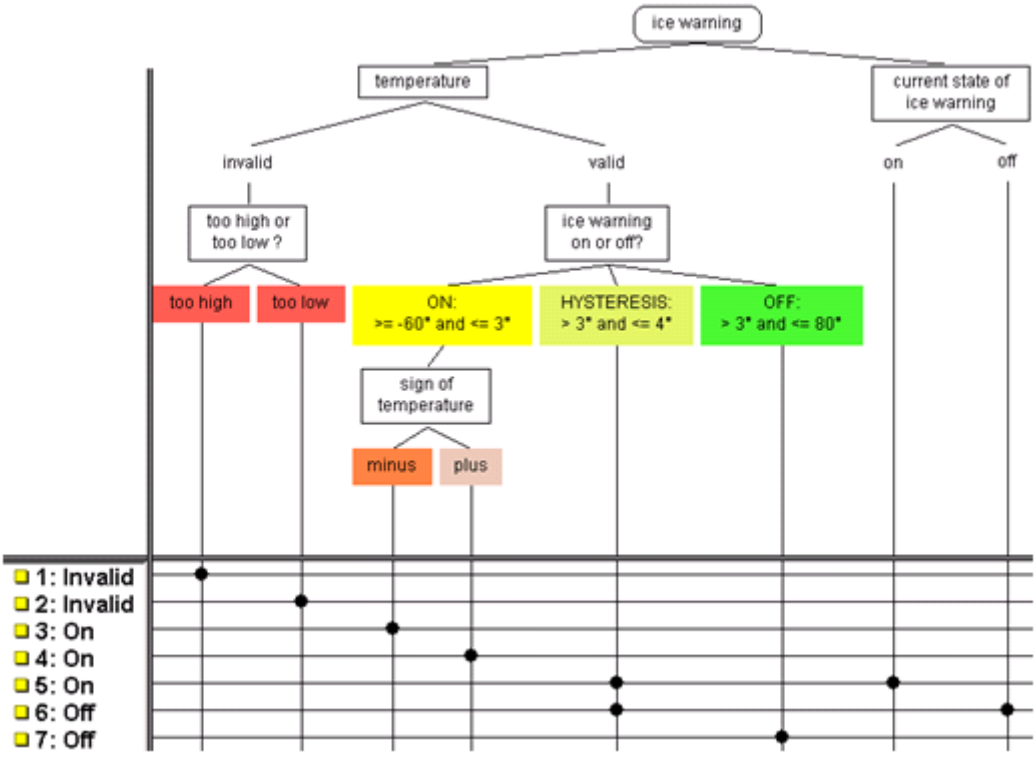

Fig. 31 Hysteresis using a state

7.3 Hysteresis Using a Sequence

For a better test of the hysteresis functionality than using a state as in the figure above, a test case could be a (time-discrete) sequence of temperature values. To specify the current state of the ice warning display for the tests would be superfluous, because this state would be maintained by the ice warning display internally.

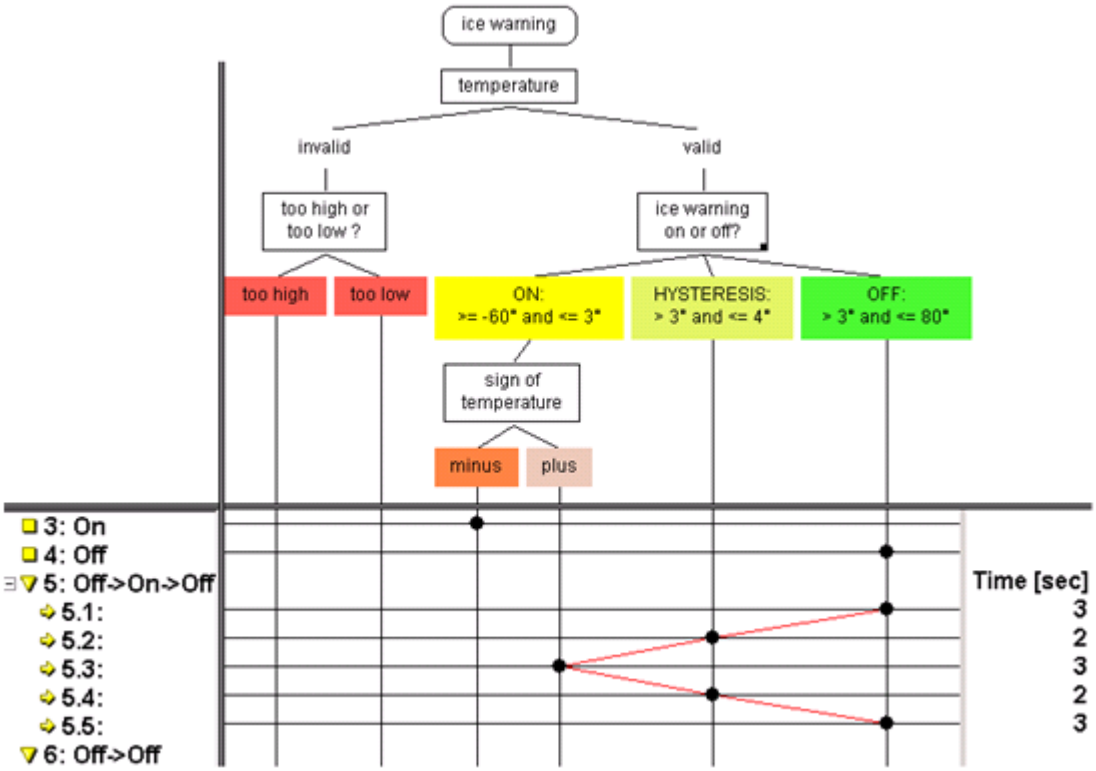

Fig. 32 Hysteresis using a sequence

The combination table in the figure above describes such a test sequence (see also section Sequences of Test Case Specifications on p. 27). Test case no. 5 consists of a sequence of five test steps. They test the following sequence of the ice warning indication: Off → On → Off. Obviously, more sequences need to be tested, e.g. On → Off → On.

The sequences can be displayed expanded or collapsed, what serves for a good overview. The test steps of a sequence are connected by lines. Each test step has properties, e.g. to note the expected result of the respective step. It is even possible to specify and to visualize the timely behavior of a sequence, as shown in the figure above. However, the current example does not require specifying a timely behavior, because a time-discrete, equidistant sampling of the temperature signal is assumed.

7.4 Hysteresis Using a Sequence and Boundary Values

But the test sequence in the figure above does not specify the exact values of the temperature. To test especially values near the borders of the hysteresis, we assume a resolution of 0.1 °C for the temperature. Now we can extend the classification tree by introducing appropriate classifications for the values of the temperature. This allows us specifying boundary values.

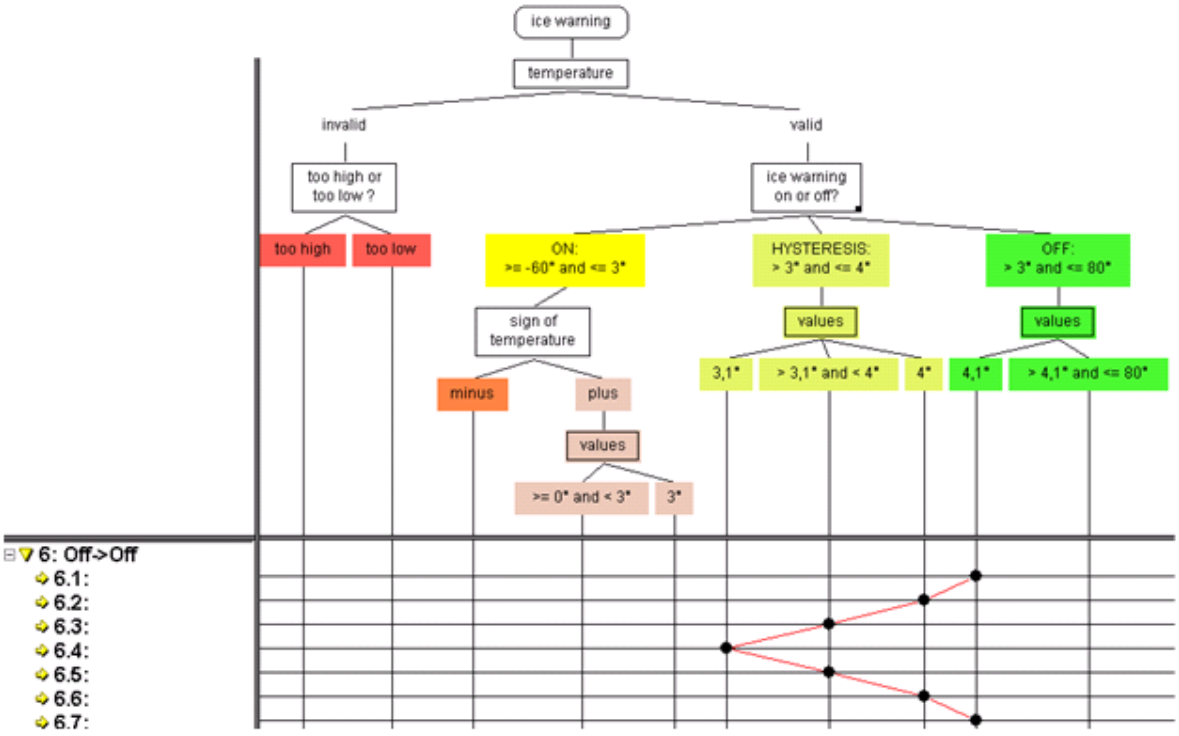

Fig. 33 Hysteresis using a sequence and boundary values

Test case specification no. 6 in the figure above now specifies rather well the characteristics of the temperature signal, which is used as test input to the ice warning display: The test starts with the value of 4.1 °C. Then it decreases and enters the hysteresis range. The lowest value reached during the test is 3.1 °C. After that, the temperature raises again. During the complete test the ice warning display is expected to be off.

The figure above reveals at a glance that it was not in the intention of the designer of the tree to use 2.9 °C and 3.9 °C as boundary values. This is another example on how well the CTM visualizes testing ideas.

8 Advantages of the Classification Tree Method

8.1 Visualizes Testing Ideas

A primary advantage of the CTM is the visualization of testing ideas. Due to this visualization, the ideas are easy to communicate and to understand, e.g. during reviews or during a certification process.

8.2 Gives Confidence

A well-designed classification tree and carefully compiled test case specifications provide a high probability, that no relevant tests are overlooked. This in turn should effect that most errors (or maybe even all errors) will be revealed. However, it should be kept in mind that the CTM is applied by a human being, and therefore is depending on the appraisal of this human. Because of human participation, there is no complete assurance that errors are omitted generally. In case of doubt, a review of the test case specifications is recommended. Because of the visualization, a review should not need oversized effort.

Having a set of test cases that you can rely on is prerequisite for the refactoring of a test object, because the test cases to re-test the test object (regression test) are available.

8.3 Reduces Complexity

When you first think about the test cases for a given problem, you usually have a lot of testing ideas immediately. The challenge is to structure these ideas and to sort out redundant test cases. The CTM reduces the effort required for that task, because it enables to consider aspects (classifications) separated from each other and to think in depth about required (and not required) classes according to the equivalence partitioning. However, a complex problem probably will need complex test case specifications.

8.4 Checks Problem Specification

When applying the CTM the analysis of the (functional) problem specification is basis. This checks automatically the problem specification for contradictions, omissions, inconsistencies and the like. In the example "is_value_in_range" two weak points were detected in the problem specification:

(1) How to cope with a negative length of the range, and

(2) how to proceed if the range exceeds the highest positive number.

8.5 Estimates Testing Effort

Just using the classification tree it is possible to calculate some metrics, e.g. the minimal number of test cases required for the tree at hand (i.e. the minimum criterion). This and other metrics can be determined using CTE.

After all test cases are specified, you have a good estimate on the effort required. Assuming that no test cases are overlooked, you have a preliminary test end criterion at hand.

9 Literature

[1] http://www.hitex.de/perm/tessy.htm: More about Tessy and CTE.

[2] Grochtmann, M., Grimm, K.: Classification Trees For Partition testing, Software testing, Verification & Reliability, Volume 3, Number 2, June 1993, Wiley, pp. 63 – 82.

[3] Wegener, J., Pitschinetz, R.: Tessy – Another Overall Unit Testing Tool, Quality Week 1995.

[4] Grimm, Klaus: Systematisches Testen von Software: Eine neue Methode und eine effektive Teststrategie. München, Wien, Oldenburg, 1995. GMD-Berichte Nr. 251.

[5] Mirko Conrad: A Systematic Approach Of Testing Automotive Control Software, Society of Automotive Engineers, 2004

[6] Broekman, B., Notenboom, E.: Testing Embedded Software. Addison-Wesly, 2003.

The Classification Tree Method is guidance for thinking, not replacement of thinking !