Test Case Design by Means of the CTE XL

Eckard Lehmann and Joachim Wegener
DaimlerChrysler AG
Research and Technology
Alt-Moabit 96 a
D-10559 Berlin
Eckard.Lehmann@daimlerchrysler.com
Joachim.Wegener@daimlerchrysler.com

Abstract

Although black-box tests are very important for the quality assurance of software systems, only a few methods for systematic black-box test case design have found their way into the industrial world. The Classification-Tree Method is a particularly efficient testing method for which tool support is available in form of the Classification-Tree Editor CTE.

This paper introduces a new version of the CTE, namely the CTE XL (eXtended Logics). The CTE XL has been fitted with a number of methodological enhancements. These extensions are the result of the intensive use of the CTE tool in industrial practice within the last couple of years.

1 Introduction

The selection of test cases determines kind and scope of the test. Therefore, the test case design determines the quality of a test. Basically, testing methods for systematic software testing may be subdivided into white-box tests and black-box tests. Established white-box testing methods are, for example, statement testing, branch testing, condition testing, or path testing. Since test case design in white-box testing is merely based on the program under test, it has the weakness that the total implementation of all specified requirements in the program cannot be checked. Consequently, in industrial practice black-box tests are more important. In black-box testing test case design is based on the functional specification of the test object or on other information on the test object's function. Although black-box testing is of significant importance for the quality assurance of software systems, only a small number of tools specialized in systematic test case design have entered the industrial world. So, in many cases test case design is still carried out with insufficient tools, as for example decision tables [Schaefer, 1999] or Excel [Kit, 1999].

One efficient method for black-box testing is the Classification-Tree Method (CTM), introduced by Grochtmann and Grimm [1993]. It supports systematic test case design and gives a compact and clear presentation of the over-all test. The Classification-Tree Editor (CTE) is an efficient tool to support the CTM [Grochtmann et al., 1993]. Over the past years CTM and CTE have been successfully applied in many industrial software development projects in fields such as aviation and space technology, rail electronics, defense electronics, car electronics, engine electronics, and automation technology as well as commercial data processing applications (e.g. [Grochtmann, 1994] and [Grochtmann and Wegener, 1995]).

Through the intensive industrial application of the CTE, several improvement possibilities have been determined resulting in extensions to the CTM as well as a new design of the CTE. The CTE XL (Classification-Tree Editor eXtended Logics) described in this paper is characterized by important new features like

- the specification of logical dependencies between elements of the classification tree, which make it possible to exclude inconsistent class combinations from test case design;
- different priority levels for classifications and classes of the classification tree in order to distinguish the importance of aspects or properties for the test case design; and
- automated generation of test cases based on the classification tree.

Additional features support a better handling of extensive tests and the automation of subsequent test activities like test data selection, expected values prediction, and test execution.

The structure of the paper is as follows: After a short introduction to the CTM in Section 2, Section 3 explains the identified improvement potentials. Section 4 describes and exemplifies the implemented solutions. Finally, Section 5 summarizes the results and gives an outlook on future work.

2 Introduction to CTM and CTE

The CTM [Grochtmann and Grimm, 1993] supports black-box test case design in regard to systematic and complete segmentation of the test object's input domain into a finite number of mutually disjoint equivalence classes. A prominent feature of the CTM is its clear proceeding. The basic idea behind the CTM is first to abstract from concrete test data and to divide the test object's input domain into separate subsets (classes) according to aspects the tester considers relevant to the test. Then, test cases are generated by combining classes from different classifications.

The first step when using the CTM is to identify test-relevant aspects. For this, the most important basis of information is the functional specification of the test object. Additionally, in order to find interesting aspects creativity and expertise of the tester are indispensable. For each aspect the input domain will be completely divided into disjoint subsets. This classification should allow a precise and clear differentiation of possible inputs. The partitioning into classes is done separately for each aspect, therefore it is easily carried out.

In many cases it is useful to introduce sub-classifications which consider not the entire input domain, but just one class of an already existing classification, e.g. the classification *shape of triangle* in Figure 2 is only relevant to building blocks of the class *triangle*. This recursive use of classifications on classes can be continued over several levels until a precise differentiation of possible inputs is achieved. The result is a tree of classifications and classes: the classification tree. This tree is graphically presented (see Figure 2). The root of the classification tree represents the test object's entire input domain, which is successively subdivided into classes.

The second step is to build test cases on the basis of the classification tree. A test case is defined through the combination of classes from different classifications. For each test case exactly one class of each classification is considered. For this purpose the classification tree is used as head of a combination table wherein the classes which are to be combined are marked. Each line in the table represents a test case, each column represents a not further refined class of the classification tree. The number of test cases depends on the tester's combinatorial choice. Something to go by in the combinatorics are the minimality criterion and the maximality criterion [Grochtmann and Grimm, 1993]. The minimality criterion describes the number of test cases that is necessary to consider each class of the classification tree in at least one test case. The maximality criterion is met when all possible combinations are considered.

Due to the separation of the test case design process into several steps and the graphical representation of classification tree and combination table, the CTM is well-suited for tool support. The Classification-Tree Editor CTE [Grochtmann and Wegener, 1995] recognizes the syntactic rules of the CTM and can act as a stepwise instruction to determining test cases. The CTE is a syntax-directed, graphical editor for test case design with the CTM.

2.1 Example

In the following there is a simple example for a CTM application. The test object is a computer vision system which should determine the size of different objects passing the camera of the system on a conveyor belt (see Figure 1). Possible inputs are various building blocks.

Appropriate aspects for the test in this particular case would be, for example, the *size*, *color* and *shape* of a building block. The category *size* will be divided into two classes only: *small* and *large* building blocks. The classification based on the aspect *color* leads to a partition of the input domain into *red*, *green* and *blue* building blocks, and the classification based on *shape* produces a partition into *circular*, *triangular* and *square* building blocks. An additional aspect is introduced for the *triangle* class: the *shape of triangle*. The various classifications and classes are noted as classification tree (Figure 2).

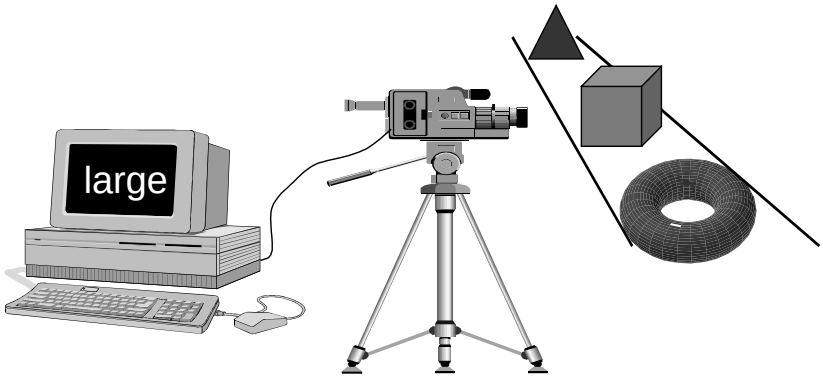

Figure 1: Computer vision system for determining the size of building blocks

In the combination table belonging to the tree some possible test cases are marked as examples. Test case three, for instance, describes the test with a small blue isosceles triangle. It requires five test cases in order to consider all classes of the classification tree in at least one test case (minimality criterion). The complete combination of all classes (maximality criterion) in this example would add up to 30 (2 * 3 * 5) test cases.

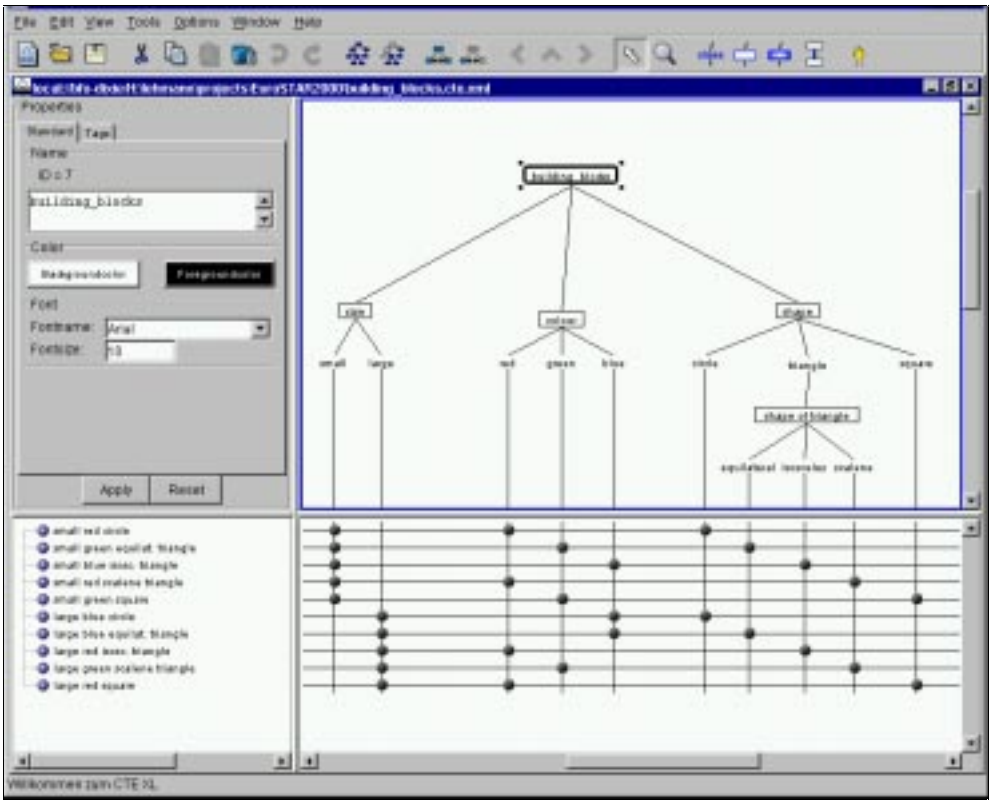

Figure 2: CTE window showing the classification tree for the computer vision system

3 Potential Improvements of CTM and CTE

From the intensive industrial application of the CTM and the CTE various ideas for improvements were generated which mainly focus on the handling and documentation of extensive tests and an increased automation of the test case design. The identified areas for improvements are listed in the following.

Logical dependencies. The current versions of the CTM and CTE are not equipped with a description feature to specify logical dependencies between classes. Therefore, when defining test

cases by combining classes of different classifications in the combination table, the tester needs to take care of the logical compatibility of the classes himself.

Supposing the input domain of the computer vision system example contains only green and blue circular building blocks but no red ones, then there is no way to describe this dependability in the classification tree in Figure 2. The tester has to keep in mind not to mark test cases specifying red circular building blocks. It should be noted, that it is possible to avoid this problem by restructuring the classification tree. In the example above, the classification *color* must be duplicated by arranging the copies under the different shapes *circle, green* and *blue*. Then the class *red* under class *circle* can be removed, which makes it impossible to select red circles. However, the result is a much more complex classification tree (six classifications instead of four). Especially in the case of extensive classification trees these kinds of transformation diminish the clarity.

Combinatorics. In many applications the CTM has shown that the chosen classifications and classes vary in their importance for the test. Currently, the CTM and CTE have no means to express this. For the computer vision system it is easy to understand that the classifications *size* and *shape* are more relevant to the test than the classification *color*. The size of the building blocks directly correlate to the specified task of the computer vision system, and the mere fact that the computer vision system recognizes the size of a square does not necessarily mean that it will also correctly recognize circles and triangles. But regarding the color it seems reasonable to assume that it influences the computer vision system's recognition performance independent from size and shape of the building blocks. Accordingly, a reasonable test would include all possible combinations of the classifications *size* and *shape* whereas the different colors would only be marked in the test cases in order to result in as manifold as possible combinations with the classes of the two other classifications (compare Figure 2).

Automatic test case generation. Practically, it takes a large amount of manual work to determine test cases when dealing with complex test objects. The classification tree has to be designed manually and the combination table has to be filled in by hand as well. In extensive tests with a large number of test cases this leads to a rapid loss of clarity and structure of the combination table. The CTE only indicates redundant test cases and classes that have not been taken into account. Consequently, it is a complex matter to combine classes in order to form a large set of test cases. An automatic test case generation would very much forward test case design. It would guarantee a systematic verification of the test object with all test-relevant class combinations. Logical dependencies and combination rules should be taken into account during automatic test case generation.

Hierarchies for large trees. The structure of the combination table also suffers from a lack of means for structuring the test cases. All test cases are plainly lined up after each other. For a better structuring it should be possible to pool test cases in groups which then can be opened and closed at will. Similarly, this applies to the columns of the combination table and subsequently to the actual classification tree. It should be possible to open and close any element of the classification tree together with its child elements to enable an easy treatment of combinations between classifications which are not placed directly next to each other in the classification tree.

Open tool interfaces. For practical application it is also desirable to have a close integration of the CTE with other development tools. Examples are requirements management tools, configuration management tools, tools for specification and design, and especially tools for test execution (compare [Conrad et al., 1999], [Wegener and Fey, 1997], and [Wegener and Pitschinetz, 1995]). In order to implement corresponding solutions a uniform API should be provided which, on the one hand, supports the supply of information from the test case generation to other tools and on the other hand the data import from other tools into the CTE.

4 CTE XL Extensions

In order to get rid of the listed weaknesses and to generally improve user friendliness of the CTM and the CTE a new version of the CTE has been developed: the CTE XL (**C**lassification **T**ree **E**ditor e**X**tended **L**ogics). The improvements, in comparison to the former version of the tool, are described in this section.

4.1 Description of Logical Dependencies

As stated above, the marks in each row of the combination table indicate which classes of the corresponding classifications have been selected for this test case. Logical dependencies for a classification tree occur when certain combinations of classes are not allowed or absolutely necessary. If, in the computer vision example above, there are no red circles this formally means that a mark on *circle* in a test case excludes a mark on *red* in the same test case. If “marked” is a logical statement then the above property could be expressed with the logical formula *"(circle => NOT red)"* (see Figure 3).

The CTE XL expresses logical dependencies in formulas of propositional logic. The propositional logic is sufficient for modeling dependencies in practice so that the use of higher order calculi (e.g. first order logic) is not necessary. All classes of a tree can be used as variables. A class that is not a leaf class can be considered as marked when at least one leaf class below is marked. Accordingly, the formula *"(red => triangle)"*, for example, describes that each red object is a triangle in any case. It does not matter which shape the triangle has.

With the CTE XL any logical rules can be related to a classification tree. For better reading and clarity each rule is labeled. For the specification of a logical rule the CTE XL provides a dependency editor into which the classes of the classification tree can be inserted using drag & drop. The logical operators can be selected via buttons. Existing logical dependencies can be automatically monitored in the combination table during manual definition of test cases. On the one hand, this prevents a selection of contradictory classes, on the other hand test cases are automatically completed when it is possible to uniquely derive further classes from the classes selected by the user and the logical rules. An automatic verification of all test cases is initiated in case the tester chooses to alter the classification tree or the specified logical rules. In Figure 3 the verification of the logical rule *"(circle => NOT red)"* would prevent the selection of test case "large red circle". If the verification procedure detects a violation of logical rules, the user receives a warning.

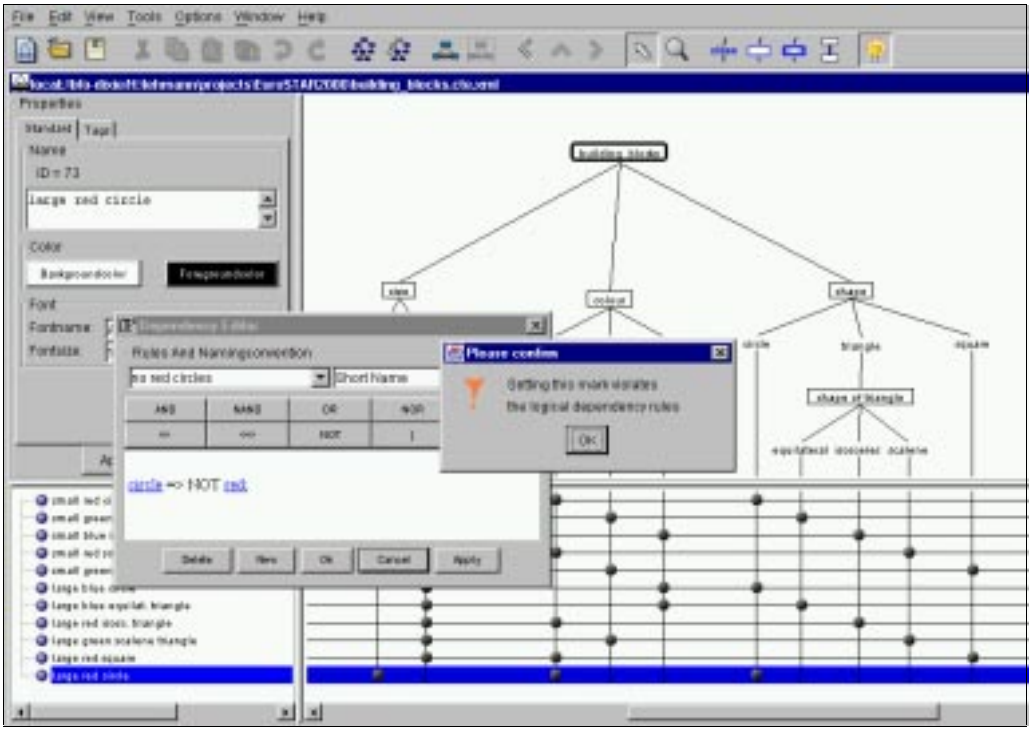

Figure 3: Logical rules in the CTE XL

The benefit of specifying logical dependencies has been proven in experiments and real-life applications. The tester can more easily define test cases and even a semi-automatic fill-in of the table is possible. In some cases the amount of theoretically possible test cases given by the classification tree was reduced by more than 90%. In the line example described by Grochtmann and Grimm [1993] a complete description of all logical dependencies leads to a reduction from 70.000 to 5.560 possible

test cases for the maximality criterion [Ebert, 1999]. Next to the maximum number of test cases, which is the sum of all possible class combinations, the CTL XL calculates further numbers taking under consideration the logic dependencies. One of these figures is the minimum number of test cases (minimality criterion) that are required to take into account each class of the classification tree in at least one test case. This number may increase through the description of the logical dependencies compared to the definition in [Grochtmann and Grimm, 1993]. Other figures relate the number of test cases defined by the tester in the combination table and the numbers given by minimality and maximality criterion. This enables the tester to estimate the degree of coverage of the test.

4.2 Combination Rules

Another important new feature of the CTE XL is the possibility to define combination rules for classifications or classes of the classification tree. With these rules we can express the importance of certain classifications or individual classes and of combinations of several classes and classifications for the test. The definition of combination rules enables the tester, for example, to specify that the classes of two particularly important classifications have to be completely combined in test cases and that the classes of other classifications are not fully combined but adequately added to the existing combinations. It is also possible to completely combine single classes with all classes of another classification: If in the computer vision example the background behind the conveyor belt is blue, all blue objects would be particularly critical because the camera might hardly recognize the color contrast. If all blue objects need to be tested, this can be expressed with the help of combination rules. The combination rule in Figure 4, for example, determines that all combinations of size and shape as well as all blue shapes have to be regarded. (Specific combinations in regard to the size of blue objects are not required in this example).

The CTE XL checks the selection of combination rules when the tester is filling in the combination table. In case the combination properties are not met by the marked test cases the user will be notified. The combination rules also express themselves in new measures which indicate how many test cases are necessary to meet all existing logical dependencies and combination rules specified by the user and how specified test cases and required test cases relate to each other.

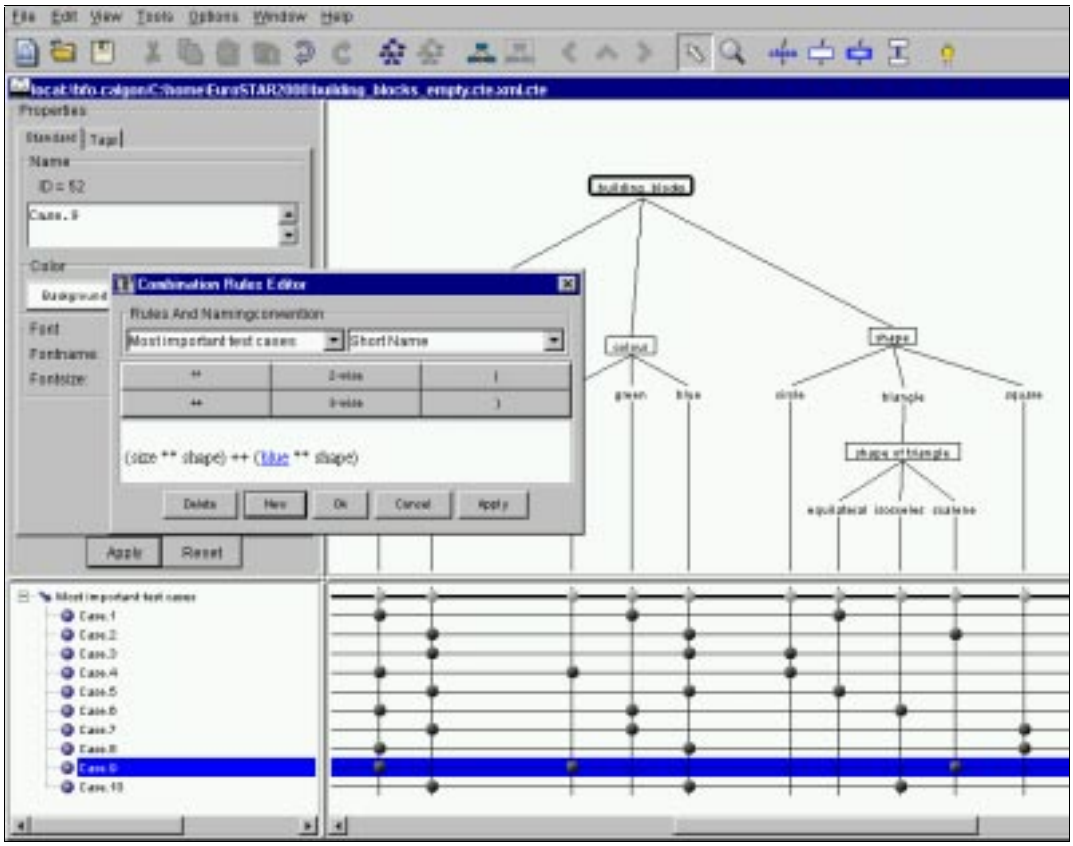

Figure 4: Combination rules in the CTE XL

In order to support a user-friendly handling of combination rules which nevertheless allow a high amount of flexibility, combination rules are, similar to the logical dependencies, represented as expressions which may be declared using a specific combination editor. All classes and classifications can be used as variables within the expressions. The following operators are available[1]:

- *Complete combination* A ** B: Each class in A is combined with each class in B.
- *Minimal combination* A ++ B: Each class in A and B are considered at least once.
- *n-wise-combinations* n-wise(A_1, A_2, ..., A_m): Each possible combination of *n* classes in A_1, A_2, ..., A_m ($1 \leq n \leq m$) is considered at least once (compare [Cohen et al., 1996]).

There may be several combination rules for one classification tree. In this way, different emphases of a test can be represented by different combination rules. Each combination rule has a label for better readability. Initial experiments have shown that the representation of combination rules as expressions with the above operators guarantee a high flexibility and readability.

4.3 Fully Automated Generation of Test Cases

An obvious objective of the description of logical dependencies between classes of the classification tree and of specifying combination rules for classes and classifications is to use this information for the automatic generation of test cases and subsequently for an automated test case specification. This is supported by the CTE XL. For filling in the combination table the user can choose from the following strategies:

- the generation of a *minimum* number of test cases which guarantees that each class is considered in at least one test case taking into account the logical dependencies (minimality criterion)
- the generation of the *maximum* number of test cases which is produced by taking into account the logical dependencies of the classification tree (maximality criterion)
- the generation of an *optimum* number of test cases, which fulfills the combination rules and the logic dependencies selected by the user (optimum combination)

For the third strategy test cases are derived with a method closely related to the combinatorial design method which is also applied with other tools (see [Cohen et al., 1996], [Kit, 1999]). Here the combination rules set by the tester are taken into account. The underlying algorithm executes a heuristic combinatorial search [Hall, 1986]. A deterministic algorithm is theoretically possible but not practicable because it requires exponential effort. The set generated by the search algorithm is locally optimized, i.e. in some cases the algorithm generates sets of test cases which include more test cases than theoretically necessary. But this is not a problem in practice. Only for a large number of test cases the generated result differs from the optimum. But in these cases the generated cases are better than the manually designed ones, due to the complexity of the combination structure. It is hard for a human to master that kind of complexity.

The automated fill-in of the combination table enables the user to concentrate fully on selecting test relevant aspects and the existing relations between those aspects. The concrete test cases are defined automatically.

4.4 Hierarchies for Large Trees

In many cases of practical testing the test objects are rather complex, so that the set of classifications and classes in focus is large. In these cases the resulting classification tree is, as a whole, difficult to comprehend. Therefore, the possibility to fold and unfold sub-trees has been added to the CTE XL. If a sub-tree is folded only the root node of this sub-tree is visible on the screen. An additional arrow indicates that there are further elements hidden under the root. The folded sub-tree can be displayed on another page (i.e. in another window).

In Figure 5 the sub-tree of the class *triangle* has been folded. From the highest level it can only be seen in which test cases a triangle is to be considered, but not which shape this triangle has. The arrow "🡻" indicates that a detailed description of *triangle* exists in a second window. In case the whole

[1] This is an extremely simplified description of the precise operators' semantics. An extensive description would go beyond the scope of this paper.

classification tree is to be displayed in one single window, the folded sub-trees can be unfolded at any time.

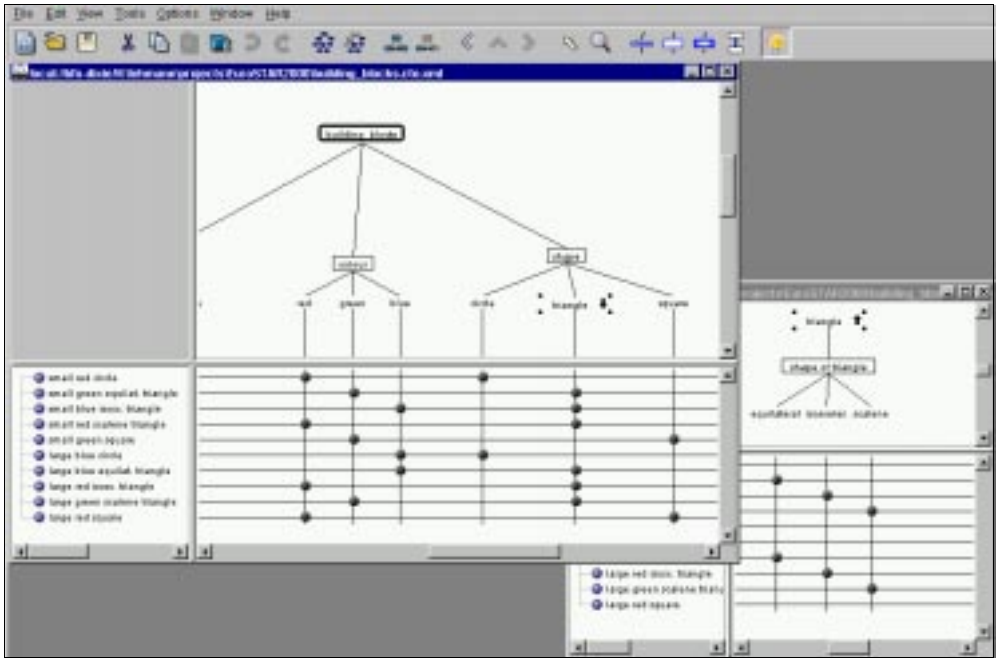

Figure 5: Folding sub-trees

Within the combination table mechanisms have been added as well in order to structure the test cases. This is especially useful with a large number of test cases, as test cases with different test emphases can be subsumed in groups (Figure 6). Any desired hierarchies of test cases are possible, i.e. groups of test cases may contain further groups. This rule is comparable to the concept of directories and files within the file-system. Test case groups can be unfolded and folded. Thus, it is possible to hide specific detailed information (content of groups) and to make this information visible again at any time.

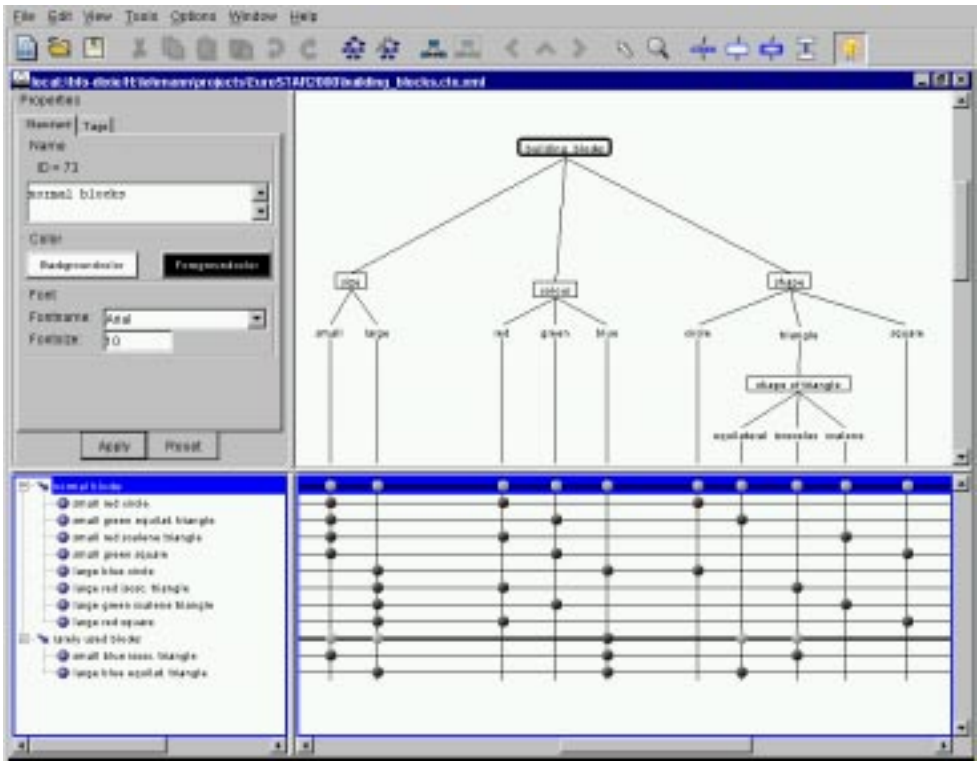

Figure 6: Hierarchies of test cases

4.5 New Types of Marks

Whereas the former version of the CTE only differentiates between two mark types for the selection of classes within one test case – that is to say *marked* and *not marked* – the CTE XL knows four mark types: *initial*, *marked*, *not marked* and *irrelevant*. At the beginning of a test case all leaf-classes of the classification tree are marked as *initial*. As soon as the tester selects a class, this class is *marked*. All contradictory classes are automatically marked as *not marked*. Thus, the state of work concerning a test case can immediately be visualized: Only when the test case contains no more marks of the type *initial* it is fully defined. The initial marks clearly indicate whether a test case has to be specified more precisely.

Another means of expression are the *irrelevant* marks. In case a classification with its respective classes is relevant to the over-all test but irrelevant to an actual test case because of its content, or because it has already been taken into account in detail in several other test cases, all classes of this classification can be marked as *irrelevant* to this test case. Semantically this means that any class of the classification can be arbitrarily selected for the test case.

4.6 Open Interfaces and Architecture of the CTE XL

Unlike its preceding version the CTE XL is not only conceived to be a test specification tool but also a platform for the integration with other test relevant tools. Therefore all data concerning classification trees and test cases are managed by one central server component which possesses a clearly defined API. The GUI front end of the CTE XL is based on this server. In addition, it is possible to relate further tools to the server in case they want to read or write test relevant data. Useful tools are, for example, tools for requirements management, configuration management, specification and test execution. (see for example [Conrad et al., 1999], [Wegener and Fey, 1997], and [Wegener and Pitschinetz, 1995]). Using so-called active tags the server offers a mechanism to manage any desired test-relevant data from other tools together with the classification tree. It is, for example, possible to manage test data and set values as well as references on requirements and version tags in the CTE.

For the simplified use with other tools a script interface helps, among other features, to export data to any desired format. It is also possible to use the script interface to integrate further tools which support the test case design: For example the integration of test assistants or the cooperation with library tools in which frequently used sub-trees are managed and then dragged into the CTE XL.

4.7 Additional Features

Furthermore, the handling of the CTE XL is more user friendly now. This is due to the dynamic enlarging, unlimited canvas for generating the tree, the management of any number of test cases, an auto-layout function for classification trees, the implementation of usual design features like color and font of classification-tree elements. And, now the trees can be scaled at will and property dialogues simplify the handling of tree attributes and test cases. These and other improvements have increased the applicability of the CTE.

5 Conclusion and Future Work

The new features implemented in the CTE XL result in a noticeable improvement for the test case design for black-box tests. In initial applications the efficiency and the effectiveness of the test could be improved. It is now possible to document all test-relevant information. Extensive classification trees and long lists of test cases can be easily handled. Test cases can be automatically generated. Information required for the test automation can be integrated via the tag concept.

Future work will concentrate on the improvement of individual technical tool processes as well as the development of further tools extending the CTE XL, like test assistants, library components, or tools for distributed work with classification trees. Additionally, there will be reinforced work in developing testing methods for special application fields where the CTM is only of limited applicability. The central

field of application of the CTM is testing the functional behavior of systems that compute discrete input data. For testing embedded control systems currently the Time Partition Testing is under development [Lehmann, 2000]. Furthermore, it is planned to automate the testing of model-based systems with the test and simulation tool MTest [Conrad et al., 1999]. Structural testing and temporal system behavior testing will be automated by means of evolutionary testing (e.g. [Wegener et al., 1999], [Wegener and Grochtmann, 1998], [Jones et al., 1998]).

6 References

ATS (2000): *Web-Page ATS*, www.ats-software.de/html/prod_cte.htm

Cohen, D., Dalal, S., Parelius, J., and Patton, G. (1996): *The Combinatorial Design Approach to Automatic Test Generation*. IEEE Software, September 1996, pp. 83-87.

Conrad, M., Dörr, H., Fey, I., Yap, A. (1999): *Model-based Generation and Structured Representation of Test Scenarios.* Proceedings of the Workshop on Software-Embedded Systems Testing, Gaithersburg, Maryland, USA, 1999.

Ebert, J. (1999): Spezifikation logischer Abhängigkeiten in Klassifikationsbäumen (Specification of Logical Dependencies for Classification Trees). Diploma Thesis, Technical University Berlin, 1999.

Grochtmann, M., and Grimm, K. (1993): *Classification Trees for Partition Testing*. Software Testing, Verification & Reliability, Volume 3, No 2, 1993, pp. 63-82.

Grochtmann, M., Grimm, K., and Wegener, J. (1993): *Tool-Supported Test Case Design for Black-Box Testing by Menas of the Classification-Tree Editor*. Proceedings of EuroSTAR'93, London, Great Britain, 1993.

Grochtmann, M., and Wegener, J. (1995): *Test Case Design Using Classification-Trees and the Classification-Tree Editor CTE*. Proceedings of the 8th International Software Quality Week, San Franzisco, USA, 1995.

Hall, M. (1986): *Combinatorial Theory*. John Wiley, interscience series, 1986.

Jones., B., Eyres, D., and Sthamer, H. (1998): *A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing*. Computer Journal, Volume 41, No. 2, 1998, pp. 98-107.

Kit, E. (1999): *The New Frontier ... Third Generation Software Testing Automation*. Proceedings of EuroSTAR'99, Barcelona, Spain, 1999.

Lehmann, E. (2000): *Time Partition Testing: A Method for Testing Dynamic Functional Behaviour*. Proceedings of TEST2000, London, Great Britain, 2000.

Schaefer, H. (1999): *Testing in the Dark: What to do with Poor or Missing Specifications*. Proceedings of EuroSTAR'99, Barcelona, Spain, 1999.

Wegener, J.; Fey, I. (1997): *Systematic Unit-Testing of Ada Programs*. Proceedings of the International Conference on Reliable Software Technologies – Ada-Europe'97, London, Great Britain, 1997, pp. 64 - 75.

Wegener, J., and Grochtmann, M. (1998): *Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing*. Real-Time Systems, Volume 15, No. 3, 1998, pp. 275-298.

Wegener, J., and Pitschinetz, R. (1995): *TESSY - An Overall Unit Testing Tool*. Proceedings of the 8th International Software Quality Week, San Franzisco, USA, 1995.

Wegener, J., Pohlheim, H., and Sthamer, H. (1999): *Testing the Temporal Behavior of Real-Time Tasks using Extended Evolutionary Algorithms.* Proceedings of EuroSTAR'99, Barcelona, Spain, 1999.