STAR'94, 8 - 12 May 1994, Washington, D.C.

Test Case Design Using Classification Trees

Matthias Grochtmann
Daimler-Benz AG
Forschung und Technik
Alt-Moabit 91b
D-10559 Berlin, Germany
Tel: +49 30 39 982-229
Fax: +49 30 39 982-107
email: grochtm@dbresearch-berlin.de

Abstract

The systematic test is an inevitable part of the verification and validation process for software. The most important prerequisite for a thorough software test is the design of relevant test cases, since they determine the kind and scope and hence the quality of the test. The classification-tree method and the graphical editor CTE (classification-tree editor) support the systematic design of black-box test cases. The classification-tree method is an approach to partition testing which uses a descriptive tree-like notation and which is especially suited for automation. Method and tool have already been tried out successfully on actual examples in various divisions of the Daimler-Benz Group. Three steps were found to be most important in the test case design for real-world applications: (1) Selecting test objects, (2) Designing a classification tree and (3) Combining classes to form test cases. In this paper a strategy to accomplish these steps systematically is outlined.

1. Introduction

The systematic test is an inevitable part of the verification and validation process for software. Testing is aimed at finding errors in the test object *and* giving confidence in its correct behaviour by executing the test object with selected input values.

The overall testing process can be structured into the following central test activities: During test case determination the input situations to be tested are defined. Concrete input values which meet the abstract test cases are determined during test data generation. For these test data the expected outputs are then predicted. The test object is run with the test data and thus the actual output values are produced. By comparing expected and actual values the test results are determined. Additionally, monitoring can give information on the behaviour of the test object during test execution.

The most important prerequisite for a thorough software test is the design of relevant test cases, since they determine the kind and scope of the test.

2. State of the Art

As experience shows, methods and tools are extremely helpful in real-world test problems (DEMI, GRAH). Methods and tools for white-box testing (i.e. testing based on the structure of the program itself) are widely used in practice. A typical example is branch testing which is automated by coverage analyzers.

However, there is a lack of methods and tools for test case design using a black-box approach (i.e. testing based on the functional specification). To improve this situation the classification-tree method and the classification-tree editor were developed by Daimler-Benz Research.

3. The Classification-Tree Method

The classification-tree method (GROC1) is a special approach to (black-box) partition testing partly using and improving ideas from the category-partition method defined by Ostrand and Balcer (OSTR).

By means of the classification-tree method, the input domain of a test object is regarded under various aspects assessed as relevant for the test. For each aspect, disjoint and complete classifications are formed. Classes resulting from these classifications may be further classified – even recursively. The stepwise partition of the input domain by means of classifications is represented graphically in the form of a tree. Subsequently, test cases are formed by combining classes of different classifications. This is done by using the tree as the head of a combination table in which the test cases are marked. When using the classification-tree method, the most important source of information for the tester is the functional specification of the given test object. A major advantage of the classification-tree method is that it turns test case design into a process comprising several structured and systematized parts – making it easy to handle, understandable and also documentable.

The use of the classification-tree method will be explained using a simple example. The test object is a Computer Vision System which should determine the size of different objects (Figure

1). The possible inputs are various building blocks. Appropriate aspects in this particular case would be, for example, the size, colour and shape of a block (Figure 2).

The classification based on the aspect ‘colour’ leads, for example, to a partition of the input domain into red, green and blue blocks, the classification based on shape produces a partition into circular, triangular and square 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 3).

In the combination table associated with the tree some possible test cases are marked as examples. Test case three, for instance, describes the test with a small blue isosceles triangle.

The classification-tree method is especially suited for automation since (a) it decomposes the test case design process into several steps which can be automated individually allowing the tool to

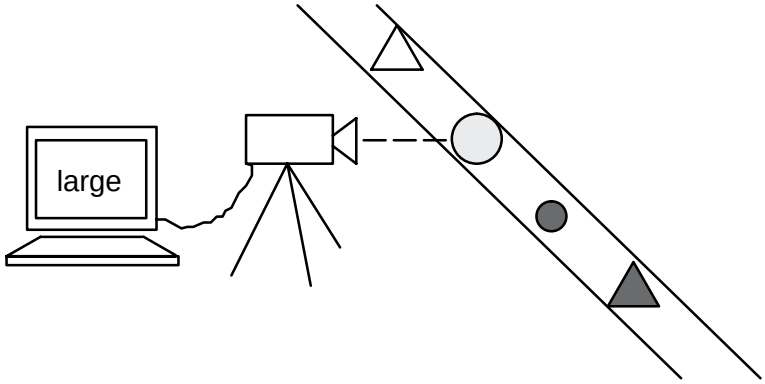


Figure 1: Computer Vision System

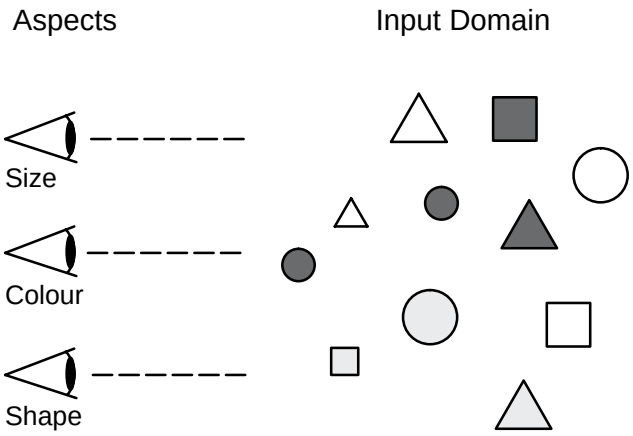


Figure 2: Aspects for Classification

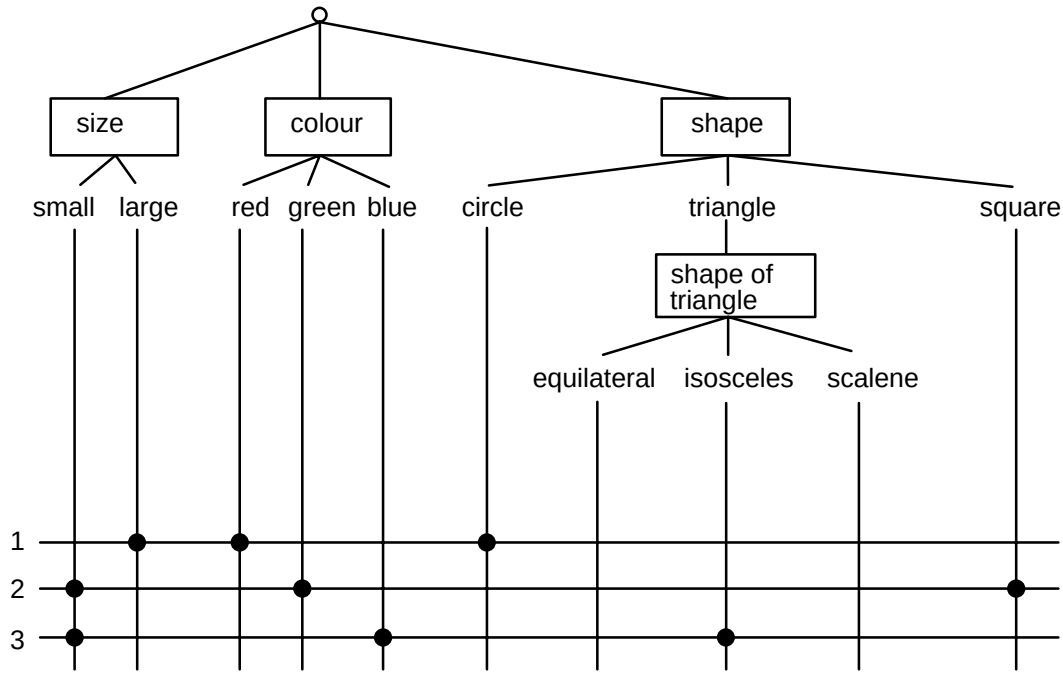


Figure 3: Classification Tree

appropriately guide the user and (b) it offers a graphical notation well suited for visualization in a modern graphical user interface.

4. The Classification-Tree Editor CTE

The classification-tree editor CTE is based on the classification-tree method and supports systematic and efficient test case determination for black-box testing (GROC2). The two main phases of the classification-tree method – design of a classification tree and definition of test cases in the table – are both supported by the tool. For each phase a suitable working area is provided.

The classification-tree editor CTE uses a separate window on the screen (Figure 4). In the upper part of the window there is a drawing area in which the user can build up a classification tree interactively (1). The lower part of the window depicts a corresponding table in which test cases can be marked interactively (2). Each test case row is numbered (3). The menu bar (4) offers access to several pull-down menus which provide various commands, e.g. for saving, editing and printing. The current working mode of the CTE is displayed in the status area (5). Pop-up menus are used to choose element-specific commands in the working area (6).

To give the user optimal support, editing is done in a syntax-directed and object-oriented way. Several functions are performed automatically. These include drawing of connections between tree elements, updating the combination table after changes in the tree and checking the syntactical consistency of table entries.

The CTE offers features which allow large-scale classification trees to be structured in order to support the test case design for large testing problems efficiently. This can be illustrated, for ex-

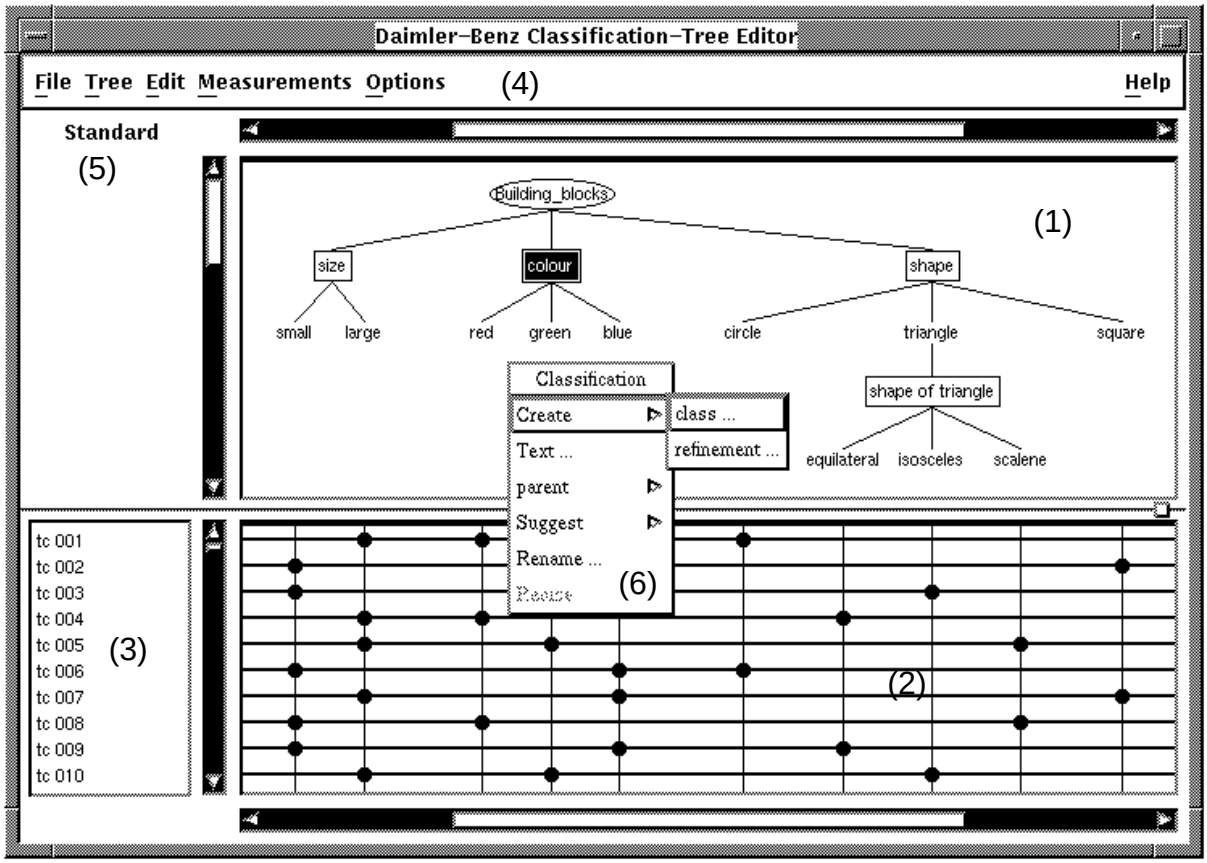


Figure 4: Classification-Tree Editor

ample, in Figure 5 where a screen dump of the CTE used for the test of a part of the CTE itself is shown. The test object is a procedure ‘is_line_covered_by_rectangle’ which should determine, whether a distinct line is covered by a given rectangle. This procedure is used in the CTE for determining the need of redrawing parts of the tree in case of window exposure events.

The main window (in the background) shows that the input domain of the test object is distinguished for instance according to the existence and degree of coverage and according to the positions of the end points P1 and P2 of the line. Refinement symbols are used at various places, in order to make more detailed differentiations in separate windows. For example, the child window in the foreground shows that the case that P1 lies outside the rectangle is further distinguished according to the position of P1 with respect to the rectangle and according to the distance of P1 from the rectangle. Other windows can be opened to show the complete tree and table. In this example, the test case determination process led to 49 test cases and one error in the test object could be detected.

As test documentation plays an important role in systematic testing, the CTE offers suitable support for this activity. For example, the test case design can be documented easily by printing out the trees and tables. Furthermore, the tool can automatically generate text versions of the test cases, based on the test case definition in the table. For example, the text version of test case 45 of the example is shown as:

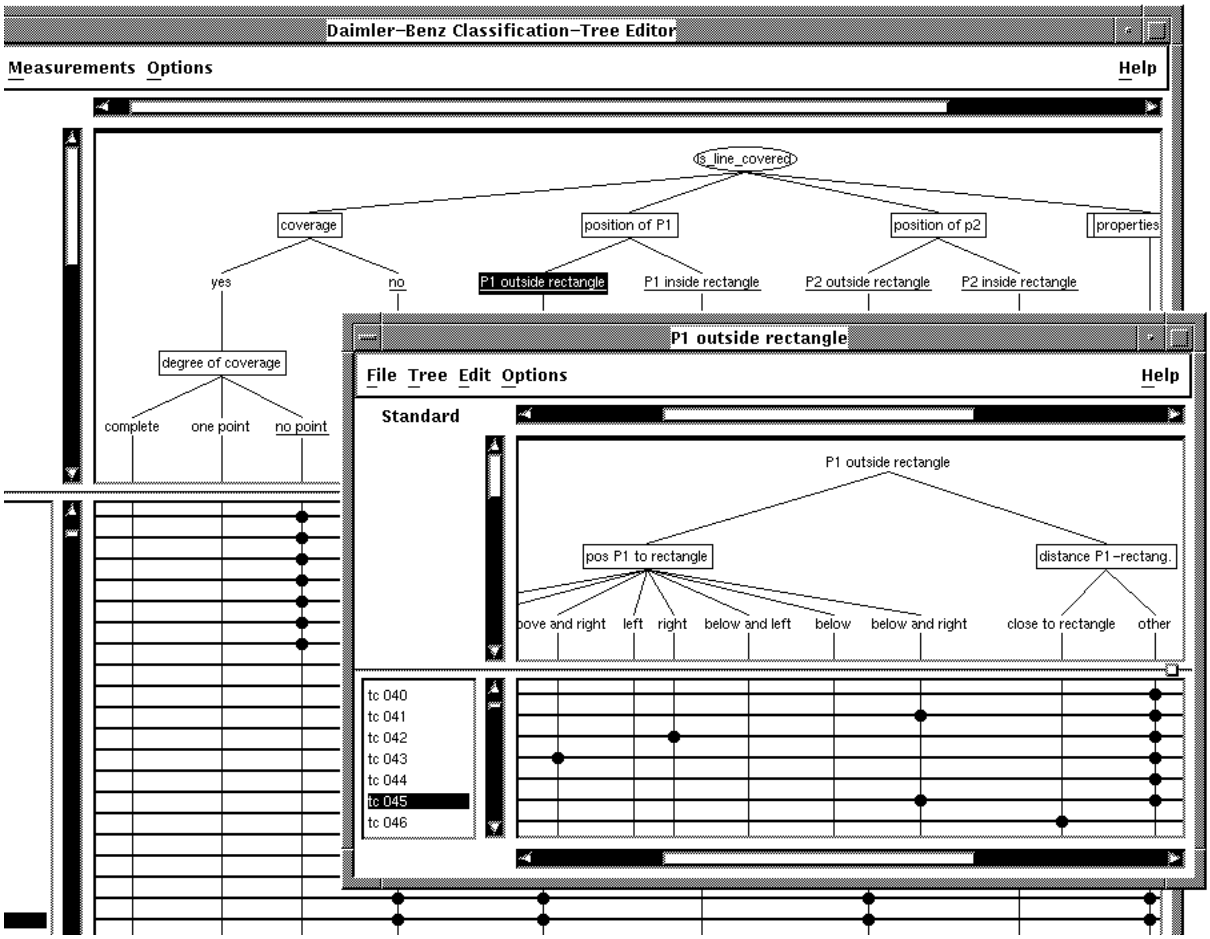


Figure 5: CTE Used for Large Testing Problem

Coverage: no
    Minimum distance line – rectangle: very small
Position of P1: P1 outside rectangle
    Position of P1 with respect to rectangle: below and right
    Distance of P1 from rectangle: other
Position of P2: P2 outside rectangle
    Position of P2 with respect to rectangle: left
    Distance of P2 from rectangle: other
Course of line: slanting
    Direction of line (P1 –> P2): bottom right –> top left
    Gradient of line: medium

These text versions serve on the one hand as documentation, on the other hand as a basis for the subsequent activities of software testing like the generation of concrete test data.

The CTE has been developed as an in-house tool on VAXstation under VMS and OSF/Motif in C. It is also available for Ultrix and for HP UX. A version for SUN/Solaris is planned.

5. Practical Experience

The method – in conjunction with the tool – has already been tried out successfully on actual examples in various divisions of the Daimler-Benz Group. As a result, some divisions recently started to use the method and tool in larger projects.

Examples for such real-world applications are a control system for the airfield lighting of an international airport, an identification system for automatic mail sorting machines and an integrated ship management system.

During the trials, the test documentation generated proved to be appropriate and useful. The fact that the method guides and supports testers but does not limit them was also positively judged by users.

The most important feature of the classification-tree method, however, was observed to be a good error detection rate. For example, in one module test for the identification system the number of test cases could be halved compared to an existing set of test cases and at the same time two errors were found. A detailed description of some results of the practical applications of the classification-tree method and the CTE can be found in GROC1.

6. The Three Most Important Steps in Practice

Three steps were found to be most important when using the classification-tree method for real-world applications:

1. Selecting test objects,
2. Designing a classification tree,
3. Combining classes to form test cases.

A large, real-world system cannot be tested reasonably with a single classification-tree, as such a tree would become too large to handle. Therefore, during step 1, the functionality of the system under test has to be divided into several separate test objects. This has to be done in such a way that each of the resulting test objects can be tested individually and that by testing all test objects the complete system is tested thoroughly.

During step 2, a classification tree has to be built up for each of the test objects, reflecting all test-relevant aspects. During step 3, test cases have to be generated covering the most important test situations for each test object.

7. Outline of a Testing Strategy

Although the classification-tree method and the CTE proved to be very useful for practical testing problems, a strategy for carrying out the three steps mentioned above systematically can further enhance the testing process.

A first approach to such a testing strategy aimed primarily at interactive business software has been developed from experience gained in the process of test case determination for parts of a

management system for a large educational institution. Important tasks of this window-based system are, for example, registration of students, management of lodging and scheduling of courses and teachers.

7.1 Selecting Test Objects

The following procedure seems to be useful to carry out the first step *selecting test objects* systematically:

- Each window is tested separately.
- The dialogue of each window is analyzed. During the case study state transition diagrams were used successfully for this task.

Test objects are then selected by observing the following rules:

- Each dialogue state transition is a candidate for being a test object. The functionality of the window is, thus, covered completely in a systematic way.
- Complex transitions are further divided to get simpler test objects. To do this, the use of integration aspects is often useful. For example, if a list is processed in a state transition and each item in the list is treated in the same way, then two test objects are selected: The first test object is the processing of a single, arbitrary item in the list, the second test object is the processing of the complete list. However, during the test of the second test object only aspects relevant to the list as a whole are considered. Aspects relevant to individual items are tested with the first, smaller test object only. Thus each of the test objects is testable with reasonable cost without loosing any test-relevant aspects.
- On the other hand, simple dialogue transitions which represent the same functionality accessible from different dialogue states are joined to reduce the number of test objects. An important aspect for such a compound test object is the dialogue state it is started from.

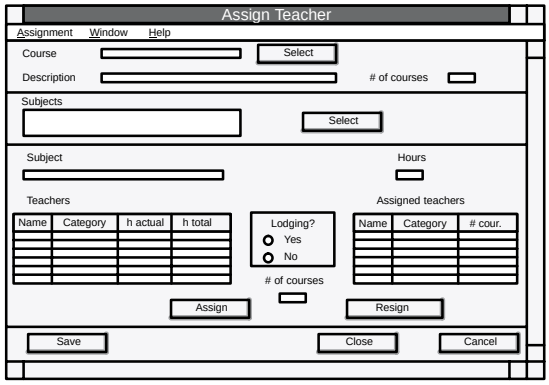
Figure 6 shows an example from the case study: The window “Assign teacher” was analyzed by means of a state transition diagram and 15 test objects were selected.

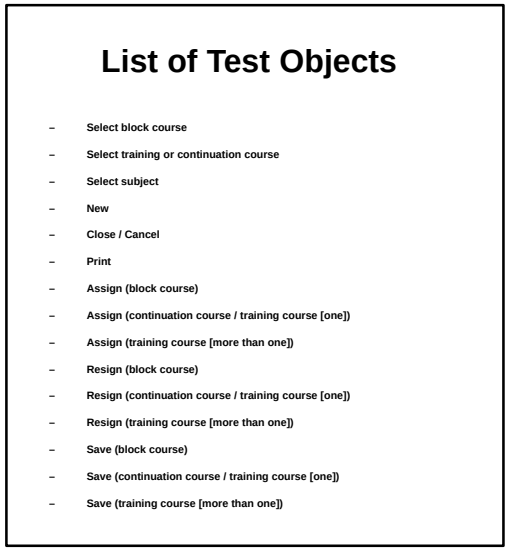
7.2 Designing a Classification Tree

During the second step *designing a classification tree*, one classification tree is produced for each of the selected test objects. Experience from the case study shows that the most important sources of information are the functional specification and the window descriptions. The relational data model was not that important.

The following approaches might be useful to design a tree:

- Transform aspects which are mentioned explicitly in the specification into classifications.
- Use values which are mentioned explicitly in the specification as classes.
- Introduce aspects for input parameters as well as for the system state. Consider also aspects concerning the relationship between both.
- Specific data structures imply specific aspects. For example, a list might imply the aspects “length of list” and “sorting”.





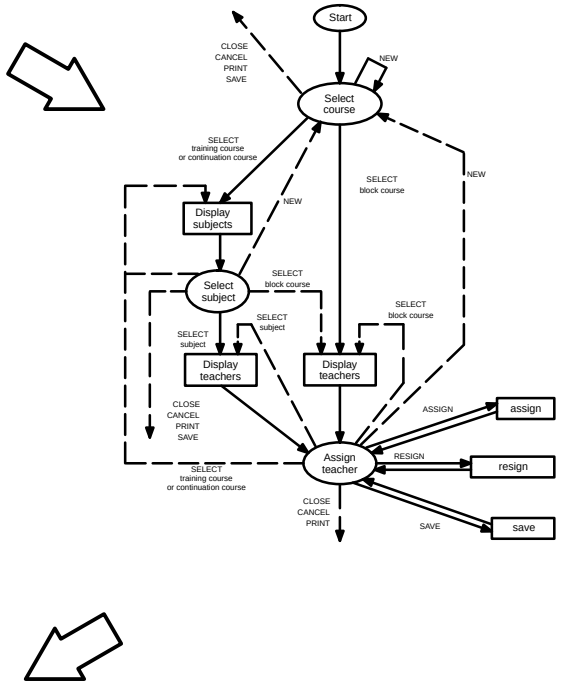


Figure 6: Selecting Test Objects

– Sometimes knowledge of the implementation is useful to decide whether a specific aspect is of importance.

It has to be noted, however, that creativity and experience are especially required to design an adequate classification tree.

7.3 Combining Classes to Form Test Cases

The classification tree is then used as the head of a combination table in which relevant class combinations are defined as test cases during step 3 *combining classes to form test cases*.

The main point here is the common distinction between *valid* and *invalid* test cases. Valid test cases describe input situations which are processed regularly by the test object. Invalid test cases provoke an error handling reaction by the test object. Valid and invalid classes and class combinations are defined analogously. Note that *indifferent* situations where the outcome (error reaction *or* normal processing) is left open are also possible.

Rules for the combination of classes to test cases include:

- Each invalid class is combined individually with an indifferent combination of classes of the other classifications to form invalid test cases.
- Valid and indifferent classes are combined minimally to form valid test cases, i.e. the lowest possible number of valid test cases is selected in such a way that each valid or indifferent class is used in at least one test case.
- More valid test case combinations can be used to enhance the test quality.
- If a number of indifferent classes together provoke an error handling reaction the first-mentioned rule should be used accordingly to form more invalid test cases.

Note that in some cases rules cannot be met completely due to the construction of the specific tree.

The underlying assumption behind these rules is that the first invalid input condition recognized by a test object will lead instantly to an error handling reaction and that other conditions will be ignored in that case. Therefore, invalid classes have to be tested individually. However, it is assumed that all classes in a valid test case contribute to the normal behaviour of the test object. Hence, a number of classes can be tested together in a single valid test case.

Figure 7 shows the top-level diagram of a classification tree from the case study where one of the 15 test objects for the window “Assign teacher” was tested. This test of the assignment of a teacher to a training course led to 14 test cases where cases 1 to 4 are invalid cases, 5 to 14 valid cases. Two specification errors could be found during this test case determination.

8. Conclusion

In future, more case studies are planned to provide more information on how to improve the test strategy further. This information will be used to work in two directions:

- On the one hand, the strategy will be generalized to be applicable not only to business software but to other kinds of systems, too. This affects, in particular, step 1 *selecting test objects.*
- On the other hand, the effectiveness of the strategy will be improved. For example, strategies for testing sequences and testing based on the data model as well as additional tool support are planned. The development of catalogues of aspects and checklists can support the tester additionally.

Recently, two related projects were started in this area. In the first project, a larger case study is conducted for a successor of the educational management system. This project will later be continued to produce an enhanced strategy for business software. The second project is aimed at developing a general strategy for testing large systems on the system level.

9. References

DEMI DeMillo, R.A., McCracken, W.M., Martin, R.J., Passafiume, J.F., Software Testing and Evaluation, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1987.

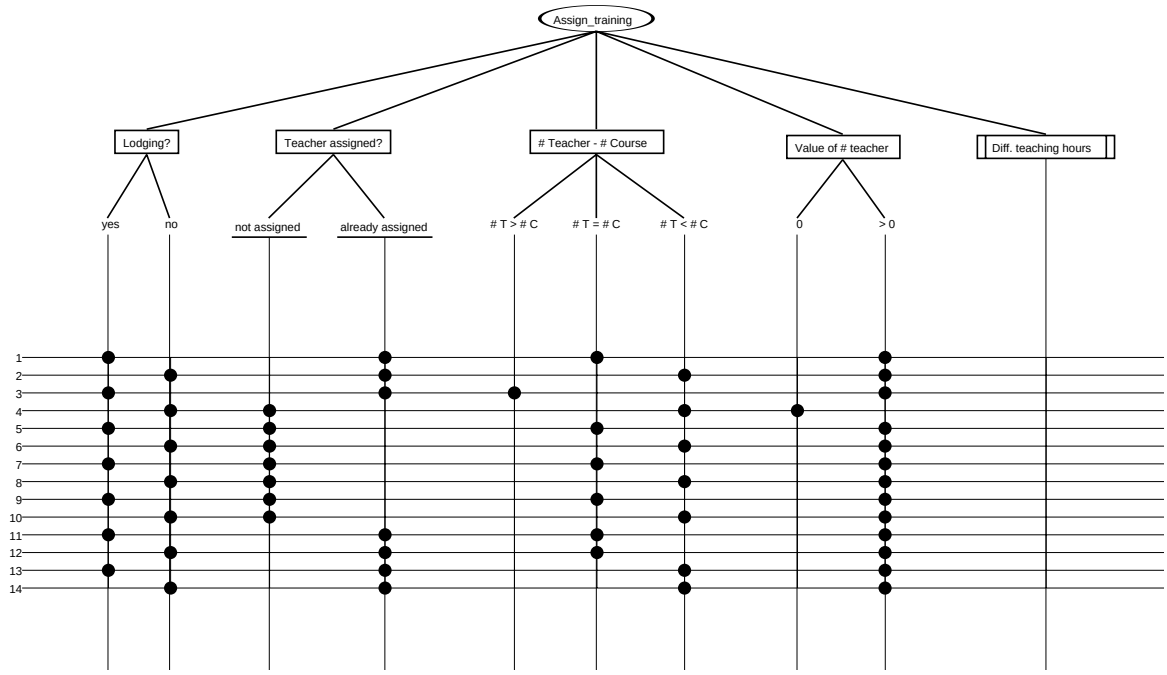


Figure 7: Classification Tree “Assign_training” (Top-Level Diagram)

GRAH Graham, D.R. (Ed.), Computer-Aided Software Testing: The CAST Report, Unicom Seminars Ltd., Middlesex, UK, 1991.

GROC1 Grochtmann, M., Grimm, K., Classification Trees for Partition Testing, Software Testing, Verification & Reliability, Volume 3, Number 2, June 1993, Wiley, pp. 63 - 82.

GROC2 Grochtmann, M., Grimm, K., Wegener, J., Tool-Supported Test Case Design for Black-Box Testing by Means of the Classification-Tree Editor, EuroSTAR ’93 – 1st European International Conference on Software Testing Analysis and Review, 25 - 28 October 1993, London, UK

OSTR Ostrand, T., Balcer, M., The Category-Partition Method for Specifying and Generating Functional Tests, Communications of the ACM, Volume 31, Number 6, June 1988, pp. 676 - 686.