Reuse of Classification Tree Models for Complex Software Projects

S. Alekseev · P. Tollkühn · P. Palaga [1]

Nokia Siemens Networks

Z. R. Dai · A. Hoffmann · A. Rennoch · I. Schieferdecker

Fraunhofer FOKUS

Abstract

Customizable mobile services are needed to address ever changing service requirements and shortened service development and deployment times. However, service flexibility is a natural evil for quality assurance: the service functionality and the overall system can no longer be guaranteed by the developer and tester. Instead, a post-customization test phase is needed, which is flexible enough to provide a good coverage of the customized service features. In order to make this post-customization testing efficient, reuse of rather general tests is proposed and outlined in this paper.

1 Introduction

Today's mobile services providers operate in rapidly changing markets. As a consequence of this, they are forced to react quickly. Especially, they must be able to implement new business models and billing services in a short time. The same holds for the rearrangement of the existing services. The usual approach in such a situation is that the service software is provided with many customizable parameters and customization tools which allow quick and effective customisation.

Once released, the customizable service does not need to be recompiled anymore and it can be adapted by several mechanisms including

1 On leave to Max-Planck-Institute, Berlin

- Data Administration: table administration (e.g. tariffs, announcements) or parameter modification (e.g. bonus points),
- Administration of predefined service logics: enabling/disabling of services or for-matting of predefined parameters (e.g. country specific adjustment of number normalization),
- Modification of service logics: enabling/disabling/reordering of logical components (e.g. disabling of an announcement), or
- Creation of new service logics (e.g. new/extended voice menu or tariff model).

To guarantee the smooth operation of the service, comprehensive tests of possible configurations and modifications are needed. Sections 2 and 3 give a short overview of the Classification Tree Method (CTM) and its adaptation for complex software systems. However, the customizable services exceed the scope of even such methods. The reason is that the customizable services introduce a new dimension of complexity which the CTM cannot handle. Namely, the reordering of logical components makes it impossible to represent such a service as a classification tree. Thus it is impossible just to generate the test cases through the combinations of all possible customization instances.

Instead, each release of the customizable service is provided with one or more typical service configurations, which are called *service blueprints*. The blueprint is taken as a starting point for the customisations of a service. The solution proposed in Section 4 is based on several points:

- Blueprints are taken as a starting point for the customization of a service.
- Blueprints are associated with test models (generated by use of CTM).
- The blueprint tests are reused and refined along the service customization.

The reuse and refinement of tests along blueprints is a new challenge in the area of software engineering and can be called adaptability of tests. The goal is to select such a subset of test scenarios, which allows for optimal coverage of customization requirements with minimal test effort.

2 Classification Tree Method

The Classification Tree Method (CTM) is an approach for systematic design of tests [Gro93]. It is a modification of the category-partition method defined by Ostrand and Balcer [Ost88]. The CTM suits the black-box testing paradigm well as only the input and output domains of the system under test are needed for the analysis. The basic idea behind CTM is to abstract from the concrete test data and to take only those aspects of the input/output domain into consideration which are relevant to the test. In this way, the domain of e.g. each input parameter (classification) is systematically segmented into separate subsets, called classes. The test cases are generated by combining classes from different classifications. Thus, every test case constitutes a unique combination of classes.

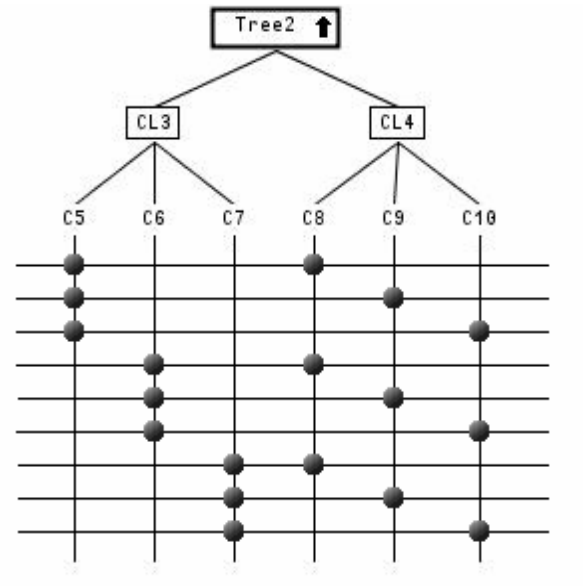

Fig. 1 *Classification Tree*

The relationships between classes, classifications and test cases can be visualized in a classification tree whose leafs represent the possible values (classes) of the parameters (classifications). The test cases can be represented in a combination matrix below the tree. Its rows correspond to test cases and its columns are linked to the leaf classes. The marks in appropriate cells of the matrix show the combination of the classes for a given test case. To put it more precisely, only classification nodes that have leaves need to be considered for the test case generation.

There is a freely available tool called CTE XL for editing classification trees based on CTM [Gro95], [Leh00] and [Weg93b]. With CTE XL, the test case table can be built automatically.

Figure 1 shows an example of a classification tree created with CTE XL. The name of the root node can be defined by the user and has only an informal meaning. The nodes CL3 and CL4 are classifications. Their children(C5 – C10) are equivalence classes. The test case table can be seen below the tree: every test case selects an equivalence class per classification such as C5 and C8 for test case 1, C5 and C9 for test case 2 and so forth.

A further important benefit of the CTM is its ability to mirror not only the input/output domains of the SUT, but also the functional requirements which should be covered by the SUT [Gro94], [Weg93a]. An arbitrary node of a classification tree can be marked as covering certain requirement(s) from the requirement specification. Every test case inherits the requirements along its classification tree paths. Thus for every test case, it can easily be shown which requirements it covers and vice versa, e.g. an overall coverage of the requirements specification can be calculated.

3 Classification Tree Modelling

Although the CTM is quite old, it has found only limited application in the industry. The main reason for this is probably the fact that the classification trees for common real world systems can become very big and untraceable, resulting in too many test cases which must be reduced either manually or through sophisticated dependency rules. [Ale07] proposes a modification of the CTM to eliminate most of these limitations. The experience at Siemens Networks demonstrates the usability of the modified CTM for the testing of complex software systems.

In the case of testing customized services, it is important at the beginning of the customization to detect which requirement can be reused, which are new and which should be modified. In the next step ,the models are adapted and references to the new or modified requirements are added. The last step is the identification of test cases.

The use of classification tree models allows an exact matching of necessary test cases. The creation of CTE trees for blueprints is a straightforward procedure. To do so, the scenarios need to be mapped to classifications and classes in the tree. Sub-trees can be used for sub-requirement specifications.

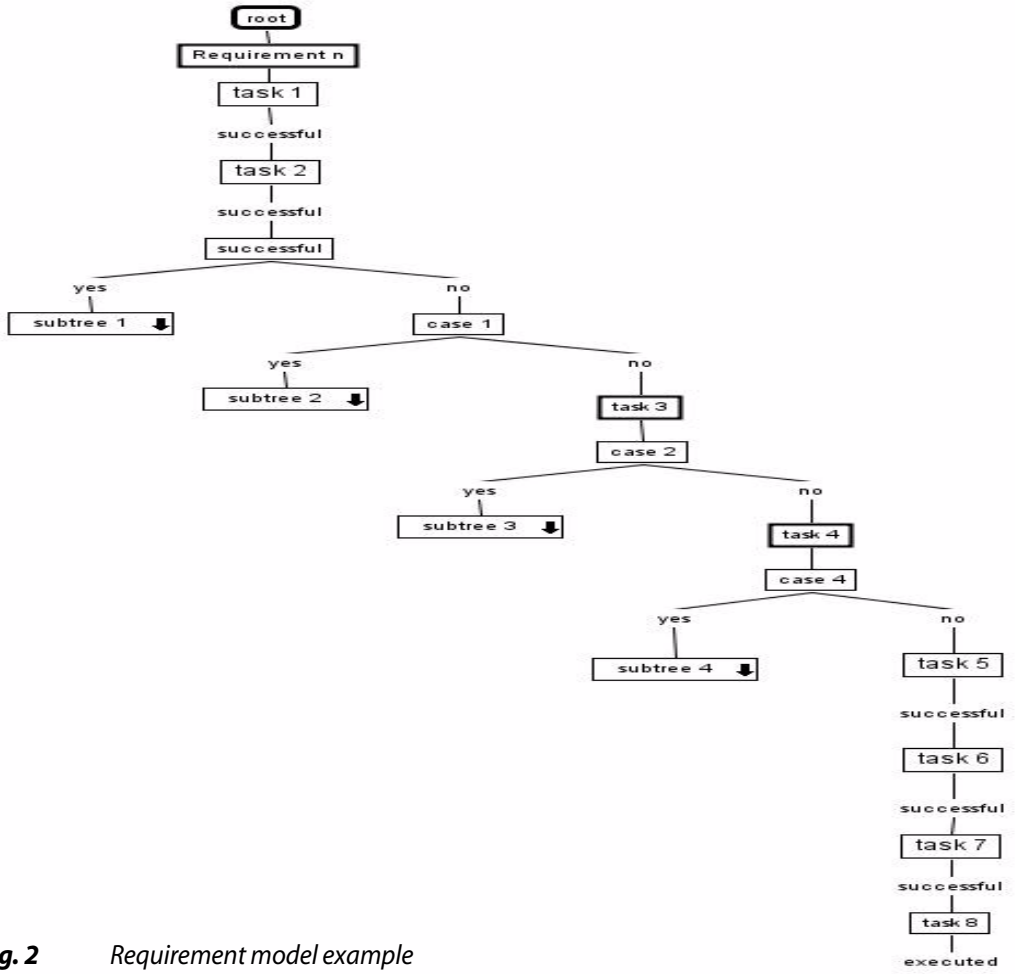

Fig. 2 *Requirement model example*

Figure 2 shows the CTE of an example blueprint feature. The tree has branches which are conditions defined in the requirements specification. For each service feature (or sub-feature) blueprint, a CTE tree is specified. The combination of feature blueprints can be realized also by the CTE trees. To do so, multiple trees can be merged into a bigger tree. An overall node is needed in order to connect the sub-trees together into a big tree. Features can also be assigned (i.e. tagged) by weight. According to the relevance of the features to be tested, priorities can be assigned in order to show their importance for the test. Weight assignments can be done in a CTE tree node as comments. Later on, this information is collected while evaluating the weights and priorities of a feature which should be tested. By means of the classification trees, the structure of the requirements specifications (R-Spec) can be derived and understood easily by its user.

The graphical representation of a CTE tree provides good means for test data selection. When creating test data sets in classification trees, the CTE XL editor provides means to automatically generate and combine test data according to the needs of the tester. Automatic test data generation and manual test data creation are supported in the tool. Also, existing test cases can be imported into new classification trees for reuse of the old test cases. This makes test data generation/specification more efficient.

4 Reuse of Test Models

Whenever a service is to be customized and deployed, basically a typical development process can be used. At first, the requirements specification is defined. From the requirement specification the tests are derived, implemented and executed. To reduce the costs of software based on customizable components, the classical development process is adapted. It is necessary to reuse test cases from main service releases. There are two main aspects our approach that allow:

1. modelling of the test cases from requirements with CTM and
2. the reuse of requirements and of associated test cases in a service customization.

The CTM has been chosen because it allows the identification of test cases through customization of requirements as depicted in Figure 3.

In the following, we assume the existence of a complete set of tree models for the blueprint and discuss their reuse. In principle, the following types of service customization exist:

- options for (de)activation of pre-defined features (in opposite to the blueprint implementation),
- selection of elements from pre-defined enumeration lists (e.g. menu language),

- specification of value representatives (e.g. number of menu repetitions, announcement text), formats and (in)valid representatives (e.g. phone numbers),
- logic changes (e.g. reordering of functional blocks).

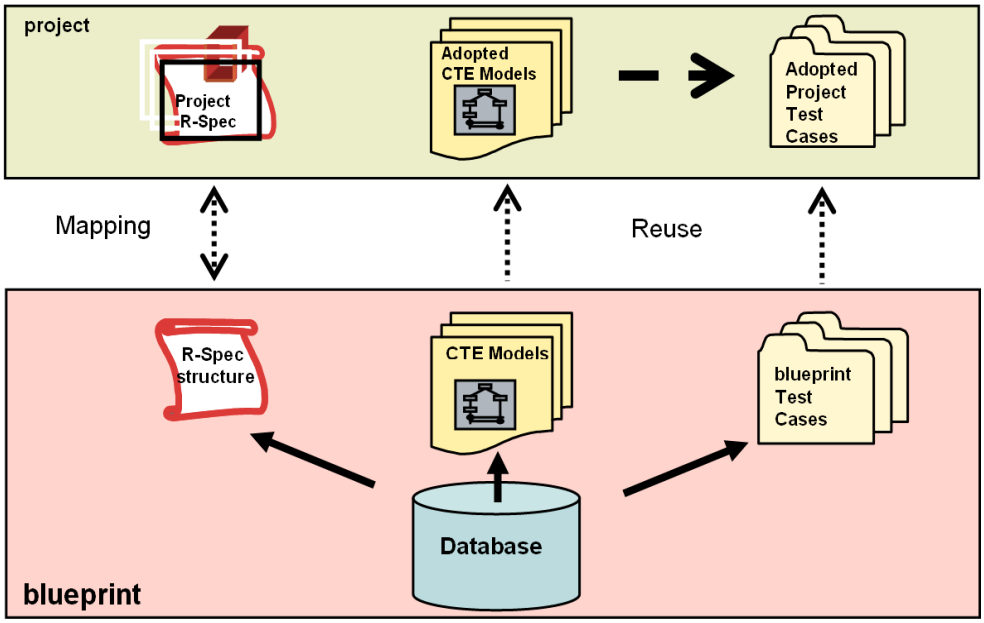

Fig. 3 *Process overview*

Hence for the test customization, we have to distinct three different cases: (a) deletion, (b) insertion, and (c) modification of tree nodes in the classification tree:

Figure 4 illustrates the reduction of a classification tree model due to the deletion of a requirement. Following this modification, the number of tests is decreasing and one test needs to be modified.

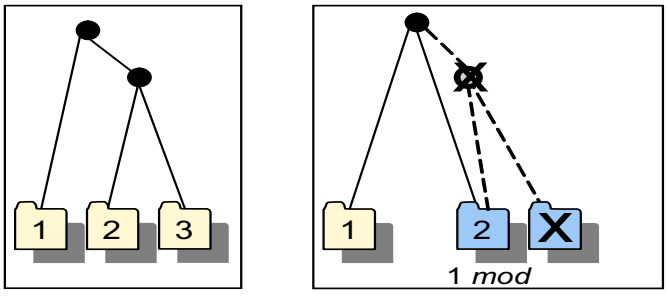

Fig. 4 *Reduction of requirements*

In principle there are two different situations if an additional project requirement appears in a classification tree model: (a) the new requirement will be added below the last tree branching and effects only one test case that needs to be modified, the number of tests is increasing; (b) the new requirement will be added

before the last tree branching and causes at least one new test and modifies all tests that have been attached after the position of the new requirement that has been introduced. Both situations are illustrated in Figures 5 and 6.

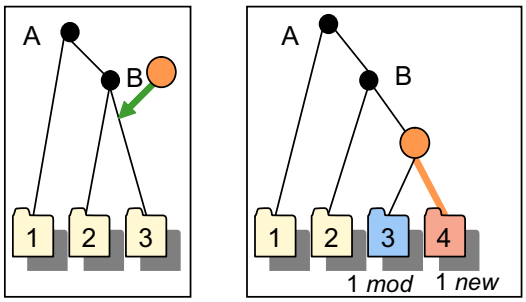

Fig. 5 *Additional requirement (case a)*

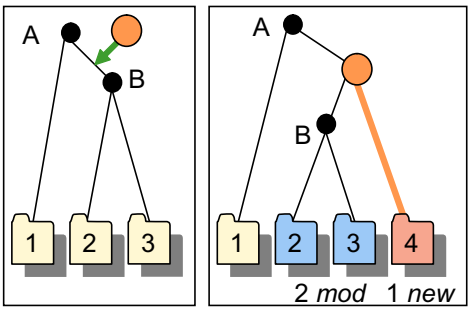

Fig. 6 *Additional requirement (case b)*

Finally modification of a classification is possible and affects all tests with subsequent nodes. In case of combined changes of the tree models the different rules need to be considered, e.g. Figure 7 exposes a redefinition of a parent node element for a classification that leads three modified tests (O1, O2, O3) and one new test case (N1).

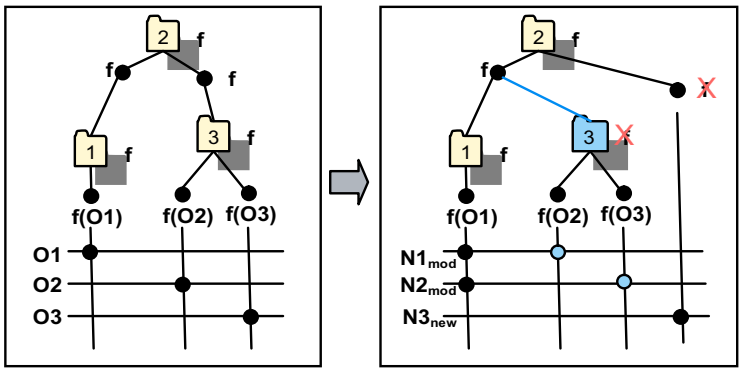

Fig. 7 *Redefinition of the parent tree node*

We are currently implementing a supporting algorithm for test customization that uses a freeze marking approach to allow an identification of old, new and modified test case. In the algorithm all tree elements get a "freeze" mark and the list of test case names is stored. After customizing the model, we re-generate the test cases in order to allow a comparison with the original tree. It is an "old" test if all elements in test case path still have freeze-marks. If at least one element has no freeze-mark, the test has been modified, and it is a new test if the final element has no freeze-mark (see Figure 8).

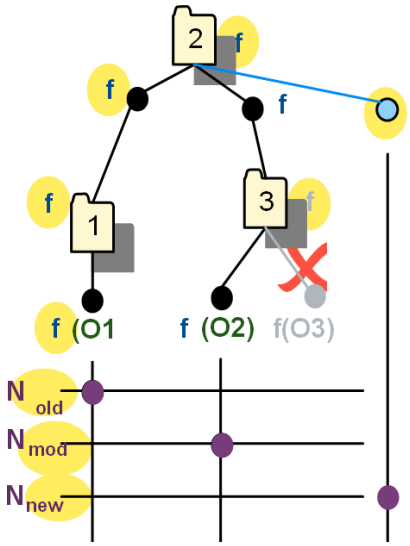

Fig. 8 *Distinction between test cases*

5 Conclusion

This paper presents a new approach of test reuse and refinement for customized services. For that, a specific use of the classification tree method is being used, in which the requirements specification is directly reflected in the structure of the classification tree. The resulting test model relates to the service blueprint, which defines the service configuration under test.

This allows an easier customization of the tests in correspondence to the service customization: changes to a blueprint translate into deletion, modification and addition of nodes in the classification tree.

In result, the blueprint based test reuse and refinement based on the classification tree method preserves the advantages of test models – the explicit and objective denotation of test cases – and adds to it the traceability in relation to service requirements combined with a clearness in test specification and documentation.

In further work, we have also investigated the weighting of nodes in the classification tree, so as to reflect importance, risk, etc. of service features. These weights allow to control better the selection process for test cases. Future work will address a further spreading of the presented methodology and a more fine-grained analysis of test weights.

References

[Ale07] Alekseev, S. Tiede, R. and Tollkühn, P. Systematic Approach for using the Classification Tree Method for Testing Complex Software-Systems. In Proceeding of the IASTED Conference on Software Engineering, Innsbruck, Austria, February 2007.

[Gro93] Grochtmann, M. und Grimm, K. Classification Trees for Partition Testing. In Software Testing, Verification & Reliability, pages 63 – 62, vol. 3., 1993.

[Gro94] Grochtmann, M. Test Case Design using Classification Trees. In Proceedings of the International Conference on Software Testing Analysis & Review (STAR), Washington D.C., USA, pages 95 – 102, May 1994.

[Gro95] Grochtmann, M.; Wegener, J. Graph Theory in the Control Flow Analysis of the large time critical Applications. In Proceedings of the 8th International Software Quality Week, San Francisco, USA, May 1995.

[Leh00] Lehmann, E.; Wegener, J. Test Case Design by Means of the CTE XL. In Proceedings of the 8th European International Conference on Software Testing, Analysis & Review, Copenhagen, Denmark, EuroSTAR 2000, December 2000.

[Ost88] Ostrand, T., and Balcer, M. The Category-Partition Method for Specifying and Generating Functional Tests. Communications of the ACM, pages 676–686, 1988.

[Weg93a] Wegener, J. Grochtmann, M. Werkzeugunterstützte Testfallermittlung für den funktionalen Test mit dem Klassifikationsbaum-Editor CTE. In Proceedings der GIFachtagung Softwaretechnik 93, Dortmund, Germany, pages 95 – 102, November 1993.

[Weg93b] Wegener, J. Grochtmann, M.; Grimm. Tool-Supported Test Case Design for Black Box Testing by Means of the Classification- Tree Editor. In Proceedings of the 1st European International Conference on Software Testing Analysis & Review, London, Great Britain, pages 169 – 176. EuroSTAR, 1993.