Testing Customizable Software for Telecommunication Services

Sergej Alekseev, Peter Tollkühn
Nokia Siemens Networks GmbH & Ko KG, Germany, 13629 Berlin
E-Mail: {sergej.alekseev|peter.tollkuehn}@nsn.com

Zhen Ru Dai, Andreas Hoffmann, Axel Rennoch, Ina Schieferdecker
Fraunhofer Research Institute for Open Communication Systems, Kaiserin-Augusta-Allee 31, Germany, 10589 Berlin
E-Mail: {zhen.ru.dai|andreas.hoffmann|axel.rennoch|ina.schieferdecker}@fokus.fraunhofer.de

Abstract

Today's telecommunication providers demand for flexible software solution with the goal to reduce the time to market for their marketing ideas significantly. The contribution of this paper is a new concept for testing the software adaptability. We present an approach which allows for testing the software with focusing on the coverage of customization possibilities. Additionally, we present the adaptation of the development process and tools which are necessary for the realization of the projects based on highly customizable service software.

Key Words: Requirements Specifications, Design Tools and Techniques, Software Testing

1 Introduction

Nowadays, for mobile service providers it is very important to have a software solution which allows for flexible rearranging of the existing charging structures and fast implementation of new business and billing models for their telecommunication services currently operated. Such solutions should significantly shorten the time to market for new billing services. Recently, this approach has come into the focus of the telecom industry and is currently attracting more and more attention. To reach this goal, the service software gets many customizable parameters and tools which allow the provider to realize his business ideas by himself in a very short time. We present in this paper the new problem of *testing the software adaptability* and an approach which allows for testing the software with focusing on the coverage of customization possibilities.

In the next section the problem and how this problem can be tackled is described. This is followed by a description of a typical customization lifecycle (section 3) and our approach for testing software adaptability (section 4). In Section 5 we propose an adaptation of the development process followed by a section on the tool support. The paper is concluded by a summary with a brief outlook on the future work.

2 Software Adaptability - Definitions and Problem Description

Software adaptability (or customization) is the capability of software to be adapted to customer's need just by reconfiguration and parameterization, i.e. without recompilation of source code. The problem discussed in this paper is how to test the variations resulting from the customization process of the software.

A *customization mechanism* is the implementation of the customization possibilities. It is a basic mean to modify the software according to the customer's requirements. Customization mechanisms can be grouped using several levels according to their capabilities. For example: Level 1 - the customization mechanisms for data administration, Level 2 - reordering of the predefined logics, Level 3 - modification of logics etc.

A software *feature* is defined as the well-defined functional capability of the software product and is often related to one or more functional system requirements.

The creation of tests for highly customizable software is quite complex. The amount of system features and their customization possibilities lead easily to a huge number of combinations. In figure 1, six bullets are shown indicating the combination of customization levels and feature numbers. The grey arrow illustrates the drastic effort increase according to the number of customization levels and combined features. The total amount of tests increases exponentially due to the combinations of features and its customization possibilities. In order to control the growing complexity and resource consumption, tests have to be selected systematically. In our approach described in the following sections, this can be achieved by annotating the test cases with proper weight values and building uses cases from those ones with the highest values.

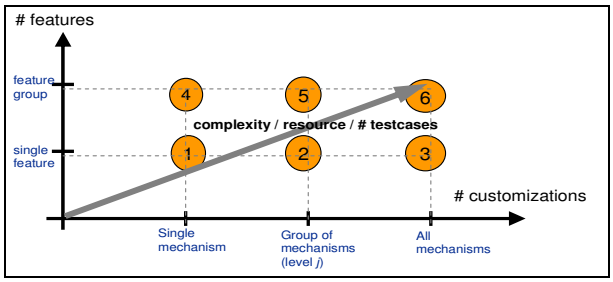

Figure 1: Customization complexity

To illustrate the growing of the complexity the following example is considered:

We assume two groups of functional features: G1 and G2. Each group includes two features: G1 = {f1, f2} and G2 = {f3, f4}. For customization of this feature two mechanisms groups, each with three mechanisms, are introduced: M1 = {m1, m2, m3} and M2 = {m4, m5, m6}.

To test the customization of four single features f1 … f4 with each of the six single mechanisms 24 tests are necessary (point 1, figure 1):

4 (feat.) X 6 (mech.) = 24 (tests)

To test the influence if two or more mechanisms from each group (3 unordered combinations with 2 mechanisms plus one combination with 3 mech.) are used in parallel for customization of the single feature 64 tests are necessary (point 2, figure 1):

4 (feat.) X 4 (comb.) X 4 (comb.) = 64 (tests)

The test of four single features with all possible combinations demands 256 tests (point 3, figure 1):

4 (feat.) X 64 (mech. comb.) = 256 (tests)

To test the customization for all possible feature combinations with all possible combinations of mechanisms 1024 tests are necessary:

16 (feat. comb.) X 64 (mech. comb.) = 1024 (tests)

The number of tests grows exponentially on the number of features and customization mechanisms. In the real projects the number of the customization mechanisms and number of feature which have to be customized is much higher as this example. For example for 10 features and 10 mechanisms 1.048.576 possible tests can be build. This number of tests can not be executed and will be never accepted in a real project.

3 Customization Life Cycle

The life cycle of a customizable service is shown in figure 2.

The service package consists of the core part and the customizable part. The core part is a well defined but fixed set of logics which provides the

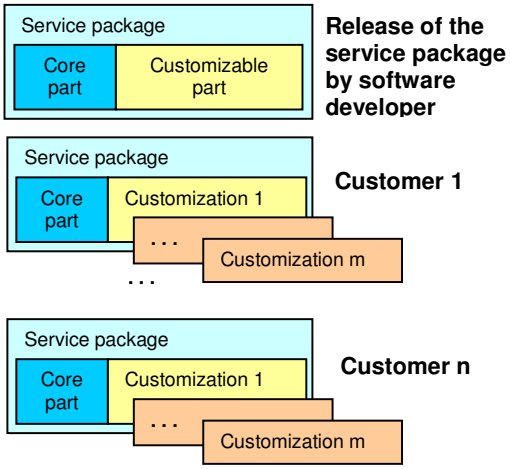

Figure 2: Customization life cycle

basic functionality. The customizable part is a package which can be adapted to the service provider's needs by customization mechanisms such as:

- Data administration
  - Table administration (e.g. tariffs, announcements)
  - Parameter modification (e.g. bonus points)
- Administration of predefined service logics
  - Enabling/disabling
  - Formatting of predefined parameters (e. g. country specific adjustment of number normalization)
- Modification of service logics
  - Enabling/disabling of logical component (e.g. disabling of an announcement)
  - Reordering of logical components
- Creation of new service logics
  - New/extended voice menu
  - New tariff model

The customizable part of the service package can be used by the provider to define new capabilities without modification and recompilation of source code. The life cycle shown in figure 2 requires that the service package is released only once by the software company. After that a customer is allowed to adapt and modify the service logic by himself by using appropriate customization tools. However, the service logic modified by a customer should still work accurately and not introduce errors or cause negative side effects on the system. To guarantee the smooth operation of the service, comprehensive tests of possible customizations and modifications are needed. This is a new challenge in the area of software engineering and can be called *adaptability testing*. The goal is to select such a subset of test scenarios which allows

for optimal coverage of customization requirements with minimal test effort.

In the following we present the results from a real project at Siemens Networks. Table 1 shows some selected results.

Number of Functional Requirements	400
Number of Customization mechanisms	30
Number of Customization sub-mechanisms	140
Number of customizable components	120
Number of Test Cases	1400

Table 1: Project Data

4 Testing of Software Adaptability

In this section we describe an approach for building the test specification and its optimization.

4.1. Test Modelling

For creation of the test specifications the development process has to include a modelling step. For this step the classification tree method (CTM) has been chosen, because it allows for the identification of test cases through customization of requirements. The idea of the classification tree method is descended from the category-partition method defined by Ostrand and Balcer [1] in 1988. The CTM supports black-box test case design in regard to systematic and complete segmentation of the input domain into a finite number of mutually disjoint equivalence classes [2].

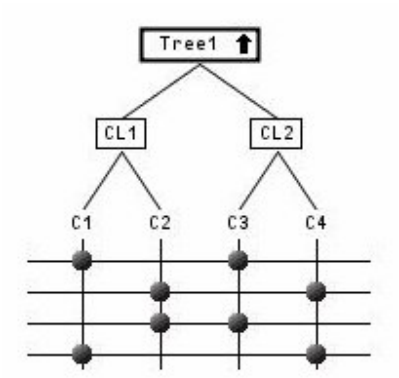

Figure 3: Classification-tree

Figure 3 shows an exemplary classification-tree. The root name of the tree can be defined by the user and has informal meaning. The nodes CL1 and CL2 are classifications. The children of them (C1 - C4) are equivalence classes. The resulting test cases must consist in exactly one child of every classification. More precisely only classification nodes that have leafs have to be considered for the test case generation. CTM can build a table of all combinations of the equivalence classes. This table is shown under the classification-tree. The points represent selected equivalence classes. Classes of the same parent can not be selected at the same time. Every line of the table is one generated test case that is unique per table.

For supporting CTM, a couple of tools exist which are called Classification Tree Editor (CTE) [4], [6]. The CTM has been successfully applied for modeling of functional requirements [3], [5] and adapted for complex software systems [7].

4.2. Building Use Cases

As described above, it is impossible to test all combinations of customization mechanisms and features. Thus, the goal of the use cases is to address all customization mechanisms and all features, but not all combinations of them. In order to be able to find and build customization use cases in our approach, we start with the identification of the most important customization mechanisms which are not yet covered by existing use cases. The importance of a customization mechanism may be estimated by certain weighting approaches (e.g. relevance ranking analysis) or simply by counting the number of occurrence of features associated with that customization mechanism (see Figure 4).

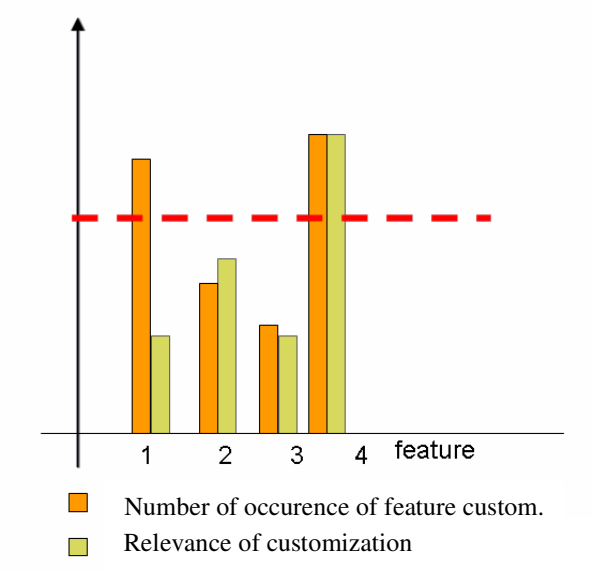

Figure 4: Relevance ranking analysis

For this purpose, either classification trees (e.g. figure 6) or tabular formats (figure 5) may be used. In the former case, weight values are assigned to the tree nodes, whereas in the latter case a matrix with the list of mechanisms on each customization

level and a list of features with customization variations are recommended. Within this selected customization mechanism the feature with the highest weight will constitute the fundament of the customization use case.

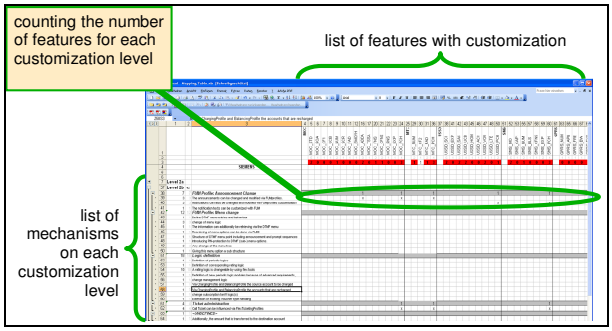

Figure 5: Selection of customization mechanisms

Since there is no formal criterion for the termination of the procedural method the test engineer has to decide by himself on the completion of the use case building. It has to be taken into account that each additional customization feature will increase the effort and resource consumption, but may not necessarily improve the test coverage of customization mechanism.

4.3. Test Case Weighting

The determination and assignment of path weight values for alternative branches or decisions to be selected within the test definition process shall lead to the following two goals:

- Optimization: A prioritization of test cases in order to find a sequence for the execution of tests during the test campaign.
- Selection: A reduction of the amount of test cases by the definition and application of a threshold value with the intention that only those test cases with a weight value above the threshold will be executed.

The most difficult question herein is to find the appropriate path weight values. This question could not be answered in general since in particular several viewpoints could be used depending on the context of the test suite to be evaluated:

- quantification of probability, due to the number or frequency of occurrence,
- fault risks, due to the probability of any failure,
- severity, due to the resulting effects if a failure may occur in the selected branch, and
- combination of several models.

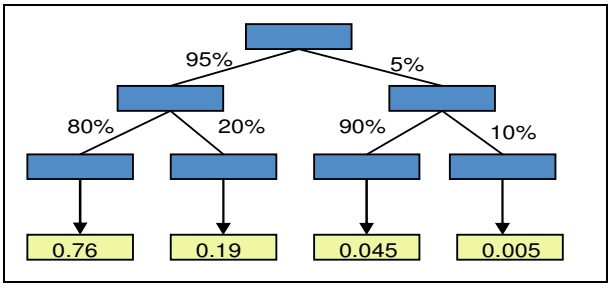

Figure 6: Classification tree with weight calculations

The values may be observed and assessed from previous projects, estimated by some experienced person, or calculated from mathematical models.

5 Adaptation of Development Process

For the release of the main service package the usual test development process can be used. Once the requirements specification is defined, test models and test cases are derived, implemented and executed. To reduce the costs of software based on customizable components, the classical development process needs to be adapted. In order to enable the reuse of test cases from the main release also for the testing the customizations, the approach focuses on: (1) the modeling of the test cases from requirements with the classification tree method and (2) on the reuse of release requirements for the customization (see Figure 7).

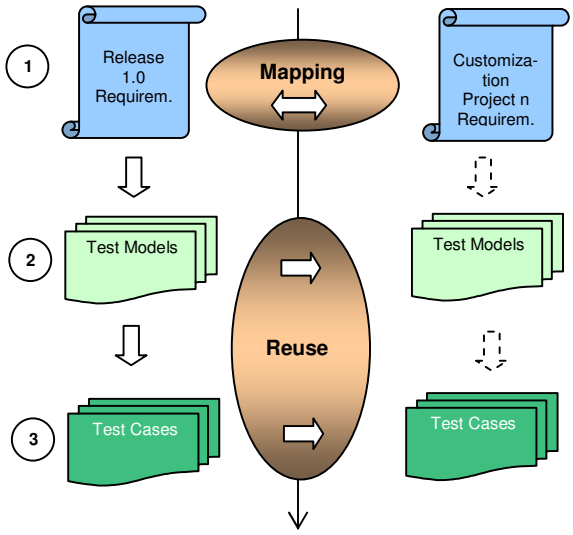

Figure 7: Reuse of tests for projects based on customizable components

At the beginning of the customization process it is important to detect which requirement can be reused, which is a new one and which requirement should be modified. In the next step the test models are adapted and references to the new or modified requirements are included. The last step is the identification of test cases. Using classification tree models allows now for an exact matching of necessary test cases. Thereby it is possible to reduce the costs for testing, decrease time to market and keep the top-quality of software products.

6 Tool Support

For supporting the described test method the CTE XL tool is used [4]. For supporting the reusing of test suits a function for detection of new and modified test cases in the classification tree model has been implemented. As well functions for selection of test cases by their priorities have been introduced.

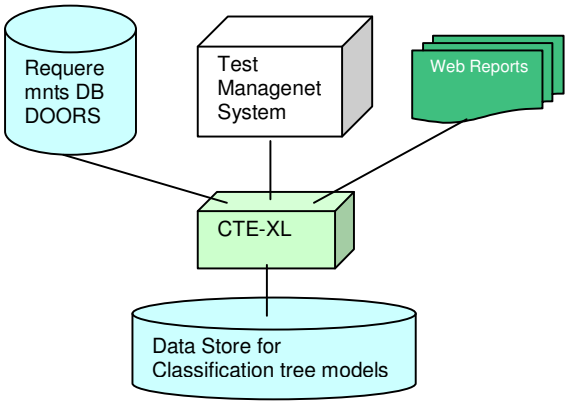

Figure 8: Tool Chain

For better integration of the CTE tool into the existing development, test and management tool environment the interface to the requirements management system DOORS [8] and the interface to the test object management system have been extended. Generation of the web based reports provides users with searching functions, enables them for tracking of requirements assignments and contributes to the better review process. Furthermore the test models are stored in the version control system and can be reused by projects.

7 Summary/Outlook

In this paper, an approach for the new area of testing customizable telecommunication services was presented. The logic of such services can be easily modified by service providers and developers without recompilation of the source code using appropriate tools. Due to the complexity of customizable software it is impossible to test all combinations of available customizations in advance. Thus, the test approach presented is based on use cases and weighting mechanisms for selecting and combining customization mechanisms for the test campaign. As discussed, our test approach has been integrated into the overall software test process. The benefit of this approach is that it makes the reuse of test models and test cases possible and thus helps to shorten the time-to-market for customized services. At the same time the proposed approach allows to reduce the test costs due to the exact match of necessary tests for the customized modules.

References

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

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

[3] Grochtmann, M. Test Case Design using Classification Trees. *In Proceedings of the International Conference on Software Testing Analysis & Review,* Washington D.C., USA, May 1994. STAR 1994, 1994.

[4] 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,* Kopenhagen, Denmark, EuroSTAR 2000, December 2000.

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

[6] 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, October 1993, pages 169 – 176. EuroSTAR 1993, 1993.

[7] S. Alekseev, R. Tiede, and P. Tollkühn. Systematic Approach for using the Classification Tree Method for Testing Complex Software-Systems. *In Proceeding of the IASTED Conference on Software Engineering, February 13-15, 2007*

[8] Telelogic, Introduction to Requirements Management and Telelogic DOOR, Webinar presentation, http://www.telelogic.com/download/index.cfm