Model-Based System Testing of Software Product Families[1]

Andreas Reuys, Erik Kamsties, Klaus Pohl, and Sacha Reis

University of Duisburg–Essen, Software Systems Engineering,
Schuetzenbahn 70, 45117 Essen, Germany
{Reuys, Kamsties, Pohl, Reis}@sse.uni-essen.de

Abstract. In software product family engineering reusable artifacts are produced during domain engineering and applications are built from these artifacts during application engineering. Modeling variability of current and future applications is the key for enabling reuse. The proactive reuse leads to a reduction in development costs and a shorter time to market. Up to now, these benefits have been realized for the constructive development phases, but not for testing. This paper presents the ScenTED technique (Scenario based TEst case Derivation), which aims at reducing effort in product family testing. ScenTED is a model-based, reuse-oriented technique for test case derivation in the system test of software product families. Reuse of test cases is ensured by preserving variability during test case derivation. Thus, concepts known from model-based testing in single system engineering, e.g., coverage metrics, must be adapted. Experiences with our technique gained from an industrial case study are discussed and prototypical tool support is illustrated.

1 Introduction

Software product family engineering (PFE) is an emerging discipline. The goals of PFE are to reduce development costs and Time-to-Market as well as to increase quality of individual applications [4]. An essential concept is proactive reuse [14]. Following this concept, PFE is structured into domain engineering (*development for reuse*) and application engineering (*development with reuse*) [21]. Reusable artifacts are created during domain engineering and are reused during application engineering to create customer-specific applications.

The increased productivity in product family engineering requires a more efficient test approach than those used in single system engineering. Testing consumes up to 50% of the total effort in single system engineering [1]. This percentage increases in PFE, because the effort for constructing applications decreases due to comprehensive reuse. Testing becomes a serious bottleneck of product family development.

[1] This work was partially funded by the CAFÉ-project "From Concept to Application in System Family Engineering" (BMBF, Foerderkennzeichen 01 IS 002 C) and the ITEA Project ip02009 FAMILIES "FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering", Eureka Σ! 2023 Programme.

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 519 – 534, 2005.

We argue that the idea of proactive reuse should be extended to product family testing to accelerate this activity [12]. Following this idea, reusable test cases are created in domain engineering, which are reused for testing an application.

The ScenTED technique (Scenario based TEst case Derivation) adapts model-based testing to product family engineering and supports the proactive reuse of test cases. Model-based testing is an approach to systematically derive test cases in single system engineering. In essence, model-based testing consists of two steps (see Fig. 1). A test model is built from the requirements. It is common to use state charts [17] or activity diagrams [9] as representation for the models. In a second step, test cases are created using coverage criteria or other derivation techniques.

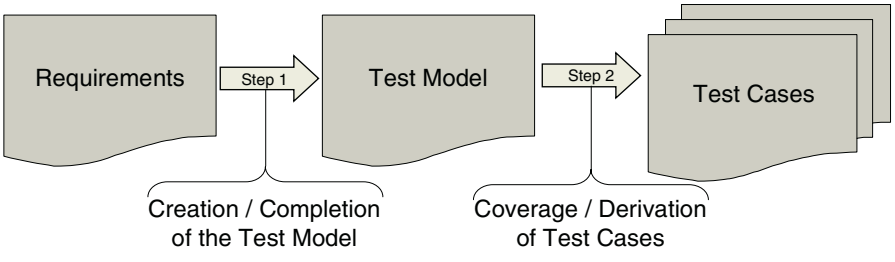


Fig. 1. Model-based Testing in Single System Development (adapted from [6])

Model-based testing offers several advantages, e.g. test cases can be created in a systematic, i.e., repeatable fashion, and stopping rules can be defined [17]. Therefore, model-based testing is considered a prerequisite for automated test generation [3][6]. Another very important aspect is that test engineers validate the requirements by creating the test model. Defects in requirements, such as ambiguities and incompleteness, may be detected during the development of the test model, which is cheaper than correcting them in later development phases. These benefits can be realized for product family engineering by adapting model-based testing.

Variability is the key challenge for adapting model-based testing in product family engineering. The variability of a product family specifies the differences among applications to be build and is defined during domain engineering. Functionality is called a variant whenever it is not planned to be part of all applications. For example *pay per credit card* is a variant in an eShop, because it is not part of all applications. To cope with variability, it is necessary to adapt the test model, its creation process, test case derivation, and representation of test cases. In summary, the goals of our approach are:

- to achieve a reduction in test case development effort compared to the use of single system techniques in product family engineering, and
- to realize the benefits of model-based testing for product families by considering variability in test models and techniques.

In the following, the related work on product family testing is reviewed.

1.1 Related Work

Three approaches support the idea of extending proactive reuse to product family testing. McGregor [13] and Geppert et al. [17] create reusable test cases during domain engineering, but these approaches are not model-based. The test cases are derived from natural language requirements [13] and generalized from existing test cases [7].

Nebut et al. [16] follows also the idea of proactive reuse. They consider scenario fragments in domain engineering that are assembled to test case scenarios. However, there is no test model that guides the assembling of these fragments during domain engineering. Dependencies between use cases are specified in a use case transition graph, but test case scenarios are only derived for specific applications when the variability has already been bound.

Hartmann et al. [10] use an activity diagram as test model, which contains variability, but test cases are derived only in application engineering. Therefore, it is a model-based testing approach, but does not consider the reuse of test cases. Bertolino and Gnesi [2] do not use a test model, but a structured test specification that contains variability. Test cases are created for each application based on this specification.

In summary, the approaches by McGregor, Geppert et al. and Nebut support the idea of extending proactive reuse to product family testing. Hartmann et al. support the idea of model-based testing in product family engineering. However, there is no approach for product family testing up to now, which combines pro-active reuse with the benefits of model-based testing.

1.2 Overview

In this paper the ScenTED technique (Scenario based TEst case Derivation) is presented. ScenTED is a model-based technique for system testing in product family engineering. The key idea of the technique is to create reusable test case scenarios in domain engineering and to reuse these test case scenarios in application engineering.

The adaptation of model-based testing for product family engineering and its realization in ScenTED is explained in Section 2. The ScenTED activities for model-based testing are described in Section 3. ScenTED has been used in an industrial setting where a case study has been performed to measure the reuse benefit. This case study is described in Section 4. Section 5 gives an overview on prototypical tool support for ScenTED. The paper concludes with a summary and outlook on future work.

2 Model-Based System Testing in Product Family Engineering

In this section we present an adaptation of model-based testing. Test models used within the ScenTED technique are defined, whereas the technique itself is explained in the next chapter.

Model-based testing has to be performed in domain engineering as well as in application engineering. It has to be conducted in domain engineering for two reasons. First, the domain test model and the test cases are used to facilitate an early validation of the domain requirements. Second, the test cases are created for reuse in application engineering.

Test models must contain variability and techniques must consider this variability in domain engineering (Fig. 2). Commonalities and variability are specified in domain requirements [8]. As requirements include variability, test cases must also contain explicit variability information. Test cases that include variability are called variant test cases. Test cases without variability are called common test cases and can be applied to all applications.

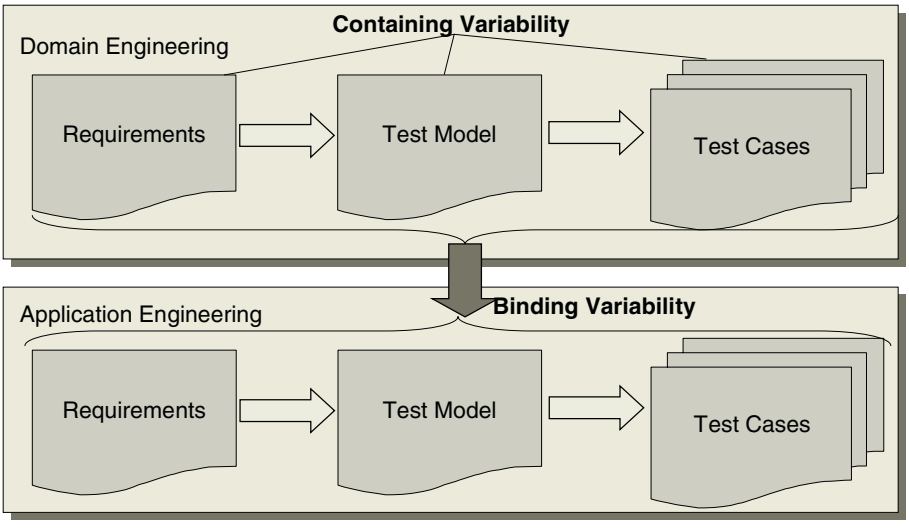


Fig. 2. Model-based Testing in Software Product Family Engineering

The model-based testing approach has to be conducted during application engineering, too. Thereby, no new test models must be created, but the test models that contain variability must be adapted to application specific needs. Application test models are required for two reasons. First, test models and test cases document the test activities. This is necessary when customers or laws and standards demand a proof that requirements have been tested successfully (e.g. U.S. Food and Drug Administration requires a proof of requirement coverage in testing). Second, an application test model is required to identify reusable test cases from domain engineering and to derive application-specific test cases.

In application engineering, the application engineer defines requirements with a customer. The application test model is built based upon the application requirements and by reusing the domain test model. Variability within the domain test model is removed and new requirements from customers may be added to get the application test model. Test cases are identified and derived in the following two steps. First, the reusable test cases from domain engineering are selected. Some of these test cases require an adaptation due to the selected variability in the application. Secondly, new test cases must be derived, if new requirements were incorporated into the test model.

The ScenTED technique is based on the assumption that requirements have been specified as use cases (Fig. 3). Activity diagrams are used as test model from which

test case scenarios are derived. The test case scenarios are specified in sequence diagrams. Test case scenarios describe the test engineer's actions and the responses of the system without specifying concrete test data.

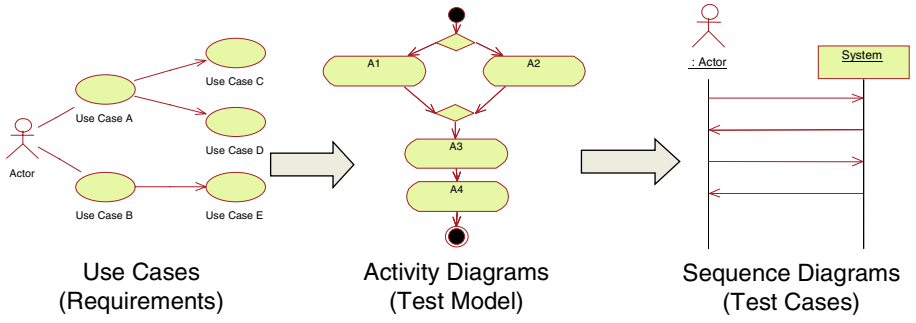


Fig. 3. Models Used within ScenTED

We have chosen use cases as requirements specification, because use cases are well suited to elicit and document customer requirements for information systems in single system engineering as well as in product family engineering (see [8] for a survey on the extension of use cases for PFE). Use cases serve also as good starting point for system testing as they describe the system behavior from an external point of view, which is the focus of the system test. Activity diagrams or state charts are the most common test models to represent possible scenarios of use cases in single system engineering and have already been used for model-based system testing [9]. Sequence diagrams are simple forms of representation to negotiate the test scenarios with requirements engineers and can be incorporated into commercial test tools [11].

Test case scenarios are related to test case specifications. A test case specification refines a test case scenario and comprises detailed test inputs (e.g., *4711* as valid PIN number), expected results (e.g., system response *Enter withdraw amount*), additional information (e.g., to use a touch screen or number block), and test scripts relevant to this scenario. Test case scenarios can be generated automatically, but test case specifications are usually developed manually, which is a time-consuming task. Thus, it is desirable to create both test case scenarios and specifications in domain engineering and to reuse them during application engineering [4][7][13].

We do not describe the creation of the test case specification, as it is out of the scope of this paper. However, it is important to recognize that the reuse of test case scenarios implies the reuse of associated test case specifications and thus reduces the amount of test specifications that have to be created manually.

3 Product Family System Test with ScenTED

In this chapter, the activities of ScenTED for the model-based system testing are explained. During activity 1 (Fig. 4) requirements specified in use cases are modeled as domain activity diagram containing variability (Section 3.1). During activity 2 in

domain engineering test case scenarios are derived using an adapted coverage criterion for product family engineering (Section 3.2). Activity 3 comprises the adaptation of these test case scenarios for a customer-specific application (Section 3.3). Traceability information created during the first two activities is required to perform the third activity (depicted by the dotted arrows).

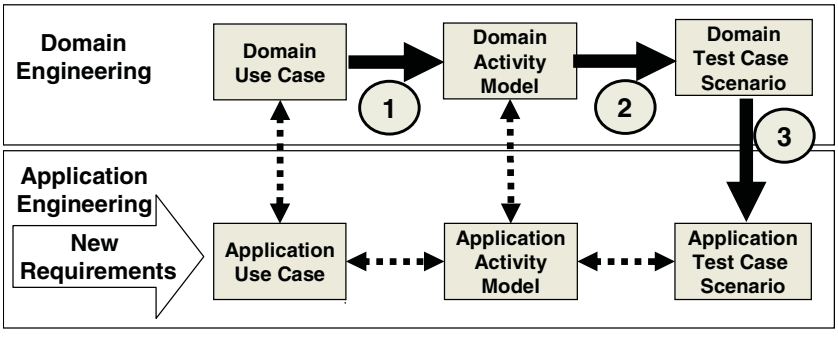


Fig. 4. ScenTED Activities

3.1 Creating the Hierarchical Activity Diagram

Within the first activity of ScenTED (Fig. 4) domain use cases are taken as input and a hierarchical activity diagram is created. In a first step, one activity diagram per use case is created that represents all scenarios specified within the use case. In a second step, the activity diagrams are integrated into an overall activity diagram enabling the derivation of end-to-end scenarios.

A prerequisite to use activity diagrams in domain engineering is the ability to represent variability in the control flow [20]. A variation point is documented as a special decision within the activity diagram. The variants are represented as different control flows, e.g. as sequences of activities at the variation point [19]. Variants may be optional, alternative, or co-existing. *Optional* means that the variants may be chosen additionally for an application ([0..1 out of 1]-dependency). *Alternative* is a [0..1 out of n]-dependency, meaning that one variant out of many variants may be chosen at maximum. *Co-existing* enables the selection of zero to m variants out of n possible variants ([0..m out of n]). A variation point is called *mandatory,* if at least one variant has to be chosen. In that case, an alternative-dependency has a cardinality of [1 out of n] and a co-existing-dependency is [1..m out of n]. Variation points that are not mandatory are also called *optional*.

For system testing it is necessary to create end-to-end scenarios. Therefore, the activity diagrams are arranged in one hierarchical diagram. The starting point to create the hierarchical diagram is the use case model. The top level of the hierarchy is a diagram, which contains the use cases that a user can perform directly. These use cases are modeled as activities, each activity includes another activity diagram. These activity diagrams on the second level model the behavior within the use cases. Included use cases are modeled on the next level and variants on the last level.

The creation of the hierarchical activity diagram is shown for the example of an e-Shop, depicted in Fig. 5. The use case diagram in Fig. 5a) shows that the buyer may *Register* or *Buy goods*. The use cases *Search goods* and *Search catalogue* are not directly initiated by the buyer, but included in use case *Buy goods*. Diagram b) shows the resulting overall activity diagram. The user may *register* or *buy goods* as he enters the eShop. These two activities are the only activities in the activity diagram as these are the only activities directly triggered by the actor.

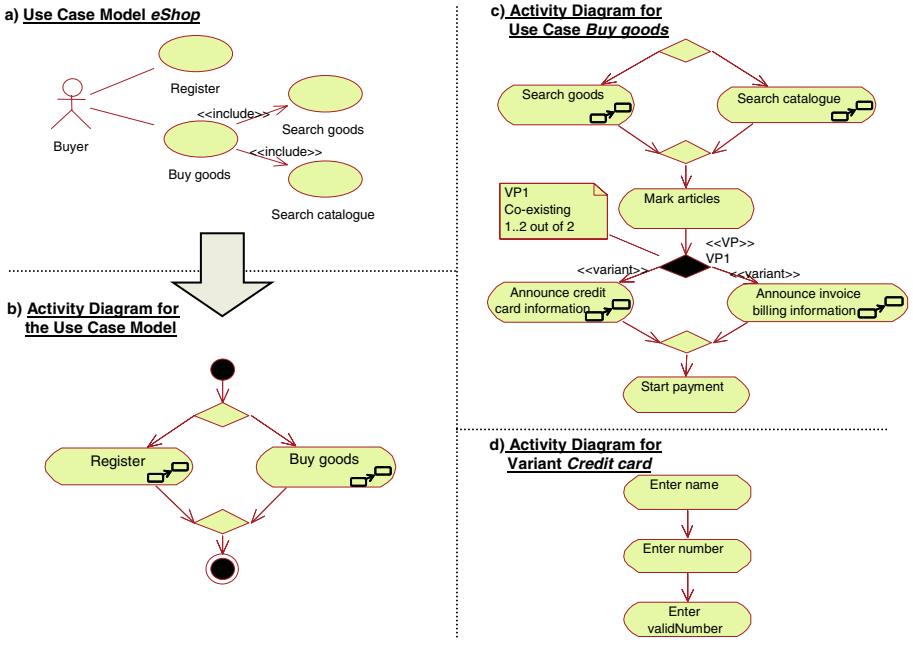


Fig. 5. Activity Diagrams containing Variability

The activity diagram for a single use case is shown in diagram c). This is the diagram for use case *Buy goods*, which is the refinement of the activity *Buy goods* in the activity diagram on the top level. The activity diagram for *Buy goods* represents all possible scenarios specified in the use case. Variability defined within the use cases is represented by the specific decision point (stereotype <<*VP*>>) and outgoing branches that are additionally marked with stereotypes [19]. Customers of future applications must choose if their application should provide the *credit card* variant, the *invoice billing* variant, or both variants as VP1 is specified as co-existing.

The next level of activity diagrams bears the included use cases *Search goods* and *Search catalogue* as well as diagrams for the variants *CreditCard* and *InvoiceBilling*. The activity diagram for variant *Credit Card* is depicted in diagram d).

3.2 Deriving Test Case Scenarios

In the second ScenTED activity, the test model created during domain engineering is used to derive test case scenarios. This is achieved by using a coverage criterion. The adaptation of well known code coverage criteria for model-based testing, esp. activity diagrams is also considered in [3]. Important criteria are the statement coverage criterion, the path coverage criterion, and the branch coverage criterion. The statement coverage criterion is only a weak criterion whereas the path coverage criterion leads to a huge amount or even an infinite number of scenarios in a complex system [15]. The branch coverage criterion is most commonly used in single system development. Therefore, we adapted this criterion for product family engineering within ScenTED.

The original branch coverage criterion is fulfilled when all branches of the activity diagram are covered by at least one scenario. But variability within the activity diagrams may lead to the fact that branch coverage is no more fulfilled during application engineering as a result of bound variants. Consider the following example of two scenarios for use case *Buy goods* (Fig. 5c), which fulfill the branch coverage criterion:

SC_1: {Search goods, Mark articles, Announce credit card information, Start payment}
SC_2: {Search catalogue, Mark articles, Announce billing information, Start payment}

If only variant *Credit Card* is selected for an application, then the second scenario is invalid. So the original branch coverage criterion fails when performed during domain engineering. Therefore, we adapted the branch coverage criterion:
Each branch of the control flow for every possible application has to be covered by at least one scenario.

In order to derive scenarios that fulfill this criterion, the key idea is to abstract temporarily from the variability, to derive the scenarios, and afterwards to detail the variability again. This works for all mandatory variation points, but only if the variability in the flow of events has only local impact. The application of this idea on the example leads to the following temporary scenarios in domain engineering for *Buy goods*.

SCV_{temp1}: {Search goods, Mark articles, { }$_{VP1}$, Start payment}
SCV_{temp2}: {Search catalogue, Mark articles, { }$_{VP1}$, Start payment}

These two temporary test case scenarios cover all branches except for the branches containing variability. Therefore, the variants have to be added now, thereby preserving variability. This leads to the following two domain test case scenarios.
SCV_1: {Search goods, Mark articles, {Announce credit card information, Announce billing information}$_{VP1}$, Start payment}
SCV_2: {Search catalogue, Mark articles, {Announce credit card information, Announce billing information}$_{VP1}$, Start payment}

This approach works for the example, but only because of the mandatory variation points and because the variation point is independent, i.e. the flows of events of both variants continue with the same activity (Start payment). For optional or dependent variation points this approach has to be modified. In a first iteration the variants and variation points are not considered during the coverage criterion. This leads to an

initial set of test scenarios that fulfills the branch coverage, besides the branches with variability. In a second iteration, the branches of the variants are covered and integrated into end-to-end scenarios. The resulting step is to merge the scenarios from the two iterations to the set of domain test case scenarios.

Test case scenarios contain more information than scenario steps. The result of each test case scenario step must be verifiable. Therefore, test data and the expected results of each step must be documented. Furthermore, the preconditions and post-conditions denoted in use case must be considered.

Diagram e) in Fig. 6 shows the domain test case scenario SCV_1 for use case *Buy goods*. Sequence diagrams are used in ScenTED to represent the test case scenarios. Additional information on test data and expected results are documented in notes. This information cannot be derived completely by the model-based approach. The test data is often difficult to generate from the test model. Moreover, the expected result is mostly determined by the test engineer. This information can be described in the domain test case scenario and reused during application engineering to reduce effort.

The variability preserved in the activity diagrams is still included within test case scenarios. The test case scenario contains variation point *VP1* as well as the corresponding variants. The variable region within the diagram is emphasized with a note.

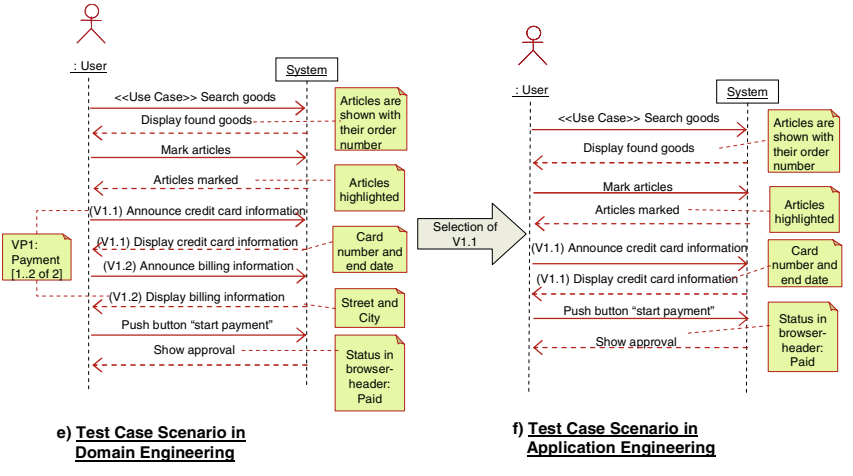


Fig. 6. Binding Variants in Test Case Scenarios

3.3 Deriving Test Case Scenarios for Specific Applications

The third ScenTED activity uses the domain test case scenarios and the application requirements (use cases) to create the set of application test case scenarios. The key idea is to reuse test case scenarios from domain engineering. The variability within the domain test case scenarios must be bound for the derived application and the test scenario is used to document the test case execution.

Customer requirements are elicited at the beginning of application engineering. Thereby, possibilities of the software product family are explained to the customer

and the customer defines the application requirements [8]. The result is documented in the application use case model.

Basically, the test engineer has to perform the same two steps as in domain engineering, but the test models created during domain engineering serve as blueprint. In the first step, the hierarchical activity diagram is used and adapted according to the application requirements. Essentially, variants are chosen for the specific application. Furthermore, it is possible that use cases or use case scenarios may be added or deleted. In the second step, the test engineer determines which test case scenarios derived during domain engineering are still valid for the application, i.e. all common test case scenarios are reviewed to decide if they are applicable or out-dated by changes. Thereby, the following cases may be distinguished:

- Test case scenarios for unchanged commonalities are directly reused.
- Test case scenarios for changed functionality are left out.
- Test case scenarios for variability are bound with the according variants.

Furthermore, additional, application-specific test case scenarios must be created, if there are still uncovered branches within the application activity diagram, esp. for changed functionality. The original branch coverage can be used as the application activity diagram does not contain variability anymore.

Fig. 6 shows the binding of variability for the test case scenario SCV_1. The variant V1.1 *Pay per credit card* (diagram e)) is chosen for the application. The resulting scenario is shown in diagram f), which does exclude any not-selected variant.

The three activities of ScenTED can reduce effort in test case creation for software product families. This is achieved by reusing test case scenarios created during domain engineering. The time spent in the creation of test case specifications (e.g. scripts, data, and results) can be economized if the domain test case scenarios are related to the specifications and are reused in application engineering.

The described technique has been formalized and implemented into a prototype. The prototype will be sketched in Section 5.

4 Case Study: ScenTED at Siemens

ScenTED has been applied within a case study at Siemens AG Medical Solutions HS IM. The goal of the case study was to reduce the effort for test creation by reusing test case scenarios.

Siemens Medical Solutions develops workplaces for radiologists. The radiologists' tasks include the creation of an examination request, the filming of images, and the completion of a patient report. The considered product family supports the tasks of creation and administration of patient records and image data. The recorded data is stored on a central server. The creation of a report and image-post processing take place at a client workstation. Several different clients are developed based on the same documents (requirements, architecture, and code). The clients have different qualitative variants as well as functional variants. The qualitative variants define workstations, which either view images on a normal monitor or on several dedicated, high resolution monitors. The functional variants define applications that are only

able to view images or may copy the images to other embedded applications, e.g. 3D viewing applications.

4.1 Adaptation and Application of ScenTED

This section describes the adaptation of the three ScenTED activities to Siemens-specific needs. Use of commercial tools within the application of ScenTED is shown.

Creation of the Activity Diagram: The developers at Siemens already use a hierarchical activity diagram. The described notation of variability in activity diagrams (see Section 3.1) has been introduced and adapted. Fig. 7 shows such an activity diagram. The notation was slightly changed. The stereotypes specify now the activities that can be used in the specific application (Product A/B/C). A stereotype has been defined for each meaningful combination of applications. The activity diagram therefore represents the domain test model with its possible variants as well as the application test model, and thereby the connection between variant functionality and applications is made explicit. Activities without stereotype are supported by all applications. Thus, these activities represent common functionality.

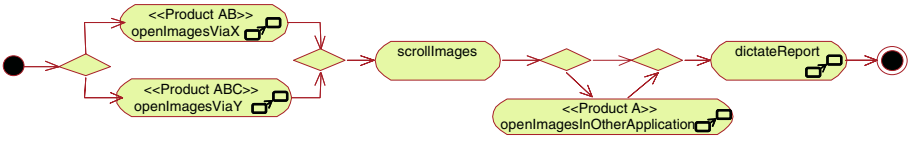


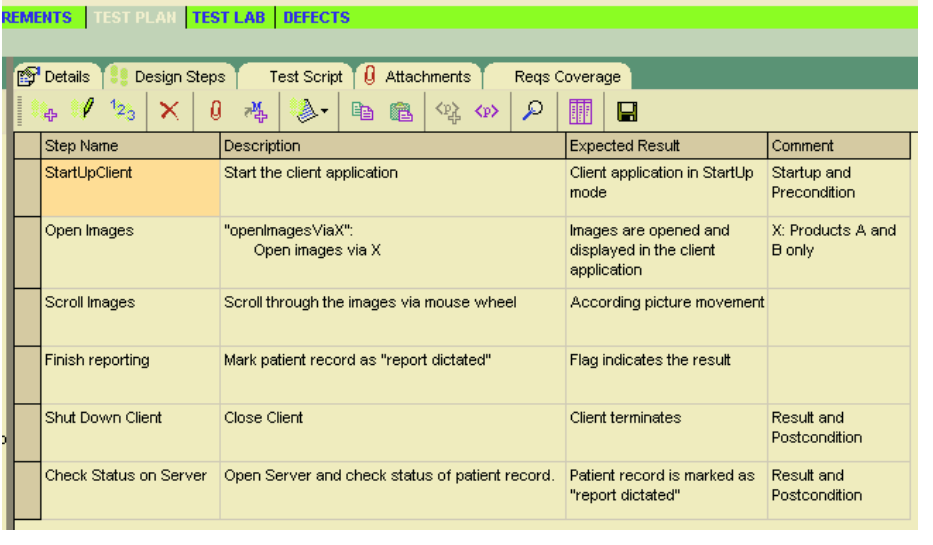
Fig. 7. Adapted Activity Diagram used at Siemens

Derivation of Domain Test Case Scenarios: Test case scenarios are derived from the activity diagram using the adapted coverage criterion. There were a lot of optional variation points, because one application realizes a lot of functionality, but others do not. According to the described approach in Section 0, the branches without variability are considered first, creating the test case scenarios for common functionality. Then test case scenarios are derived that contain variability. The test case scenarios are supplemented with information on application restrictions, i.e. on which application they are usable. This information is taken from the stereotype in the activity diagram. The resulting scenarios are also modeled in IBM Rational Rose. The diagrams include the test data and expected results.

Derivation of Test Case Scenarios for an Application: Test case scenarios for an application are derived from domain test case scenarios. All common test case scenarios are reused and test case scenarios that are possible for the considered application (specified within the domain test case scenarios) are selected additionally. Application test case scenarios are administrated in the test tool Mercury TestDirector (Fig. 8). The steps to ensure the precondition are marked as *Startup and Precondition* in the comment field. The field *Expected Result* describes the result for each step. The steps to validate the overall test case result and post-condition are specifically marked in the comment. Furthermore, the comment contains the specification in which application the test case scenario is applicable.

The benefit from the test tool is that test cases for common behavior and test steps from variability containing test cases may be reused in a copy-and-paste manner with only minor adaptations. This is working for the test case design (see Fig. 8) as well as for the underlying test script.

An idea developed from the Siemens test engineers is to define a library that contains a test script fragment for each activity within the activity diagram. Each fragment can be called with a parameter for test data. This library thus includes test fragments for all activities within the Rational Rose model. New test case scenarios can be created using drag-and-drop whenever the scenario is derived from an existing and already covered activity diagram.



Step Name	Description	Expected Result	Comment
StartUpClient	Start the client application	Client application in StartUp mode	Startup and Precondition
Open Images	"openImagesViaX": Open images via X	Images are opened and displayed in the client application	X: Products A and B only
Scroll Images	Scroll through the images via mouse wheel	According picture movement	
Finish reporting	Mark patient record as "report dictated"	Flag indicates the result	
Shut Down Client	Close Client	Client terminates	Result and Postcondition
Check Status on Server	Open Server and check status of patient record.	Patient record is marked as "report dictated"	Result and Postcondition

Fig. 8. Test Case Scenario in Mercury TestDirector

4.2 Results

ScenTED has been evaluated regarding two aspects: the amount of reused test cases and the test engineer's opinion whether ScenTED supports reuse of test cases. Therefore, the number of reused test case scenarios was determined at the end of the testing phase and a questionnaire was given to the test engineers.

The analysis of the applied test case scenarios led to the following results: 27 end-to-end test case scenarios were created during domain engineering. 63 test case scenarios were derived from these domain scenarios during application engineering. Each of the 27 test case scenarios had to be implemented in the TestDirector first. The 63 test case scenarios were reused from the originating 27 scenarios. Applying single system testing techniques would result in the implementation of 63 independent test case scenarios. Therefore, the reuse-size and frequency measure R_{sf} for reuse within single systems engineering from [5] leads to a value of $R_{sf} = (63 - 27) / 63 = 0.57$. This means that 57% of the needed artifacts are created by reuse. Thus, we have a

benefit of 57% for the considered part of the system compared to single system development.

A questionnaire was used to get an estimation of the test engineers' opinion regarding the support of reuse offered by ScenTED. Statistical analysis has shown that ScenTED really supports the creation and reuse of test case scenarios. Details regarding the analysis of the questionnaire can be found in [18].

The case study stresses that the ScenTED technique reaches the goal of reduction in test creation effort. Due to the derivation and reuse of domain test case scenarios reduced test creation effort was achieved.

5 Tool Support for ScenTED

During the case study, the need for tool support was raised by the test engineers from Siemens. Especially the derivation of test case scenarios from activity diagrams was a time-consuming activity. The coverage had to be considered and navigation in the activity diagram hierarchy required concentration. Moreover, the retrieval of the application test case scenarios is cumbersome. Meanwhile, we have developed prototypical tool support for the ScenTED technique. We focused on the automated derivation of test case scenarios in domain engineering.

We have adapted an existing algorithm for branch coverage. This algorithm has been extended by the different cases specified in Section 3.1, i.e. the mandatory and optional as well as dependent and independent variation points. The formalization of the product family branch coverage criterion and its implementation led to the ScenTED-DTCD tool (Domain Test Case Scenario Derivation) as depicted in Fig. 9.

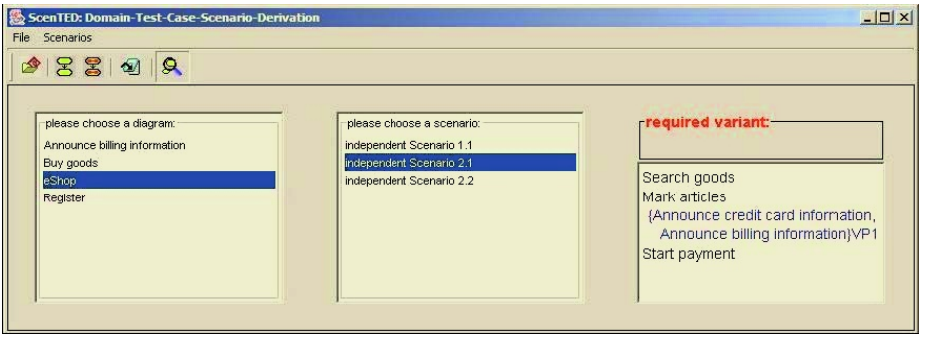


Fig. 9. Tool Support for Derivation of Domain Test Case Scenarios

The input for the tool is the mdl-file created by Rational Rose. The tool is able to deal with variation points and variants as far as they are specified as described in Section 3.1. The outputs of the tool are test case scenarios that describe the test engineer's activities and the system's response if modeled within the activity diagram. These scenarios are shown in the right section of this tool, but may also be saved

within a text file. An extension allowing the export of test case scenarios to a test case administration tool is currently under construction.

The screenshot in Fig. 9 shows the derivation test case scenarios for the eShop example. Three scenarios cover all branches of the example (see Fig. 5). The variability is preserved within the scenarios as shown in the right part of the image. The depicted scenario is one end-to-end scenario. All derived scenarios are independent scenarios due to the given example; therefore the field *required variant* is empty.

The implementation of the tool shows the formalization of the coverage criterion. We have explicitly defined variability in models and considered variability within test case scenario derivation. Model-based testing enables automation and the tool shows that this advantage is still available after the extension to software product family engineering.

6 Summary and Outlook

Testing is a serious problem in product family engineering. The productivity gains that have been realized for the constructive development phases could not be realized for testing until now. Moreover, the variability of the product family complicates testing in domain engineering.

In this paper we have presented the ScenTED technique for the system test of software product families. The contribution of this paper is the adaptation of modelbased testing to product family engineering: Test artifacts in domain engineering are extended to represent variability, in particular activity and sequence diagrams. The activities to derive test cases are enriched by an adapted branch coverage criterion for product family engineering and the binding of variants in test case scenarios.

ScenTED leads to reduction of test case development effort. In a case study within a recent project at Siemens Medical Solutions, the reuse benefit was up to 57% compared to application of single system testing techniques. Due to this success Siemens Medical Solutions HS IM adapted their testing process. ScenTED was put into daily practice and it is currently propagated to other departments at Siemens.

The advantages of model-based testing are realized by our technique, esp. early validation and automated test generation. During the application of ScenTED at Siemens, it was observed that requirements and in particular the variability within the requirements could be validated early due to the creation of the domain test case scenarios. Moreover, we demonstrated using a prototypical tool that the coverage criterion can be formalized and implemented.

Our future work aims at improving the technique and its tool support. We plan to improve ScenTED to save even more effort in product family testing. Effort can be saved in test case execution, if only those test cases are executed that test differences to a previously tested application. Even though there is a prototypical implementation, the technique lacks integrated tool support. The test models can be created in commercial tools like RationalRose or TestDirector, but creating the activity diagrams

as well as reusing test case scenarios for application testing has still to be performed manually. Our current work aims at providing full tool support for ScenTED.

Acknowledgements

We would like to thank the staff at Siemens Medical Solutions HS IM that ontributed to the case study, especially Helmut Goetz, Frank Rometsch, Juergen Neumann, and Harald Lauritsch. Last, but not least we would also like to thank Thomas Rinke for his support in the creation of the prototype.

References

1. Beizer, B.; "Black box testing", Van Nostrand Reinold, New York, 1990.
2. Bertolino, A.; Gnesi, S.; "PLUTO: A Test Methodology for Product Families", 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
3. Binder, R.; "Testing Object-Oriented Systems: Models, Patterns, and Tools", Addison-Wesley, Reading, 2000.
4. Clemens, P.; Northrop, L.; "Software Product Lines: Practices and Patterns", Addison-Wesley, Reading, 2002.
5. Devanbu, P.; Karstu, S.; Melo, W.; Thomas, W.; "Analytical and Empirical Evaluation of Software Reuse Metrics", 18th Intl. Conference on Software Engineering (ICSE), pp. 189-199, July 1995.
6. El-Far, I. K.; "Enjoying the Perks of Model-Based Testing", Software Testing, Analysis, and Review Conference (STARWEST 2001), 2001.
7. Geppert, B.; Li, J.; Roessler, F.; Weiss, D.; "Towards Generating Acceptance Tests for Product Lines", 8th Intl. Conference on Software Reuse 2004, Madrid, Spain, Springer, New York, pp. 35-48, 2004.
8. Halmans, G.; Pohl, K.: "Communicating the Variability of a Software Product Family to Customers", Software and Systems Modeling (SoSyM), Vol. 2, pp. 15-36, Springer, Hamburg, March 2003.
9. Hartmann, J.; Vieira, M.; Foster, H.; Ruder, A.; "TDE/UML: A UML-based Test Generator to Support System Testing"; 5th Annual International Software Testing Conference in India, 2005.
10. Hartmann, J.; Vieira, M.; Ruder, A.; "UML-based Approach for Validating Product Lines", Intl. Workshop on Software Product Line Testing (SPLiT), Avaya Labs Technical Report, pp. 58-64, Boston, USA, August 2004.
11. Hauber, R.; Ziegler, M.; Erskine, M.; Hilsenbeck, R.; "Modellbasiertes Testen", Objektspectrum, No 3, pp. 20 – 24, 2003 (in German).
12. Kamsties, E.; Pohl, K.; Reis, S.; Reuys, A.; "Testing Variabilities in Use Case Models", 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
13. McGregor, J.: "Testing a Software Product Line", Technical Report CMU/SEI-2001-TR-022, December 2001.
14. McGregor, J.; Northrop, L.; Jarrad, S.; Pohl, K.; "Initiating Software Product Lines", IEEE Software, Vol. 19, No. 4, pp. 24-27, July/August 2002.
15. Myers G.; "The Art of Software Testing", Wiley, New York, 1979.

16. Nebut, C.; Fleurey, F.; Le Traon, Y.; Jézéquel, J.-M.; “A Requirement-based Approach to Test Product Families”, 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
17. Offutt, J.; Abdurazik, A.; “Generating Tests from UML Specifications”, 2nd Intl. Conference on UML’99, 1999.
18. Reuys, A.; Goetz, H.; Neumann, J.; Weingaertner, J.; “Medizintechnik bei Siemens AG Medical Solutions HS IM”, In: Boeckle, G.; Knauber, P.; Pohl, K.; Schmid, K. (eds); “Software-Produktlinien: Methoden, Einführung und Praxis”, pp. 247-259, dpunkt, Heidelberg, 2004 (in German).
19. Reuys, A.; Reis, S.; Kamsties, E.; Pohl, K.; “Derivation of Domain Test Scenarios from Activity Diagrams“; Intl. Workshop on Product Line Engineering The Early Steps: Planning, Modeling, and Managing (PLEES'03), Erfurt, Germany, September 2003.
20. Riebisch, M.; Boellert, K.; Streidtferdt, D.; Franczyk, B.; “Extending the UML to Model System Families”, World Conference on Integrated Design and Process Technology (IDPT 2000), Dallas, USA, June 2000.
21. van der Linden, F.; “Software Product Families in Europe: The Esaps & Café Projects”, IEEE Software, Vol. 19, No. 4, pp. 41-49, July/August 2002.