Test Case Generation from workflow-based Requirement Specifications

Hartmut Lackner[1], Jaroslav Svacina[2] and Holger Schlingloff[1]

[1] Humboldt Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, D-10099 Berlin, Germany,
{lackner|hs}@informatik.hu-berlin.de
[2] Fraunhofer FIRST, Kekuléstraße 7, D-12489 Berlin, Germany,
jaroslav.svacina@first.fraunhofer.de

Abstract. Model-based testing is a state of the art testing technique for embedded systems engineering. Documenting requirements in a formal way with test models, improves fault detection rate at the early stages of the software life-cycle. Model-based testing automates test case design and test documentation. But designing models for testing is a time consuming and cost-intensive task. In this paper, we will show how to automate testing, using model-based techniques and workflow-based requirement documentations. Therefore we introduce a domain specific language for describing workflow-based requirements. The language is applied onto a web application example, which represents the case study. The introduced language fits into a larger context of a framework for model-based testing.

1 Motivation

Testing a software product is a cumbersome task, but a necessary burden to assure customer satisfaction and product safety. In these days model-based testing is becoming a common practice for automating test design and test documentation. With model-based testing it is possible to generate large test suites, which provide evidence for an accurate system implementation.In many cases, model-based testing is more efficient than any other known testing technique [8]. Though there are cases where model-based testing is more expensive by means of costs, it adds further value through traceability matrices [7] and defined test coverage.

In the fields of safety critical applications, model-based testing supports certification processes. Usually, saftey critical functions are fulfilled by embedded devices/systems, which are special-purpose computer systems. In contrast to general-purpose computer systems, which purpose is not fixed, embedded systems are designed for a special task. The computational power of embedded systems fits exactly the power needed for accomplishing the task. In general, embedded systems are cheap, small and available in large amounts.

1.1 Case study partner

Since embedded systems are defined as special-purpose computer systems, also servers, running web applications, may be considered as embedded systems. Therefore we decided to cooperate with ImmobilienScout24 (IS24), the leading online estate agency in Germany. We examined their test process and then identified potential for improvement by applying model-based methods.

At the state we began our examination, IS24 was already using structured, but still informal models to document the requirements of their web application. The requirement documentation is structured in several tables. For each customer kind a table is maintained. Each table contains the workflow definitions, which are associated with the particular customer kind. The workflows are clearly structured, but lacking formality. Also, designing the model follows no explicit rules – the model is understood and extended intuitively by the designers.

Currently the model is manually mapped into a keyword-driven test framework. Within the test framework test cases are crafted manually. Furthermore, automated validation for proving the consistency of a test case in respect to the control flow and input data is not possible, even though the tools support automatic validation. This is due to the fact that conditions are documented informal.

Special about the requirements model is that it contains both, structure and behavior definitions within the same model. One can argue about the sense of this approach, especially since the common modeling paradigm is to seperate structure from behaviour into different models. As IS24 was successful with this approach in the past and the designer are used to this paradigm, we chose to adapt this paradigm in our language definition. Also, we are aiming for a high user acceptance by evolution, not revolution.

1.2 Related Work

A comprehensive introduction to model-based testing is presented by Utting et al. in [8]. A more general introduction to testing and common terms can be found in [6] and [9].

Since UML is widely spread for modeling software structure and behaviour, many approaches rely on Class-, Activity-, Statemachine-diagrams and/or Message Sequence Charts [3,11,5,1,2]. Further modeling techniques used for model-based testing are B- and Z-Notation [10]. Even Microsoft is offering recently tools for model-based testing from state-based models (Spec Explorer) and providing corresponding literature [4].

1.3 Structure

The purpose of this paper is to take maximum advantage from model-based testing, by introducing a formal workflow language for describing requirements and automatically derive test cases from the created workflow model. The workflow language definition is derived and defined in section 2. Based on the workflow

models defined by the herein developed language, a test framework is created. The test framework derives test cases from the workflow model and an additionally given initial state in a fully automatic manner. In section 3 the test framework is presented. The paper closes with a discussion of the limitations and a general summary of the presented approach.

2 Workflow-centric documentation of requirements

In general, workflows are characterised by tasks, which are assigned to resources. Tasks are composed to (sub-) processes, which – again – may be composed to (sub-) processes. Resources are necessary and/or responsible for the execution of assigned tasks. The execution order of the tasks is directed by control flow elements: sequence, choice (loop resp.) and concurrency. Every Choice is constrained by a condition, which evaluates to a boolean value.

Except for concurrency, every workflow concept is found in the informal web application description of IS24. This is due to the fact, that a user is only able to process one workflow at a time, but implicitly all users are working in parallel on the web application. Tabbed browsing or browsing on multiple instances is not considered in this paper.

In the following sections the workflow concepts of the case study's model are identified and a domain specific language (DSL) is derived. Therefore a fictional example is given in the next section.

2.1 Case study example

The original model is altered to obfuscate business secrets, but still the presented model 2 relies on the same domain and business. The shown model depicts workflows associated to customers who are searching for an estate and those customers who offer an estate.

A	B	C	D	E
	Menus			
		GE Menu Rent		
			T Renting	S Renting
			T Offer	S Offer
	T Rent			
		S Renting		
3			L Search	S Renting
1			case: User has searched	L Last Search
			L Offer	S Offer
		S Offer		
2			L New Offer	

Table 1. Real estate agency web application model – customer's table

The first column (A) is a number which is related to the item(s) in the respective row. It denotes the item's importance within the web application, resp. all workflows. Each item may be a task or a (sub-) process. The first character of a item denotes its type. The type of an item must be one of the following:

- **L**ink,
- **T**ab,
- **G**roup of **E**lements,
- **I**con
- **B**utton
- **F**orm
- **S**elect **B**ox
- **C**heck **B**ox
- **R**adio **B**ox
- **S**ite
- **P**op up
- empty

Apparently there is no type denoting an edit field, which is a common element on forms. Larger forms or forms with edit fields are kept in additional tables. For every larger form exists a separate table (no example will be given here).

The example depicts a web application with a tab-menu (GE Menu) and a tab-panel (T Rent). The tab-menu is included in a container (Menus) and consists of two tab-menu entries (T Renting and T Offer) which refer to the respective site (S Renting or S Offer) via a hyperlink (underlined). The tab-panel consists of two tabs (S Renting and S Offer), while the second one consists of a single Link (S New Offer) which specifies no target at all, the first site (S Renting) is more complex. It features three links, but one of them (L Last Search) is constrained by a choice, indicated by the keyword 'case:'. Its purpose is to obtain the last search results in the case a user has already searched, did something else and then again wants to see the search results. For simplification the Link 'L Search' points directly back to its host site (S Renting) and confirms that a search has been performed (effect). In reality there should be a site with a form to specify search properties and finally a site showing the results of the search.

Hyperlinks are written in blue color and/or are underlined. If the target is defined, a hyperlink within the cell may point to the target cell to identify the target unambiguously. Hyperlinking is a feature of the spreadsheet application, though links may be broken.

Hence hyperlinks are referring to some element within the model, hyperlinking is not used for specifing a target for a link only. As well, hyperlinking is used to import elements, which are defined within another or the same element and making them accessible at runtime. This makes it difficult to clearly distinguish, if an element is only 'defined' at one's place or additionaly 'imported'.

The specification of choices is rather simple. If an item's name starts with 'case:', it is a choice. Choices may be used at the right-hand side of a links,

buttons or other elements with a target to control the elements destination. The constrained destination is written to the right-hand side of the choice. If there are several choices for a target, they are written one below the other.

Constrained imports are handled the same way. While the choice is written in the original place of the import, the import is written to its right-hand side column.

There are more features in the original documents, which will not be discussed in this paper, because they are used very seldom. Also, they complicate the design of application models.

2.2 Deriving a workflow language

On the design concepts described above a workflow language may be applied. As there is strong tool support, we chose to design a DSL in eMOF. In this case it seems feasible to use a DSL, instead of using a standard language. Use Cases are too high level for describing those kind of workflows. Workflow nets, which are based on petri nets, are too low level for documenting this kind of requirements.

There are far more languages for describing workflows, but a DSL gives us the power to design it in the exact way, we want to use it. Therefore, designing models with a DSL should be much simpler for the designers of IS24, than using a standard language. Generalising the DSL for other domains should be possible, but is not considered in this paper.

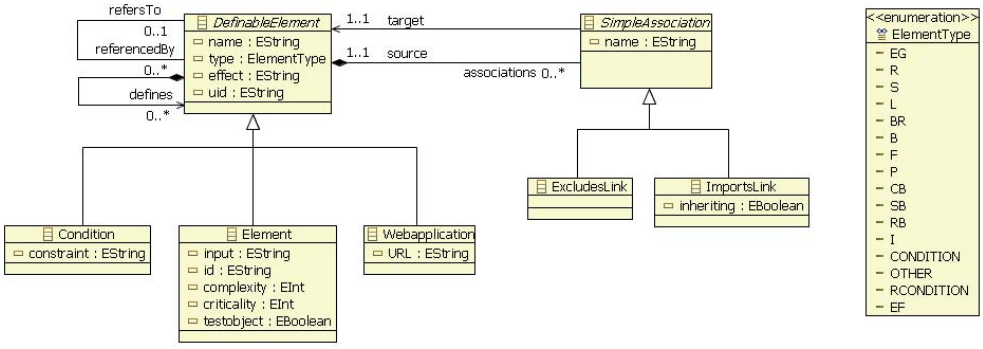

Fig. 1. DSL for workflow-based requirements

The workflow language definition is given in figure 1. The mapping is as follows:

Tasks and (sub-)processes are distinguished by their type. The type is given in the enumeration in section 2.1. The types are reflected within the DSL with the enumeration type 'ElementType'. There are some special types defined like 'CONDITION' and 'RCONDITION'. Those indicate the special handling of

Workflow concept	Case study	DSL
Task	normal cell items	Element (typed)
(Sub-)process	right-hand side to a cell item	Element(typed)
Sequence	hyperlink after Link	Attribute refersTo
Choice	case or simultanuous imports	Condition or import association
Parallelism	inherent	inherent

Table 2. Mapping of workflow concepts onto the example model and the DSL

Conditions resp. Tabs. While depciting conditions with an additional type is quite redundant, the differentiation of tabs comes in quite handy, as we will see later in this paper. Figure 2 shows the correlating model to the above example.

3 Applying model-based testing

State of the art testing is by now model-driven. Figure 3 shows a typical process for model-based testing. From the informal requirements a formal test model is derived. In theory, this should be the only manual task within a model-based test process until test result's interpretion. Between the test model design and test result interpretation are tasks, which in practice, need some manual input. Manual steps within this automated process are concerned with process configuration and optimisation as well as test case deployment.

3.1 Developing a test process

Before the test generator may start, one or more initial states of the model have to be provided. Therefore a XML Schema is defined in figure 4. Every identifier in the model has to be initialised within this XML document. Additionally all special variables for storing states of tabs are contained in this file.

The resulting test case generation process is shown in figure 5.

3.2 Simulating the test model and selecting test cases

From this point on, where a model and an initial assignment of the variable is given, the automated test process needs no further input. The model is then simulated under certain constraints (configuration). The result of the simulation is an exploration graph (fig. 6), which depicts all explored states.

For describing the exploration graph a DSL is developed which is shown in figure 7. States are composed of a runtime structure of the web application and assigned variables. Transitions have a source state, a target state and labels. The labels specify under which cirmcumstances (guard) a task (trigger) may be executed and how the assignment of variables has changed.

For selecting test cases from the exploration graph exist several criteria. In the prototype implementation of the created test framework, we chose transition coverage. To achieve transition coverage we applied the chinese postman tour

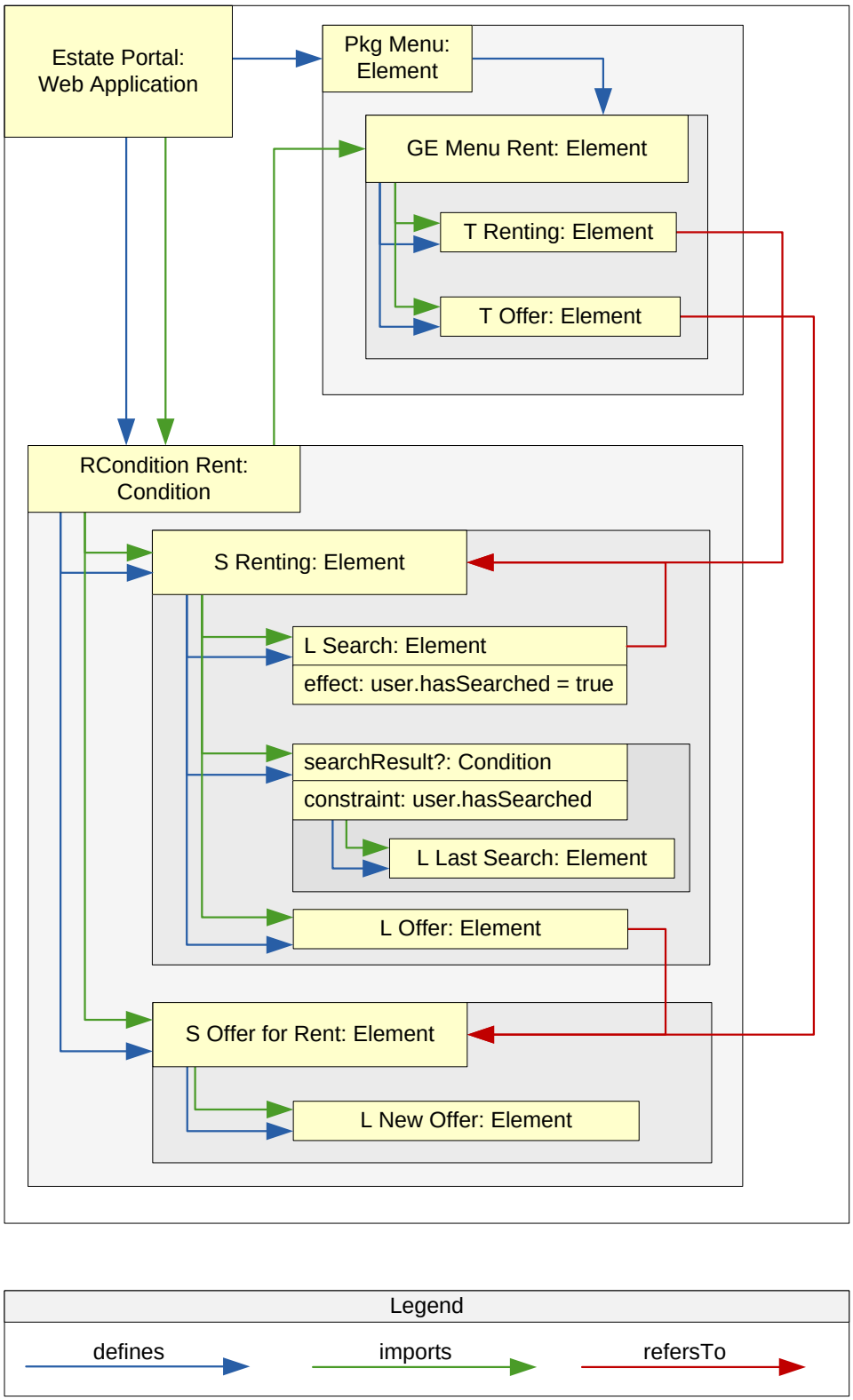

Fig. 2. Workflow-model of the case study's model

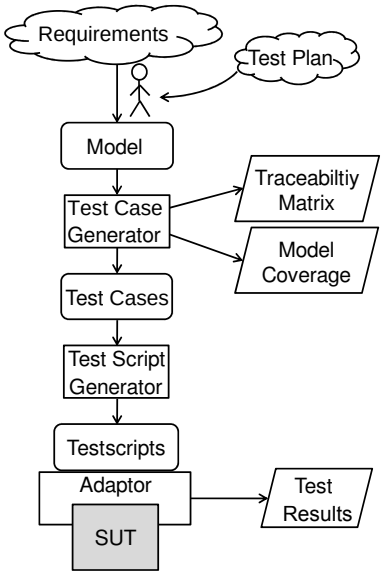

Fig. 3. Process of model-based testing

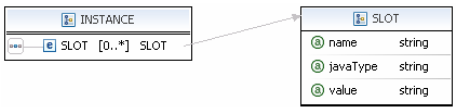

Fig. 4. XML schema definition for the initial variable assignment

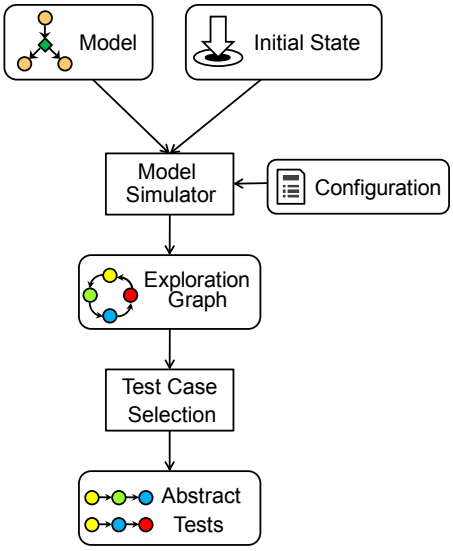

Fig. 5. The test cases generation process in detail

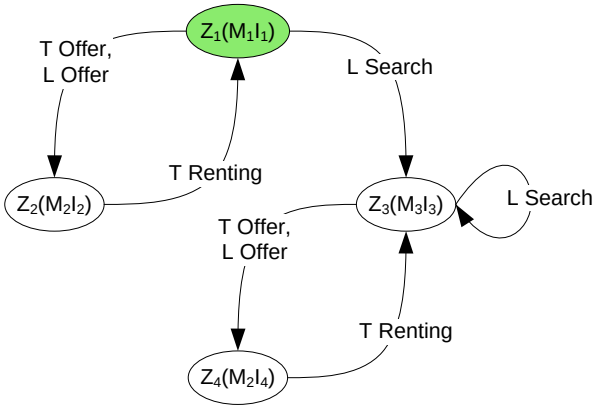

Fig. 6. Exploration graph of the example (without constraints in the simulation's configuration

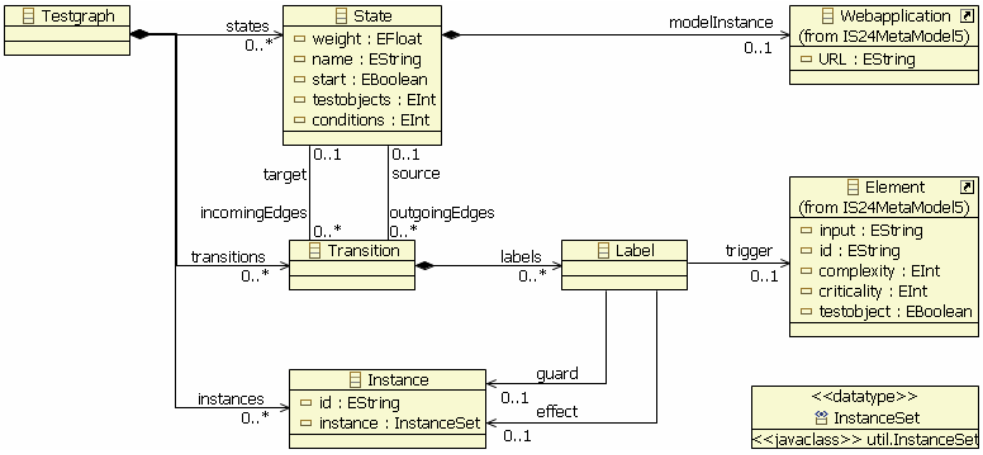

Fig. 7. Exploration graph description language in eMOF

problem (CPP) on the exploration graph. CPP searches for the shortest route covering all transitions on directed, weighted graphs. Furthermore a configuration may filter testcases by a score, which is determined by the length of the test case and the score of all elements within every state of the test case.

The resulting test cases are described with a test case description language, therefore another DSL is defined (fig. 8). The language defines concepts for test suites, test cases, actions and assertions. Again, the DSL is defined in eMOF, to provide maximum interoperability with the the simulation process and the test script generator.

3.3 Generating test scripts

Test script generation is done by a simple code generator. The generated test scripts are directly excutable by the Selenium test framework. Selenium is build for designing test case design and test execution. Since our process has designed the test cases autmatically from the test model, Selenium acts as an adapter to the SUT.

4 Discussion

Since there has not been gathered enough practical experience with this approach and especially the defined language, it is difficult to claim results. IS24 is currently applying the language and encouraged us to carry on with our work. Also the examples are now very limited, yet.

Obviously the introduced workflow language lacks of semantic strength. The ExcludesLink for example is also applicable when no import has made before,

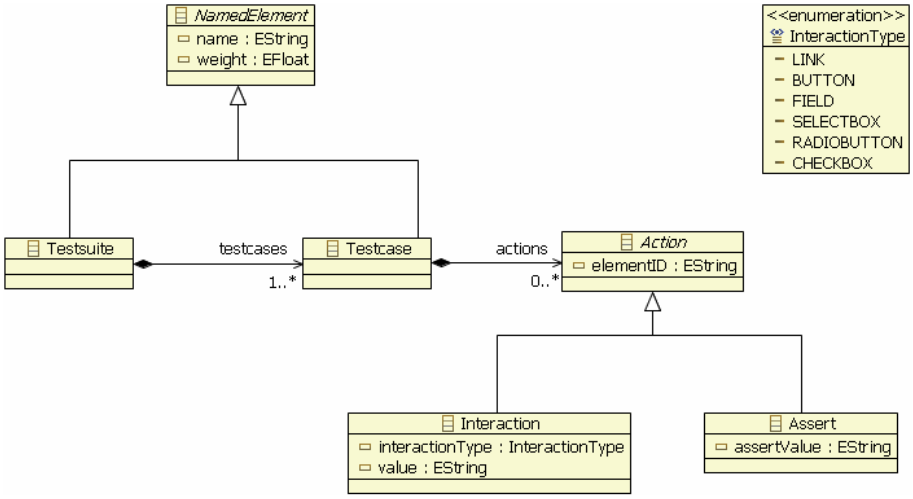

Fig. 8. Test case description language

which makes no sense. Applying OCL-constraints may solve this kind of problems in the easiest way. Also easy to handle with OCL-constraints is the fact that a Condition must not have other types than CONDITTION or RCONDITION. Assigning any other type like Link or Button to a Condition is currently of no practical use. More difficult to express in OCL is to forbid cyclic ImportsLinks.

Furthermore, the language allows the use of non-determinism as it is not explicitly forbidden by means of the metamodel. For reproducibility reasons only deterministic choices should be used.

Since a Webapplication is a DefinableElement it may define and import further Webapplication elements which is at the current state of this work inconclusive. Perhaps this feature will be useful in the distant future, when we are not talking about web applications anymore, but about services.

5 Summary

In this work a workflow description language for documenting requirements has been developed and applied onto smaller examples. We automatically simulated models based on the presented workflow description language and selected abstract test cases. From the abstract test cases, executable test scripts were generated and executed against the SUT.

Still, there is a lot of work to do in the fields of

- enhancing the workflow language, to avoid ambiguities,
- enhancing the simulator to consider all workflow language concepts,
- ascertain test suite quality,
- apply stronger test selection criteria,

We hope to accomplish these targets in the near future and prove them with real world examples.

References

[1] Briones, Laura B.: *Theories for model-based testing: real-time and coverage.* Enschede, University of Twente, Diss., March 2007. – CTIT number: 07-97

[2] Friedman, G. ; Hartman, A. ; Nagin, K. ; Shiran, T.: Projected state machine coverage for software testing. In: *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis.* New York, NY, USA : ACM, 2002. – ISBN 1–58113–562–9, S. 134–143

[3] Friske, Mario ; Schlingloff, Holger: Generierung von UML-Modellen aus formalisierten Anwendungsfallbeschreibungen. In: Conrad, M. (Hrsg.) ; Giese, H. (Hrsg.) ; Rumpe, B. (Hrsg.) ; Schätz, B. (Hrsg.) ; TU Braunschweig (Veranst.): *Tagungsband Dagstuhl-Workshop MBEES: Model Based Engineering of Embedded Systems III* TU Braunschweig, 2007 (Informatik-Bericht 2005-01)

[4] Jacky, Jonathan ; Veanes, Margus ; Campbell, Colin ; Schulte, Wolfram: *Model-Based Software Testing and Analysis with C#.* Cambridge : Cambridge University Press, 2008. – ISBN 0521687616

[5] Kösters, G. ; Six, Hans-Werner ; Winter, M.: Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications. In: *Requirements engineering* 6 (2001), Nr. 1, S. 3–17

[6] Linz, Andreas: *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard.* Dpunkt Verlag, 2005. – ISBN 3898643581

[7] Regan: *A Practical Approach to Software Quality.* Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2002. – ISBN 0387953213

[8] Utting, Mark ; Legeard, Bruno: *Practical Model-Based Testing: A Tools Approach.* 1. Morgan Kaufmann, 2006. – ISBN 0123725011

[9] Vigenschow, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis.* Dpunkt.Verlag GmbH, 2004. – ISBN 3898643050

[10] Vilkomir, Sergiy A. ; Bowen, Jonathan P.: Formalization of Software Testing Criteria using the Z Notation. In: *COMPSAC '01: Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development.* Washington, DC, USA : IEEE Computer Society, 2001. – ISBN 0–7695–1372–7, S. 351–356

[11] Weissleder, Stephan ; Schlingloff, Holger: Deriving Input Partitions from UML Models for Automatic Test Generation. In: Giese, Holger (Hrsg.): *Tagungsband Dagstuhl-Workshop MBEES: Model Based Engineering of Embedded Systems III*, 2007 (Informatik-Bericht)