The Testing Process - A Decision Based Approach

Lars Borner
University of Heidelberg
Im Neuenheimer Feld
Heidelberg, Germany
borner@informatik.uni-heidelberg.de

Timea Illes-Seifert
University of Heidelberg
Im Neuenheimer Feld
Heidelberg, Germany
illes@informatik.uni-heidelberg.de

Barbara Paech
University of Heidelberg
Im Neuenheimer Feld
Heidelberg,Germany
paech@informatik.uni-heidelberg.de

***Abstract*-Software processes often focus on artifacts, activities and roles, treating decisions to be made during the software development process only implicitly. However, awareness of these decisions increases their quality by forcing the decision-makers to search for alternatives and to trade off between them. In this paper, we propose a decision hierarchy for the testing process. This hierarchy comprises all decisions made during testing and reflects dependencies between them. Additionally, we present the results of four case studies to which we applied this decision hierarchy.**

Keywords- decision; testing process; system testing; integration testing; knowledge management

I. INTRODUCTION

Today's software systems consist of numerous software components; they realize countless requirements and are developed in an industrial environment limited by high time and resource constraints. In order to assess to which extent a software system or its parts fulfill the requirements, testing activities have to be performed. Since complete testing is impossible [17], testers are forced to make decisions, i.e. to decide which parts of the software system have to be tested in which way. Usually, these decisions are made implicitly by the corresponding roles and often, the responsible persons are not aware of the decisions they made. However, the awareness of decisions, can significantly improve their quality. Making a decision consciously forces the person who has to take this decision to search for alternatives, to establish selection criteria and to trade off between advantages and disadvantages of several alternatives. Consequently, the awareness of decisions leads to better decisions compared to implicit or ad hoc decisions and increases the quality of the testing process.

In this paper, we define a decision as follows: *A decision denotes a choice consciously or unconsciously made by a person or group of persons. A decision made consciously evolves in the process of discussing possible alternatives and considering existing success criteria.* During the software development process as well as during the testing process, several decisions have to be made. The best alternative has to be selected from e.g. alternative GUI designs, architectural patterns or testing techniques.

The remainder of this paper is organized as follows. Section 2 describes the generic decision hierarchy for the testing process, containing decision levels and corresponding decisions. Section 3 presents results of four case studies, to which we applied this decision hierarchy. Section 4 gives an overview of related work and section 5 gives a short summary and discusses the results of our approach.

II. DECISION HIERARCHY

In our research work we identified the decisions to be made during the testing process and assigned them to decision levels. At first, we identified the tasks and roles by analyzing test process descriptions mentioned in standard textbooks such as Spillner [23] and Mosley and Posey [18]. Our work is mainly based on the fundamental test process described in [23] consisting of test planning and specification, test execution, capturing and analysing test results. In a next step, we identified decisions to be made while performing testing tasks and grouped them into seven decision levels. The result is the generic decision hierarchy illustrated in Figure 1. Management decisions and issues, e.g. scheduling or training are not addressed here, as this would reach beyond the scope of this paper.

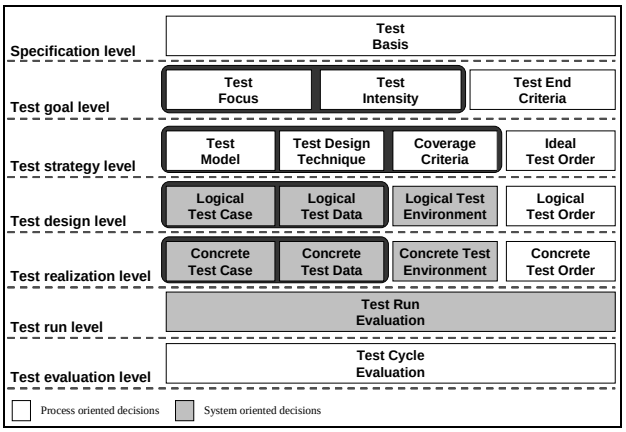

Figure 1. Decision levels and corresponding decisions of the testing process

The *principles* underlying our decision hierarchy can be defined by the following rules:

R1 Decision dependencies: Decisions at lower levels depend on decisions made on earlier levels. If decisions at top levels are left out, they are implicitly contained in decisions on lower levels. Leaving out a decision decreases the quality of this particular decision as well as the quality of all dependent ones. The goal of making decisions in the proposed order is to facilitate the decision making process.

R2 Parallelism: All decisions on the same level can be done in parallel, i.e. these decisions can be made nearly independently, but they may influence each other. Decisions

that influence each other can be combined to *decision bundles*. In Figure 1. decision bundles are represented by a dark grey box behind the corresponding decision (e.g. test focus and test intensity belong to one and the same bundle).

Moreover, two different *perspectives* on the decisions can be identified. One perspective contains decisions which influence the testing process (called process oriented decisions), i.e. which test artifacts will be created. Another perspective contains decisions concerning the system under test (called system oriented decisions), i.e. how the system will be tested. The top level decisions try to give answers to the question *which parts* of the software system have to be tested. The subsequent levels make decisions on *how* the (parts of the) system should be tested. On the last two levels, decisions concerning the *evaluation* of the test runs have to be made.

In the following, we introduce the different decision levels and detail the corresponding decisions on each level.

A. Specification Level

The specification level contains decisions which deal with the completeness of the **test basis**. The test basis includes all information needed for a successful start of the testing process and often consists of the specification of the software system at different development stages (e.g. requirements specification or system design specification). The test basis defines a set of test objects, their behavior, their input and their output as well as the specification of possible dependencies between the test objects. We presume the definition in [12] of a software system, including its specification as well as its implementation (represented by code) and define a test object as a part of a software system. At specification level it has to be decided whether the test basis is complete or not. If information in the test basis is missing, it has to be added. If important information is missing in the test basis, critical parts of the software can be overlooked and thus remain untested. The decisions on this level influence nearly all decisions on the lower levels.

B. Test Goal Level

Considering that a software project usually is limited in time, not all parts of the test basis can be tested. Therefore, at test goal level it has to be decided which parts of the system have to be tested and which not. For this purpose, it is essential to possess a complete test basis in order to select the critical test objects. We denote all parts of the system which have been selected to be tested as **test foci**. Usually, the test foci represent all critical parts of the test basis. Critical in this context means e.g. that the corresponding parts of the software will often be used during run time, they will cause high damage (to the user, to the software system or to the environment) if they fail, they are very complex so that the probability to fail is high or they contain already known defects.

Besides time pressure within the testing process another constraint influences the decisions on this decision level: cost. The cost constraints lead to a limitation of resources needed in the testing process. Therefore, the existing resources have to be split up among several test foci. To concede the correct assignment of resources to the various test foci, it has to be decided which **test intensity (**measured e.g. by man days or funds) a single test focus has to be assigned to.

The test intensity and test foci influence each other and consequently belong to a bundle. Decisions on the **test end criteria** can be made independently from this bundle. The test end criteria define conditions which have to be fulfilled to finish the testing activities, e.g. they can give information on the required rate of successful test runs.

C. Test Strategy Level

This level contains decisions concerning the test strategy to be used. The test strategy comprises decisions related to the test design techniques to be used, the test model(s) and its coverage(s) as well as the ideal test order. One decision to be made concerns the **test design technique** which will be used to derive test cases and test data from the test basis. For each test level (system, integration and unit test level) a countless number of test design techniques can be found in the literature (e.g. in [3], [4], [17], [23]). Therefore, existing test design techniques, the defined test foci and test intensities have to be taken into account in order to select the most adequate test design technique(s). In parallel, decisions on the **test model** have to be made. A test model facilitates the derivation of test cases and test data in comparison to the derivation from an informal specification. A state based model or a control flow model are examples of test models. A test design technique influences the selection of the test model and vice versa. Later in the testing process, the selected test design techniques have to be applied in order to derive test cases and test data to achieve the given test coverage and to fulfill the test **coverage criteria**. The test coverage is an indicator for the number of test cases to be derived. The test design technique influences the decision on coverage criteria and vice versa.

Furthermore, on this decision level an **ideal test order** to test the different test objects has to be specified. The ideal test order represents an optimal order to test the different parts of the system by taking into account the information on the test foci and on test intensity. An example of such an ideal test order could be that all test objects with the highest test intensity should be tested first, followed by the ones with the next lowest intensity and so on.

D. Test Design Level

The test design level is the most important and most complex level of the testing process. The main decision on this level is how to test the different test foci, i.e. the selected test objects. Therefore the given test design techniques are applied to derive **logical test cases** (also called abstract test cases) [13], [23]. A logical test case gives an abstract description of how to test a specific aspect of the objects under test. In parallel to the test case design, it has to be decided which **logical test data** serve as an input for the test objects within the test case. The logical test data represent the abstract description of the data to be sent to and returned by the test object. Both, the specification of a logical test case and the required test data, are connected. A logical test case without the required logical test data is not complete and vice versa.

The third decision on this level concerns the definition of the **logical test environment**. The decision comprises the kind

of tools, software and hardware needed during the execution of the test cases. The description of the logical test environment is also abstract similar to the specification of the logical test cases or test data and represents the general requirements on the test environment.

The last decision at test design level discussed here is related to the **logical test order**. This order refines the ideal test order considering dependencies between test cases as well as information about planned test environment factors. Execution efficiency and parallelism are main criteria influencing this decision.

E. Test Realization Level

The test realization level details the logical representation of the test cases, of the test data as well as of the test environment. It contains all decisions which influence the execution of a test case. This level contains decisions on the concrete test order, on concrete test cases, concrete test data and the concrete test environment. Setting up the **concrete test order** means to identify an actual executable test order considering the logical test order and the project environment factors. In parallel, the logical test cases are refined by **concrete test cases.** Thus, information on the specific behavior of the test case and the test object is added. Concrete test cases contain all information needed to execute the test case. To complete the specification of a concrete test case, the detailed description of **concrete test data** is needed. Consequently, it has to be decided which concrete "instances" of the logical test data are used in concrete test cases. The decisions on the **concrete test environment** consider the description of the logical test environments and the specification of the logical test cases. The concrete test cases need a corresponding concrete test environment (e.g. the specification of concrete hardware and software needed) in order to be executable.

F. Test Run Level

The test run level deals with the evaluation of test run results. After the execution of a test case the **test run evaluation** decides, whether the test run revealed a defect. If this is the case, a state (e.g. "open"), a priority (e.g. "critical") and a weight (e.g. system crash) have to be assigned to the corresponding defect [23].

G. Test Evaluation Level

This level contains the decision whether test activities can be finished. The decisions on the **test cycle evaluation** check whether the test end criteria have been fulfilled and whether every test focus has been tested with the required test intensity. Furthermore, the defects not found within this test cycle are estimated by using for example a metric like the defect detection rate. The decision not to finish the test cycle, leads to a new iteration of some (or maybe all) of the testing tasks and decisions.

III. Applying the Decision Hierarchy

We validated the decision hierarchy in several case studies by applying it in different contexts: As a framework for the comparison of system and integration testing processes (A) and as a framework for the evaluation of testing tools (B). Additionally, the decision hierarchy served as the basis for test process analysis in an industrial case study (C) and as a framework for classifying testing research (D). The results of these case studies are summarized below. A detailed description of the results can be found in the technical report in [5].

A. Evaluation Framework for System and Integration Testing Processes

We applied our decision hierarchy to the system testing process (STP) and integration testing process (ITP) in order to identify the specific issues and decisions of both processes by instantiating the generic decision hierarchy. Figure 2. summarizes the main results. It illustrates all decision levels as well as the corresponding decisions of the generic testing process (left column), the specific decisions of the system testing process (middle column) and of the integration testing process (right column). Specific decisions in the middle and the right columns refine corresponding decisions of the generic testing process at the same decision level. This is illustrated in Figure 2. by using two labels within one "decision box". The upper label of a box describes the decision of the generic testing process. The lower label specifies the corresponding specific decision of the STP, respectively of the ITP.

Within the STP as well as within the ITP, decisions concerning the test basis and test focus are refined. Both testing processes deal with different kinds of information and specifications, e.g. functional and quality requirements within the STP and components and dependencies within the ITP in order to decide on critical parts to be tested. On test strategy level the integration testing process defines several integration rules to provide guidelines for the later integration test order. Within the STP decisions on model coverage and the degree of automation refine the generic decisions.

At test design level both processes refine the decisions on the logical test environment and the logical test order. Within the STP, the kind of external systems and the automation tools to be used in the test execution phase are decided, whereas within the ITP decisions on the required stubs, drivers, monitors and the points of observation and control are made. Within the ITP the focus of the test order lays on the integration order and the integration step size, i.e. the order and the number of components added within one integration step. At this level, the STP decisions on the optimal test case order minimizing the setup-overhead for the test cases play an important role.

At test realization level the STP refines the decisions on concrete test data and concrete test cases whereas the ITP deals with decisions on the concrete test environment and the concrete test order. Especially for the STP, decisions concerning GUI steps are important in order to define the concrete test cases. Moreover, GUI data is used to select concrete test data. In parallel, the GUI layout, i.e. how the GUI data is arranged on the screen, influences the concrete test cases. At test realization level, the ITP defines a concrete test order considering the real completion time of every component, the integration rules, order, and step size. Furthermore, decisions on how to prepare the test object (e.g. inserting points of control and observation), how to implement concrete monitors, stubs and drivers have to be made. At the last two

decision levels there are no specific decisions within the STP and the ITP.

B. Evaluation Framework for Testing Tools

The decision hierarchy served as the basis for the design of a questionnaire used within a survey evaluating 13 commercial and open source test management tools [15]. The evaluation is primarily based on the information provided by tool vendors who completed the questionnaire. The goal was to analyse to what extent a decision is supported by a test management tool. Based on the decision hierarchy, questions addressing the functional characteristics of the testing tool can easily be derived. E.g. if a test management tool integrates requirements management functionality, it would provide support for decisions on specification level by facilitating the identification of functional and quality requirements.

C. Test Process Analysis

Based on our decision hierarchy, the testing process of an organisation was analyzed in order to find its strengths and weaknesses. The organisation we refer to provides system solutions in the area of remote operations. Testers in this organisation are organized in an independent testing group. The ratio of testers to developers is 1:4. The test process analysis was based on document reviews as well as on interviews. All interviewees are experienced testers, with up to ten years of experience. In the following we describe the results of our analysis.

All decisions at specification level are made by the requirements engineering team, whereas the rest of the decisions are made by the testing team. Furthermore, there are decisions made implicitly, e.g. all decisions at test goal and test strategy level and decisions made explicitly, e.g. all decisions at test design level. Implicit decisions are not documented, whereas explicit decisions are (partially) documented within test artefacts. All decisions on test goal and test strategy level are made implicitly. The testing team does not perform a risk analysis in order to make sound decisions on test foci or test intensities. Thus, the end of testing activities is not determined by criteria defined in advance, but by current test results and the "feeling" of the testing team regarding the maturity and quality of the product. The test team uses two "standard" test design techniques (domain testing and boundary value analysis). Other techniques are not considered and evaluated with respect to their efficiency in the project's context. Thus, decisions on the test model, the design technique as well as on coverage criteria are made implicitly, without a thorough analysis of alternatives.

Logical test cases and test data are explicitly defined on the basis of requirements and documented within a test management tool. Decisions concerning concrete test cases and test data are made explicitly and are mostly documented during test execution within test protocols. The decision on the concrete test order is made explicitly, but only documented in the case of a failed test run. A matrix of concrete test environments is also managed by the testing team. Decisions on logical test environments as well as on the logical test order are made implicitly and are not documented.

The evaluation of a test run is made explicitly for each executed test case. If a failure occurs a process concerning the life cycle of a defect is passed through, from its classification, localization and correction until its retest. At the end of a test cycle, the test team evaluates the results. This decision is made explicitly, but only summarizes the test results. Since the definition of test end criteria is not performed, the evaluation of the test cycle occurs without a reference to defined criteria.

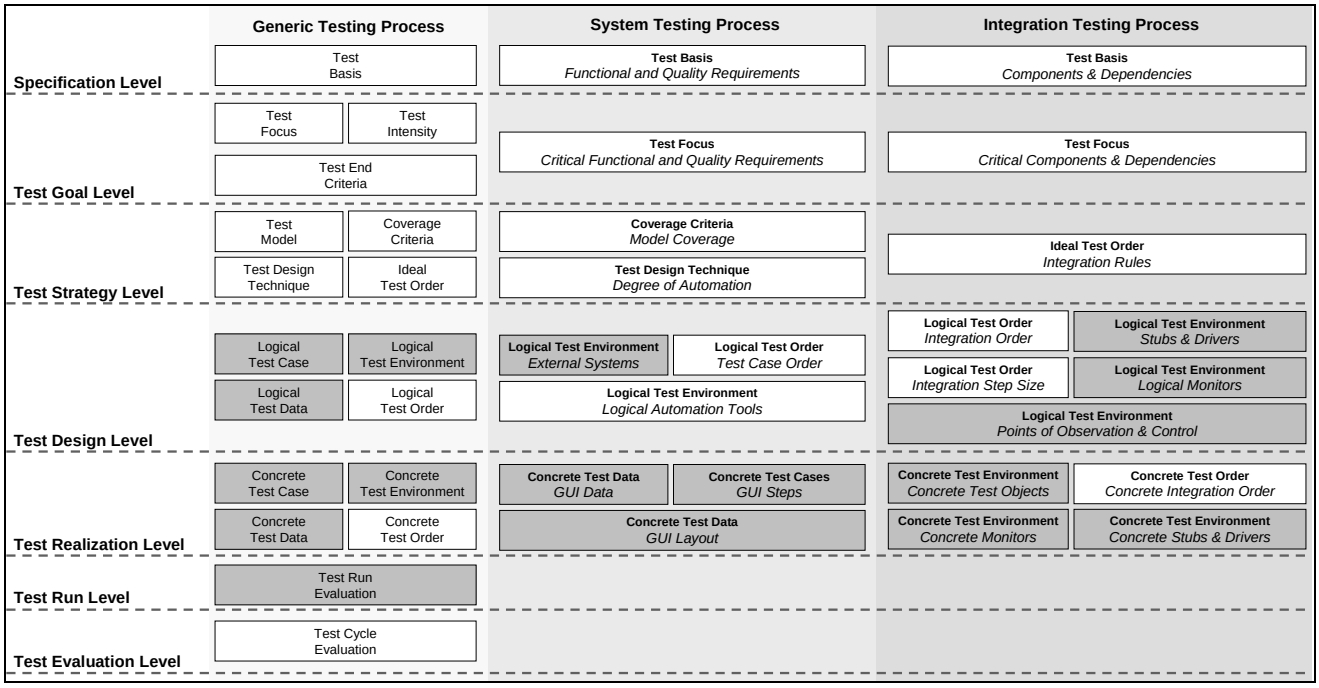

Figure 2. Specific decisions of the system and integration testing process

Implications: The decision based analysis highlights the following main strengths and weaknesses of the testing process. Missing involvement of the testing team into decisions at specification level leads to input which is not well suited to be used in the testing process. Thus, complex user scenarios are not part of the documentation provided by requirements engineers. However, these scenarios would be very precious for system testing as they lead to realistic test cases.

Another weakness concerns the unstructured decision process on test goal as well as on test strategy level. Thus, a thorough evaluation against goals is not possible. Improvement efforts should concentrate on methodologies which help testers to define objective and measurable goals in advance. A strength of the testing process is the thorough documentation of decisions concerning test cases and test data supporting the repeatability of test runs e.g. within regression testing.

D. *Evaluation Framework for Testing Approaches in the Literature*

The decision hierarchy can also be used as a framework for the comparison of different testing approaches. It permits the classification of approaches depending on whether they provide (automated) support for a specific decision or not. Figure 3. exemplifies how approaches for use case based testing can be compared to one another on the basis of the decision hierarchy, where this example considers only three of the seven decision levels. A complete overview of all approaches is presented in [14]. Comparing the approaches on the basis of the decision hierarchy allows the analysis of their similarities and differences. As illustrated in Figure 3. some decisions, e.g. the decision concerning the test model, are supported by all approaches, whereas other decisions, e.g. the decision concerning quality requirements, are partially supported by only a subset of the approaches.

IV. Related Work

A process model, which describes the main phases of the testing process, consisting of test planning, test design, test execution and test evaluation activities has been proposed in [23] by Spillner, Linz and Schäfer. In comparison to our approach, which explicitly focuses on all decisions to be made during the testing process, the process model described in [23] is very generic and does not take decisions into account. The IEEE standard for software test documentation [10] specifies all artifacts to be created during the testing process, e.g. test plan, test design specification, test case specification. The decisions made within the testing process are not part of the standard. Another group of related work comprises test process improvement models like TPI (Test Process Improvement) [16] or test maturity assessment models, e.g. TMM (Testing Maturity Model) [6]. The focus of these models is not the test process itself, but the steps for its improvement, respectively on criteria to assess the maturity of the organizational testing process.

A conceptual framework categorizing different decisions made during requirements engineering has been presented in [20] by Paech et al. and in [2] by Aurum et al., but these approaches do not consider decisions to be made during other phases of the software engineering process. Furthermore, the system Sysiphus supporting the documentation of decisions defined in [20] has been realized in [25]. Additionally, several approaches for the documentation of the decisions made during the software development process have been proposed in [10]. To the best of our knowledge there is no existing research and there are no case studies which particularly address the decision making process within quality assurance activities.

	Decisions		**Approaches**								
Decision level	**Generic tesing process**	**System testing process**	Path analysis [1]	Extended UCs [4]	TOTEM [6]	Structural Testing with UCs [8]	ASM Based Testing [9]	Requirements by Contracts [19]	Testing with UCs [21]	SCENT [22]	Simulation and Test Models [24]
Specification level	Test basis	*Functional requirements*	X	(X)	(X)	(X)	X	(X)	X	X	X
		Quality requirements							(X)	(X)	(X)
Test goal level	Test focus	*Critical functional requirements*	X	(X)		(X)		(X)			
		Critical quality requirements								(X)	
	Test intensity	Test intensity	(X)								
	Test end criteria	Test end criteria									
Test strategy level	Test model	Test model	X	X	X	X	X	X	X	X	X
	Coverage criteria	*Model coverage*	X	X	X	X	X	X	X	X	X
	Test design technique	*Degree of automation*			X		X	X		(X)	X
	Ideal test order	Ideal test order			(X)						

Figure 3. Applying the decision hierarchy to compare testing approaches, X = Approach supports decision, (X) = Approach partially supports decision

V. Conclusion

In this paper, we presented a generic decision hierarchy which contains decisions to be made during the testing process at different decision levels. We evaluated our hierarchy in four case studies.

Our decision hierarchy proved of value for both, for industry as well as for research applications. Practitioners get a deeper understanding of the complex decision making process during testing. Thus, the hierarchy can be used as an introducing guideline to the complex area of testing processes. Additionally, this approach increases the awareness of all decisions which have to be made during the testing process. The decision hierarchy is useful for researchers, too. At first, it enriches the body of knowledge on the subject of decision-making in the area of testing and builds the foundation for further research in the area of rationale management. Rationale management research aims at making design and development decisions explicit to all stakeholders involved. Additionally, as illustrated in the case studies, the decision hierarchy can be used by researchers as an evaluation framework in many contexts.

Based on our experience in applying this hierarchy in the case studies, we revealed that our approach is universal enough to be applied in different contexts. But, it is also specific enough to highlight the similarities and differences of the subject matters. Additionally, our approach is easily to be learned. Thus, students as well as practitioners get familiar with key issues of the testing process without having to get into details. Finally, our hierarchy eases the communication among testers by providing a common terminology.

Acknowledgment

We would like to thank all the interviewees for their cooperation and help in providing information and insight into documents. Furthermore, we would like to thank Andrea Herrmann for her helpful comments and Doris Keidel-Müller for reviewing previous versions of this paper.

References

[1] N. Ahlowalia, “Testing from Use Cases Using Path Analysis Technique”, International Conference On Software Testing Analysis & Review, 2002.

[2] A. Aurum, C. Wohlin, and A. Porter, „Aligning Software Project Decisions: a Case Study“. International Journal of Software Engineering and Knowledge Management, vol. 16, number 6, pp. 795 – 718, 2006.

[3] B. Beizer, “Software Testing Techniques”, Second Edition, Van Nostrand Reinhold, New York, 1990.

[4] R. Binder, “Testing Object-Oriented systems”, Addison-Wesley, 2000.

[5] L. Borner, T. Illes-Seifert, and B. Paech, “The Testing Process - A Decision Based Approach”, Technical Report, SWEHD-TR-2007-01, 2007.

[6] L. Briand, and Y. Labiche, “A UML-based Approach to System Testing”, Technical Report, Carleton University, 2002.

[7] I. Burnstein, T. Suwannasart, and C. R Carlson, “Developing a Testing Maturity Model for Software Test Process Evaluation and Improvement”, Proceedings of the IEEE International Test Conference on Test and Design Validity, 1996.

[8] A. Carniello, M. Jino, and M. Lordello, “Structural Testing with Use Cases”, WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina, 2004.

[9] W. Grieskamp, M. Lepper, W. Schulte, and N. Tillmann, “Testable Use Cases in the Abstract State Machine Language”, Second Asia-Pacific Conference on Quality Software, 2001.

[10] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, “Rationale Management in Software Engineering”. Springer-Verlag Berlin Heidelberg, 2006.

[11] IEEE Std. 829-1998, Software Engineering Technical Committee of the IEEE Computer Society, IEEE standard for software test documentation, USA, 1998.

[12] IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. New York, September 1990.

[13] International Software Testing Qualifications Board, ISTQB Standard Glossary of Terms used in Software Testing V1.1, 2005.

[14] T. Illes, and B. Paech, “An Analysis of Use Case Based Testing Approaches Based on a Defect Taxonomy”. In IFIP International Federation for Information Processing, vol. 227, Software Engineering Techniques: Design for Quality, ed. K. Sacha, Boston Springer, pp. 211-222, 2006.

[15] T. Illes, H. Pohlmann, T. Roßner, A. Schlatter, and M. Winter, „Software-Testmanagement Planung, Design, Durchführung und Auswertung von Tests - Methodenbericht und Analyse unterstützender Werkzeuge“, Heise Zeitschriften Verlag, 2006.

[16] T. Koomen, and M. Pol, “Test Process Improvement, A step-by-step guide to structured testing” Addison-Wesley, 1999.

[17] G.J. Meyers, “The Art of Software Testing”, John Wiley & Sons, New York, 1979.

[18] D. J. Mosley, and B. A. Posey, “Just Enough Software Test Automation”, Prentice Hall, July 2002.

[19] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jézéquel, “Requirements by contracts allow automated system testing”, Proc. of the 14th. IEEE International Symposium on Software Reliability Engineering (ISSRE'03), 2003.

[20] B. Paech, and K. Kohler, “Task-driven Requirements in object-oriented development”. In Leite, J. and Doorn, J. Perspectives on Requirements Engineering. Kluwer Academic Publishers 2003.

[21] C. Rupp, and S. Queins, „Vom Use-Case zum Test-Case“, OBJEKTspektrum, vol. 4., 2003.

[22] J. Ryser, and M. Glinz, “SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test”, Technical Report, University of Zürich, 2003.

[23] A. Spillner, T. Linz, and H. Schaefer, “Software Testing Foundations - A Study Guide for the Certified Tester Exam - Foundation Level - ISTQB compliant”. dpunkt.verlag, 2006.

[24] J. Whittle, J. Chakraborty, and I. Krueger, “Generating Simulation and Test Models from Scenarios”, 3rd World Congress for Software Quality, 2005.

[25] T. Wolf, and A. H. Dutoit, “Sysiphus: Combining system modeling with collaboration and rationale”, In http://wwwbruegge.in.tum.de/publications/includes/pub/wolf2004GIRE/wolf2004GIRE.pdf, 2004.