Requirements for Information Systems Model-Based Testing

ABSTRACT

In order to develop systems with high level of quality and low costs it is necessary to have adequate test tools and methods. We believe that the definition of a requirements catalog is one of the steps in such direction. This work presents a requirements catalog for information systems model-based testing that can be used as a basis for improving methods as well as a guide for the development of new methods and tools. The catalog was prepared based on the literature of the area and in the experience of several information systems developers.

1. INTRODUCTION

The software industry has the challenge of developing a great variety of software products with increasing quality and decreasing costs. Several types of initiatives have been developed in order to address this challenge. We believe that one important type of initiative is the definition of a requirements catalog capturing the goals, needs, expectations, and constraints of the developers or users of a certain area, as, for example, done by Hoffmann et al. [7]. Requirements catalogs help users and developers to compare and select methods and tools. They also help methods and tools designers to direct their efforts.

Despite the use of testing tools and methods in software development, systems are constantly delivered with an unreasonable amount of failures. These failures are not found, in part, because of inadequate testing tools and methods [13]. Model-Based Testing (MBT) involves, among other activities, test generation, execution, and evaluation using software models [3]. MBT is expected to allow more adequate software testing because it is rooted in automated procedures, avoiding manual error prone activities. Our main contribution in this paper is to present a requirements catalog for MBT methods, focusing on *System Testing* [8].

In order to simplify our work we decided to restrict the scope of our catalog to Information Systems (IS). Our work was developed with the help of a team of IS professionals, composed by coders, testers, software architects, and project managers. In this work we are concerned with the following IS features:

- The ability to manage large amounts of persistent data.
- The management of a large number of users accessing information.
- The utilization of Graphical User Interfaces (GUI).
- The integration with other enterprise applications.

We started this work preparing an initial requirements catalog based on some of the most important works in this area [1, 2, 4, 6, 14, 16]. We used this initial catalog to create a new MBT method and its supporting tool. We were convinced of the relevance of a requirements catalog since we noticed that there were requirements not addressed by all the considered methods and tools. All the aspects of our method are somehow captured in the catalog, whereas the reverse is not true. As emphasized by Hoffmann et al., a catalog is not a definitive document, rather it is subject to periodical review and modification as computer technology trends or domain development processes trends lead to new or strengthened requirements as well as to weakened or outdated ones.

The requirements catalog and the development of our MBT method, influenced each other, however the catalog is a broader effort, since, after an initial effort, it was extended by the above mentioned team of IS professionals, using several requirements elicitation techniques, such as brainstorm and JAD sessions [17].

The remainder of the paper is organized as follows. Section 2 sets up the context for the discussion of requirements and testing activities. Section 3 presents the requirements catalog. Section 4 concludes the paper and presents directions for future work.

2. TESTING AND MODEL-BASED TESTING METHODS

The Test Discipline presented here is based on the UP [11]. Despite the fact of being presented in the UP context, it represents the core testing activities described by a large

number of works, in particular our terminology is adherent to IEEE [9]. Most of the requirements presented here are related to a specific test activity. The first test activity is **Plan Test**, its purpose is to plan the testing efforts by describing a testing strategy, estimating the requirements and scheduling the testing effort. During the **Design Test** activity, the test cases and test procedures are identified and described. The **Implement Test** activity is responsible for the automation of the test procedures by creating test components. The **Perform System Test** and **Perform Integration Test** activities are related to the test executions. In these activities the tests are executed and the defects are reported to the appropriate person. The **Evaluate Test** activity compares the results with the goals outlined in the test plan in order to determine if the testing activity will continue or stop.

In this work we used three comprehensive MBT works to validate (in an informal sense) the requirements catalog. This was done by analyzing the identified requirements in the considered methods. Interestingly, the prioritization of the requirements done by the IS professional team was similar to the ones in the considered methods.

TOTEM [4] was chosen because it was the first MBT method based on UML. Although it does not have a compliant tool, it introduced several interesting concepts about how to use UML for test automation. MODEST [16] is an MBT method suitable for IS, it is also UML-based, and it has a supporting tool called MODESTo. AGEDIS [6] is an MBT method developed by a consortium headed by the IBM Research Laboratory in Haifa, with several important partners. There are several components and tools that have been integrated into the AGEDIS framework.

3. MODEL-BASED TESTING REQUIREMENTS FOR INFORMATION SYSTEMS

In this work we say that *a method prescribes a certain mechanism to meet a certain requirement*, instead of saying that *a method does something*. Our intention is to make clear that a method *should prescribe* how something is done. The people using, or, the tools supporting, the method are responsible for doing what is prescribed.

We use a structure similar to the one used by Hoffmann et al. [7]. They describe the requirements for requirements management tools and they used the hierarchical structure of requirements for software products proposed in ISO/IEC Standard for Evaluation of Software Products [10]. We use a similar hierarchy with three levels. The top level corresponds to the **workers** directly related to the requirement. The second level, differently from Hoffmann et al., corresponds to a **subsection's title** and names a goal, need, expectation or constraint. The third level corresponds to the text discussing several aspects related to the second level. Our enumeration of requirements simply try to summarize this text. In the third level there is also a discussion related to the implementation of the identified requirements provided the method implements the requirement. All requirements are prioritized according to the discussion in our requirements elicitation sessions. The priorities "high", "medium" and "low" are indicated by "(+++)", "(++)" and "(+)" respectively. We indicated the priorities in the second level in order to simplify the process.

The workers used to structure the requirements presentation are the UP **test workers** and the **Process Engineer**. The main workers of the test discipline in UP are the **Test Engineer**, **Component Engineer**, and **System Tester** [11]. The Test Engineer is responsible to plan the tests, deciding the test goals and a test schedule, select and describe the test cases and the corresponding test procedures, and for evaluating the system tests when they have been executed. The Component Engineer is responsible for the creation of test components that automate some of the test procedures. The System Tester is responsible for performing the system tests, and generating the suitable reports. The *Process Engineer*, among several tasks, is responsible for the software development process itself. This includes the continuous improvement of the process, recommending the adoption of key practices in software development.

The goals, needs, expectations, and constraints related to other workers, e.g. customers, are captured in the discussion of the considered workers.

3.1 Requirements from the Test Engineers

3.1.1 Verification of the Software and Storage relationship++

A system comprises the software and other components. Fig. 1 presents a possible organization for a system. The software is the system core. It implements complex and flexible structures. The other components are also important: the hardware platforms (Hardware), the communication resources (Communication), and the storage mechanisms (Storage). As mentioned in the Introduction, we are concerned with IS that require a Storage, and commonly uses Communication. Therefore, IS development must have persistent information modeling. This leads to the existence of a relation between the software and the storage mechanism. The MBT for IS should check this relation.

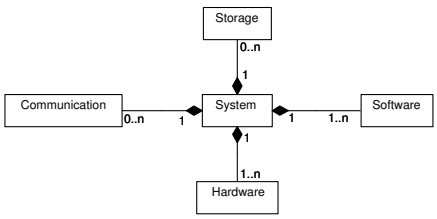

Figure 1: A system organization.

In practice, most of the MBT methods are concerned with Storage. They contain activities that prescribe the identification of the persistent data and their relationships. The MBT methods should use all the aspects of the persistent data descriptions, in order to assure the correctness of the system data. However testing only the software functionalities does not necessarily assure the correct behavior of Software and Storage relationship, since a software can use a data that is not generated by itself, as well as a software can generate data used elsewhere.

REQUIREMENT 1. An MBT method should prescribe the testing of the aspects of Software and Storage relationship.

TOTEM does not deal, in an explicit way, with the relation of Software and Storage, but it prescribes the use of the persistent data descriptions for test generation. This is done by creating class diagrams showing the persistent data, together with the definition of their relationships.

MODEST deals, in an explicit way, with the relation between Software and Storage, assuming the existence of a persistence layer. The persistent data description is similar to the description used in TOTEM, however, it prescribes the specification of additional information, and it uses this information to verify the relationship between Software and Storage, and to facilitate the oracle generation.

AGEDIS does not deal, in an explicit way, with the relation between Software and Storage. This relation is inferred from the behavioral model describing the SUT, created using using class and statechart diagrams. The behavior of each class is described in statechart diagrams integrated with an action language. The persistent classes are described in this model.

3.1.2 Test Generation for Functional Requirements+++

MBT automation methods should specify mechanisms for the automatic generation of test cases. This is hard to be captured in a single requirement, since it involves the tradeoff between specifying the requirement and discussing specific mechanisms that realizes the requirement. This is similar to the tradeoff between specifying "what" and "how", respectively. There are several aspects related to how to accomplish this in IS context: what interfaces for input/output data should be considered, the control transfer among these interfaces, and how the data input field in the interfaces will be used for testing generation purposes. These aspects are directly related to **criteria**: a test criterion is a means of deciding *what* a suitable set of test cases should be. A **criterion instance** determines *how* the data input fields should be used in order to generate tests. Criteria can be used for selecting the test cases or for checking whether or not a selected test suite is adequate in terms of coverage, determining if the test can be stopped [5]. An MBT method should prescribe the use of criteria or should define mechanisms to help the definition of criteria to be used during the test generation.

REQUIREMENT 2. An MBT method should prescribe test generation using test criteria.

TOTEM does not explicitly discuss the criteria used in the test generation. The messages in the sequence diagrams describing the system are used for test generation, but there is no detail about how this can be used to generate the tests.

MODEST prescribes the use of several criteria for test generation. MODESToo uses criteria defined by Andrews et al. [2] and Offut and Abdurazik [14], but it also defines other criteria.

AGEDIS prescribes directives for test generation indicating the criteria to be used. There are parameters sent to the test generator that can be used to specify the desired coverage degree. There are five test generation directives that can be used.

As mentioned before, one IS feature is the integration with other enterprise systems. Usually this is done by using software or hardware interfaces. These interfaces are mechanisms to communicate with software or hardware products. They define a boundary across which two independent entities meet, and interact or communicate with each other. They can be used to input or to output data from a system, therefore MBT methods should specify the modeling of these interfaces in order to generate tests using criteria related to this information.

Another IS feature is the use of GUI. These interfaces can exhibit different display formats enabling and disabling several GUI elements. There are several rules to access a specific GUI, demanding the generation of many input data in order to allow the control transfers among GUI instances. Again, the MBT methods should specify GUI modeling, including display format and navigation modeling, in order to generate tests using criteria related to this information.

In the context of IS the test criteria must focus on the issues directly related to IS features, e.g. integration with enterprise systems and GUI modeling, in order to assure more failure detection capability, but this requires the modeling of the aspects related to these IS features.

TOTEM prescribes the use of the description of the method's post-conditions to generate tests. This includes the post-conditions of the GUI related methods, but there is no analysis of the GUI in order to verify if they are correct in terms of presentation and there is no prescription about testing the control transfers among GUI instances.

MODEST has three activities related to GUI modeling. In these activities MODEST prescribes the description of the GUI, including the fields and commands accessible by end users, the specification of control transfers among GUI instances, and the description of the different presentations of them. This information is used for test generation simulating the use of the system by end users during test execution.

AGEDIS prescribes the description of all the system classes in its behavioral model, but there is no explicit prescription related to GUI. There is also no prescription about testing the control transfers among GUI instances.

An MBT method aimed at IS should be more concerned about GUI modeling since this is a critical feature in IS. A general MBT method does not emphasize GUI modeling because it aims at a broader range of applications.

3.1.3 Test Generation for Non-functional Requirements+++

As discussed before, Communication and Hardware are components of a System. These elements are directly related to the System Architecture (SA). SA is a key point in the development [12], and therefore its testing is important. It

is expected that the definition of a SA must take into account the Non-Functional Requirements (NFR), therefore a correct SA's implementation should assure that some of the NFR are met. MBT methods should prescribe the testing of SA, comprising the testing of some NFR.

Some MBT methods restrict the supported application classes in order to facilitate the testing of SA and the testing of NFR, but this could lead to a rigid structure supported by the methods. On the other hand, methods that do not restrict the supported applications can turn unfeasible, in terms of cost/benefits, the testing of SA.

REQUIREMENT 3. An MBT method should prescribe the testing of the system architecture.

As mentioned before, a common IS feature is the ability to manage large amounts of persistent data and the management of a large number of users. This is directly related to testing NFR. Some requirements, e.g. performance and stress testing can be automated by using models. Other NFR, e.g. usability, depends on human assessment and are hard to be automated. MBT methods should prescribe the test generation for NFR. This can be done by performing the possible NFR testing automation and helping the testing automation of the remaining NFR.

REQUIREMENT 4. An MBT method should prescribe test generation for NFR.

TOTEM does not specify any mechanism for verifying NFR and it does not constrain the architecture of its systems. MODEST is restricted to a pre-defined architecture, but there is an activity prescribing the specification of some architectural parameters. They are used for stress and performance test generation. AGEDIS prescribes the specification of test directives for test execution. These directives contain information about the architecture used in the test execution, including the resources configuration for distributed systems. Using this it is possible to specify configuration parameters for testing a restricted set of architectures. This can be used to run multiple instances of test cases, acting as stress testing from the functional tests created by the test generator.

3.1.4 *Oracle Generation+++*

A test oracle determines the expected results for a test case. As mentioned before, criteria are used to guide the test generation and this includes the ability to generate the expected test results. This is the hardest and the most expensive MBT requirement [3]. MBT methods should specify the requirements that allow the generation of a test oracle, preferably indicating the cost/benefits related to the practical use of these requirements.

REQUIREMENT 5. An MBT method should prescribe feasible mechanisms for automatic oracle generation.

Despite the fact that TOTEM describes aspects related to automatic generation of test oracles, there is no supporting tool that validates the described aspects.

The requirements that allow the automatic generation of a test oracle are present in MODEST. The feasibility of using these requirements is shown in MODESTtoo. The solution adopted by the method was to restrict the supported software architectures and to use some modeling conventions together with a formal language to specify constraints that indicate the system behavior. This approach is suitable for IS, but it seems that its extension is not trivial for other kinds of applications.

The requirements that allow the automatic generation of a test oracle are present in AGEDIS. The execution engine contains an oracle generated from the composition of a system behavioral model. After each test execution the system state is analyzed in order to verify its correctness. The approach is described as general, but the cost to create an oracle in some kinds of applications can be prohibitive.

3.1.5 *Test Documentation++*

Test generation using criteria can assure a certain degree of coverage, however it is necessary to evaluate the tests in order to decide for the creation of new ones. The human judgement is fundamental in deciding the necessity of further tests. MBT methods should prescribe the generation of the main test artifacts in order to help the test evaluation. Independently of the software process used during development, there are several artifacts directly related to the test activities [9]:

- Test Plan. This document details the scope, approach, resources and schedule of the test activities.
- Test Case Specification. This document specifies test inputs, expected results, and a set of execution conditions for an item to be tested.
- Test Procedure Specification. This document specifies a sequence of actions for the execution of a test.
- Test Incident Report. This document describes events occurred during testing, requiring further investigation.

REQUIREMENT 6. An MBT method should prescribe the automatic generation of test artifacts.

TOTEM does not prescribe the generation of test artifacts. MODEST prescribes the generation of a partial test plan, containing test case and test procedure specification. The result of a test execution generates a report similar to the *Test Incident Report* prescribed by IEEE. AGEDIS does not specify the generation of IEEE compliant test reports. However, the execution engine of AGEDIS stores the results of a test execution for future analysis. The report generator tool creates management documents describing the test cases, defects, models, and other artifacts of the testing process.

3.1.6 Test Evaluation+++

Test evaluation is directly related to the test evaluation activity of the test discipline. The objective of this activity is to evaluate test results and to consider the necessity of planning, designing, and implementing new tests. Most of the MBT methods are concerned with this evaluation. They mechanisms to improve the execution of this activity is implemented by providing reports detailing the test coverage using several criteria, e.g. branch coverage, condition coverage, statement coverage. Several commercial tools incorporate mechanisms to help this evaluation, allowing the creation of customized test coverage reports.

REQUIREMENT 7. An MBT method should prescribe mechanisms for test evaluation.

TOTEM does not specify mechanisms to help the evaluation of the test execution. MODEST does not prescribes mechanisms for test evaluation. It prescribes the use of test criteria in order to assure specific coverage levels. The papers describing the method also describe an initial idea of an evaluation mechanism based on mutation techniques. AGEDIS prescribes how to get data about the test execution coverage. AGEDIS tool set is integrated with a coverage tool called FoCus. Using this tool it is possible to generate customized test coverage reports.

3.1.7 Test Planning++

Test planning is the first activity of the test discipline. In this activity the resources and the schedule for the test activities are defined. Test planning requires the determination of the test team and the time to execute the test activities.

MBT methods should prescribe mechanisms to help test planning, possibly including: (i) the identification of the SUT complexity by indicating the parts deserving more attention during test activities, and (ii) use case stability estimative. Additionally, MBT methods should prescribe the existence of historical databases, in order to assure accurate estimation of test resources and test schedules.

REQUIREMENT 8. An MBT method should prescribe mechanisms to facilitate test planning.

The considered MBT methods do not consider test planning, since, as far as we know, there are no synergetic aspects between MBT and test planning. However, test planning is a fundamental test activity and its consideration under the MBT scope bears the promise of interesting results.

3.2 Requirements from the Component Engineer

3.2.1 Maintenance and Regression Support+++

Several types of maintenance are possible during system lifecycle, e.g. corrective, adaptive, and perfective [8]. The changes produced by system maintenance and by system changes during the other phases, normally render useless part of the tests. It is necessary to identify the affected parts in order to generate additional tests, discarding the obsolete ones. An MBT method should prescribe mechanisms to help the identification of the system parts affected by maintenance and system changes.

Ideally, all the regression tests that are not affected by maintenance should be kept. However the resource and time available for testing will dictate the regression tests that must be considered. An MBT method should prescribe mechanisms to help the selection of tests composing a regression battery.

REQUIREMENT 9. An MBT method should prescribe mechanisms for helping maintenance and systems change.

The MBT methods considered do not prescribe mechanisms for maintenance and systems change support. This is a significative difference between the requirements identified in this work and the requirements implemented by the analyzed methods. In our requirements session we identified the priority of this requirement as high (+++), and the analyzed methods did not consider it in their development. System change is not exclusive of the maintenance phase, it is usual during the development, hence the fundamentalness of regression support.

3.2.2 Methods Interoperability++

The testing infrastructure, similarly to the software itself, depends on specific technologies. In case of technology changes the tests should be updated incurring in increasing costs. An interesting alternative to this scenario is the creation of tests using a Platform Independent Model (PIM) and allowing transformations for a Platform Specific Model (PSM), as used in MDA[1]. A promising approach to MBT methods is to follow the MDA concepts, generating tests in a high level of abstraction, allowing transformations among the abstract format and specific technologies.

The UML Testing Profile [15] is one initiative in this direction. It defines a language for designing, visualizing, specifying, analyzing, constructing and documenting the artifacts of test systems. The UML Testing Profile is a test modeling language that can be used with all major object and component technologies and applied to testing systems in various application domains.

REQUIREMENT 10. An MBT method should prescribe mechanisms allowing the interoperability among methods, tools, and languages.

TOTEM does not prescribe mechanisms related to test standardization. MODEST uses part of the UML Testing Profile in its definition, representing the tests and the SUT in an independent format. AGEDIS developed the AGEDIS Modeling Language (AML) with two main goals: to support the creation of a SUT abstraction and to set meaningful test directives for the testing process. This language is used to described the SUT and the test model.

[1] http://www.omg.org/mda/

3.2.3 Manual Test Generation++

MBT methods cannot always assure the ideal coverage for testing. There are a great number of aspects to be considered during test generation, e.g. complexity, criticality factors, priority, and size. Additionally to coverage issues, test evaluation involves human beings that can always use subjective aspects to justify the creation of additional tests.

It is important for MBT methods, despite the reason, the prescription of mechanisms to support the incorporation of new tests, in addition to the automatically generated test suites. A common way to do this, implemented by many commercial tools, is to allow the recording of several actions during a system use, generating a script/program able to replay the recorded actions. Another alternative is to generate the main test components using a language supporting reuse, allowing the extension of the test programs by creating new tests programs.

REQUIREMENT 11. An MBT method should prescribe mechanisms to support the incorporation of new tests.

TOTEM does not prescribe support for additional tests. MODESToo has a specific component, called *Test Management*, for supporting the creation of additional tests. AGEDIS prescribes the support for additional test creation, providing a specific tool for this task.

3.3 Requirements from the System Tester

3.3.1 Automatic Test Execution+++

The automatic test execution is one of the first test discipline activities considered under the scope of MBT. It is responsible to execute the actions prescribed in the test procedures using the data specified in the test cases, comparing the actual results with the expected results also specified in the test cases. There are several comercial tools that support this activity. In the context of IS, the automatic test execution comprises several steps:

- Storage mechanism population, using the data required for a test.
- Interface elements instantiation for data input.
- Assignment of data to the respective user interface elements.
- Execution of test related actions.
- Verification of the results, comparing them against the expected.

MBT methods should prescribe all the required elements for automatic execution, e.g. code conventions and use of patterns. It is also important the ability to select tests in order to create test batteries, and to schedule batteries execution.

REQUIREMENT 12. An MBT method should prescribe the automatic test execution.

Despite the fact that TOTEM reports some aspects related to automatic test execution, there is no supporting tool to validate the reported aspects. MODEST prescribes the mechanisms that allowed MODESToo to contain a component responsible for the automatic execution. AGEDIS prescribes mechanisms that allow the automatic test execution. There is an execution machine that uses the abstract test suite and the test execution directives for this task. As far we know AGEDIS and MODEST do not support test grouping and scheduling.

3.3.2 Bug Tracking+

The test execution can find several bugs, demanding automatic support in order to keep track of them. Bug Tracking Systems allow developers to effectively keep track of bugs, facilitating the management activities. The absence of a mechanism to keep track of bugs can endanger the use of MBT methods, so it is interesting for the MBT methods to offer mechanisms to automate or to help this task.

REQUIREMENT 13. An MBT method should prescribe the support of bug registering and tracking.

TOTEM does not prescribe supporting for bug tracking. MODEST prescribes integration with some bug tracking system in order to keep track of bugs. AGEDIS prescribes bug tracking and automatic register of bugs found during test execution. A defect analysis and feedback tool was created for the AGEDIS tool set.

3.4 Requirements from the Process Engineer

3.4.1 Increase of Productivity and Quality+++

The main objective of the MBT methods is to improve the system quality, preferably decreasing the related costs. Most of the MBT methods are concerned about this and they have studies showing the effort reduction in the test activities by using them. They also show benefits in terms of test quality, creating tests with a higher failure capability detection. But surprisingly, few of the MBT methods consider the costs related to model creation in their studies. This is a critical element in MBT methods evaluation, since the model creation effort can be larger than the effort for manual execution of test activities.

Although this requirement and many other discussed previously can be considered obvious, there are many MBT methods that do not meet them. There are several actions that can improve the cost/benefit of the MBT methods. We emphasize the use of a dominant modeling, e.g. UML, and the restriction of the kind of application, e.g. IS, as some interesting concepts towards the improvement of productivity and quality of MBT methods.

REQUIREMENT 14. An MBT method should decrease the development cost and improve the test quality.

TOTEM does not describe an evaluation of its use. MODEST describes an initial experimental study presenting data that indicates benefits in terms of quality improvement and

cost reduction for IS development. AGEDIS presents several case studies, but we did not find a discussion related to costs. These case studies indicate that the use of the method can generates benefits related to test quality.

4. CONCLUSION

In this work we presented a set of requirements for MBT methods related to IS. As far as we know, there is no work presenting a requirements catalog for MBT methods, focusing on IS system testing.

When we started developing our MBT method we were convinced of the importance of creating an MBT requirements catalog. The initial versions of our catalog were mainly based on the works cited here. These versions of the catalog were used to guide our method development. This development, together with our experience in developing testing methods and tools for the industry and academy, and the help of several IS professionals, improved the catalog with a focus on IS.

The development of our method and the analysis of other MBT methods validates the requirements catalog, however we are studying strategies to more formally validate the catalog. We believe that a more formal validation of the requirements must be done in instances of the catalog, where, even requirement prioritizations, can be differently set.

We believe that our requirements catalog captures the needs, expectations, and constraints of IS development workers. Even though, this catalog was directly influenced by IS development background, we think that it can be quite useful as a basis for other domains as well.

5. REFERENCES

[1] A. Abdurazik and J. Offutt. Using UML collaboration diagrams for static checking and test generation. In *Proceedings of the 3rd International Conference on the Unified Modeling Language (UML'00)*, pages 383–395, York, UK, October 2000.

[2] A. Andrews, R. France, S. Ghosh, and G. Craig. Test adequacy criteria for UML design models. *Jornal of Software Testing, Verification, and Reliability*, 13(2):95–127, June 2003.

[3] R. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

[4] L. Briand and Y. Labiche. A UML-based approach to system testing. In *Proceedings of the 4th Unified Modeling Language Conference (UML'01)*, pages 194–208, Toronto, Canada, October 2001.

[5] J. Goodenough and S. Gerhart. Towards a theory of test data selection. *IEEE Transactions on Software Engineering*, 1(2):156–173, June 1975.

[6] A. Hartman and K. Nagin. The AGEDIS tools for model based testing. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004)*, Boston, Massachusetts, USA, July 2004.

[7] M. Hoffmann, N. Kuhn, M. Weber, and M. Bittner. Requirements for requirements management tools. In *Proceedings of the IEEE International Requirements Engineering Conference (RE'04)*, pages 301–308, Kyoto, Japan, September 2004.

[8] IEEE. *IEEE Standard Glossary of Software Engineering Terminology - IEEE Std.610.12-1990*. IEEE Computer Society, 1990.

[9] IEEE. *IEEE Standard for Software Test Documentation - IEEE Std 829-1998*. IEEE Computer Society, 1998.

[10] ISO/IEC. *ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use*. International Organization for Standardization, 1991.

[11] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[12] P. Kruchten. Architectural blueprints - the "4+1" view model of software architecture. *IEEE Software*, 12(6):42–50, November 1995.

[13] NIST. Planning Report 02-3, National Institute of Standards and Technology, http://www.nist.gov/, 2002.

[14] J. Offutt and A. Abdurazik. Generating tests from UML specifications. In *Proceedings of the 2nd Unified Modeling Language Conference (UML'99)*, Fort Collins, Colorado, USA, October 1999.

[15] OMG. *UML Testing Profile*. Object Management Group, http://www.omg.org, March 2003. last access on november 2004.

[16] P. Santos-Neto, R. Resende, and C. Pádua. A method for information system testing automation. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, Porto, Portugal, June 2005.

[17] J. Wood and D. Silver. *Joint Application Development*. John Wiley & Sons, 1995.