Including Model-Based Statistical Testing in the MATERA Approach

Andreas Bäcklund, Fredrik Abbors, and Dragos Truscan

Åbo Akademi University, IT Dept., Joukahaisenkatu 3-5B, 20520, Turku, Finland
Andreas.C.Backlund@abo.fi, Fredrik.Abbors@abo.fi,
Dragos.Truscan@abo.fi

Abstract. In this paper, we present a Model-Based Testing (MBT) approach in which statistical data contained in Unified Modeling Language (UML) models are used to prioritize test cases. The models are used by a test derivation tool for automatic generation of test cases. The statistical data included in the models is used by the tool to determine the order of the resulting test cases before being implemented and executed. The test outputs are analyzed and information about requirement coverage is gathered. Based on the gathered statistics, the results are automatically fed back to the UML models to prioritize those sections of the system where failures are frequent.

1 Introduction

The complexity of software systems is constantly increasing. Hence, the amount of tests needed to properly test a software system is also increasing. Software companies usually do not have enough time to run all their test cases, and are therefore forced to prioritize them in such a way that the test cases cover as much functionality of the system as possible [1].

Especially in the telecommunications domain, which we target in this paper, the amount of test cases needed to be executed against the System Under Test (SUT) is rather large, and in practice only a part of these tests can be executed. Thus, there is a need to be able to order the test cases based on their importance. By determining the priority-specific paths within the system, it is possible to order the test cases in such a way that test cases of statistically higher priority are executed before others. In this way, specific sections of the system can be given higher priority, resulting in earlier execution of test cases running the highest prioritized paths of the system.

There are several benefits with using statistical testing [2, 3]. One of the main benefits is that more testing effort can be put into the most important sections of SUT, while less important section can be left less tested. Another benefit of conducting statistical testing is that statistical data from previous iterations of the testing process can be included in latter iterations, in order to target the test execution towards the system sections that are more important or yielded more failures.

Model-Based Testing (MBT) [4] is a testing approach that addresses some of the shortcomings in traditional testing by using an abstract representation (a *model*) of the system for automatic generation of test cases. The models can be implemented either as program code representations or as graphical representations using graphical specification languages, such as the Unified Modeling Language (UML) [5] or various tool

specific languages. The main idea with MBT techniques is to automatically generate tests by applying algorithms that are able to explore paths through the model.

According to [1], statistical testing can be integrated into the development process at the point when requirements have been gathered and approved. In other words, statistical testing can be initialized at the same phase as the model construction in MBT. Combining this with the benefits of using models to prioritize certain sections of the SUT, makes statistical testing beneficial when used in a MBT process.

There are several advantages of using MBT in a software development process. One advantage is that large amounts of tests can be generated in a short amount of time when there exists an appropriate model representation of the system. This adds additional value especially to conducting regression testing in the end of the software development project. Another advantage is that models are usually easier to modify than manually created test cases, which especially benefits projects where requirements are changing frequently. The third advantage is that the modeling of the system can be initiated immediately when the requirements have been specified. This means that a testing process using MBT can already be initiated in the design phase. Since the test model in MBT is typically an abstract representation of the system, it is easier to maintain it compared to manually written test cases.

2 Related Work

Previous research on combining statistical testing and MBT has been done under the acronym Model-based Statistical Testing (MBST). For instance, Prowell [6] presents an approach in which the transitions of a test (usage) model are annotated with probability of occurrence information that is later used during test generation by the JUMBL tool. A similar approach, targeted at telecommunication protocols, is presented in [7]. An operational profile (a Markov process) is used to describe the usage and behavior of the SUT. The probabilities included in the operational profile are later on used during test generation. In our approach we will use a test model describing the behavior of the system. The generated test cases will be ordered *after* test generation based on the statistical information, and information resulted from test reporting will be used to update the priorities for the generated test cases. In addition, requirements of the system are modeled and traced throughout the testing process.

Other similar work on MBST is presented in [8–10]. For instance, the author of [8] uses UML activity diagrams to express high level requirements. The nodes and edges in the activity diagram are assigned with weights indicating priority, based on complexity and possibility of occurrence of defects. The activity diagram is later translated into a tree structure, from which prioritized test scenarios are generated.

Work related to statistical testing has also been preformed in the context of the MaTeLo tool [11, 12]. In MaTeLo, test cases are generated from statistical models of the SUT expressed using Markov chains usage models. However, while MaTeLo-based approaches utilize a usage model for describing the SUT, our approach utilizes a system model to represent the SUT.

In [9] the author presents an approach for using MBST together with time durations to test real-time embedded systems. The author's approach differs slightly from ours, since it uses statistical information to test the reliability of the system. In the approach,

reliability is tested by generating test cases from a model that represents the actual use of the system. In our approach, statistical information about the system is not used to test the intended usage of the system, but rather to order test cases according to weighted probabilities calculated from statistics of requirement priority and use case probability.

The most similar approach is presented in [10]. Here the authors take advantage of an approach in which they go from a requirements document, via a statistical model, to a statistical test report. Similarly to our approach, their approach benefits from a high degree of automation in each phase of the testing process.

3 Overview of MATERA

MATERA (Modeling for Automated TEst deRivation at Åbo Akademi) [13] is an approach for integrating modeling in UML and requirement traceability across a custom MBT process (see Figure 1). UML models are created from the system requirements, using a UML modeling tool. The models are validated by checking that they are consistent and that all the information required by the modeling process is included. Consequently, the models are transformed into input for the test derivation tool. The resulting test cases are executed (after being concretized) using a test execution framework. The results of the test execution are analyzed and a report is generated. Requirements are linked to artifacts at different levels of the testing process and finally attached to the generated test cases. The approach enables requirements to be backtraced to models in order to identify which test cases have covered different modeling artifacts or from which part of the models a failed test case has originated.

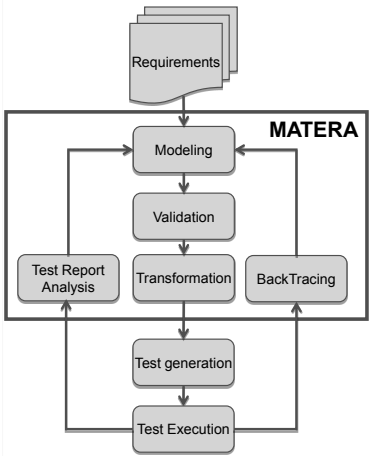


Fig. 1. MATERA process

4 Statistical Approach for MATERA

Our statistical approach relies on two sources of information: (1) that the functionality of the system (use cases) has associated *probability* values, depicting the chances for functionality to be invoked by the external user of the system during the use of the SUT; (2) that the requirements of the system are classified based on their importance (for testing) by associating them with *priority* values. The priorities and probabilities of the system are considered to be given from external sources (e.g., system requirements or stakeholder recommendations) and *a priori* to the first iteration of the testing process. In latter test cycles, the priorities can be adjusted based on statistics of uncovered requirements from previous test cycles for targeting the testing process towards a certain part of the SUT.

There is a slight difference between probability and priority. Even though they both mean that specific sections of the SUT are prioritized, it is important to recognize that probability is part of the model, while requirement priority is a property for ordering system requirements according to importance. Hence, UML use case elements are given a probability value indicating the chance of the use case to be executed, whereas requirements are given a priority value indicating their importance for testing. The values are manually assigned to each use case in part. The two types of values are then combined in the test model from where test cases are generated. Each resulting test case will have a *weighted priority* calculated based on the cumulative probabilities and priorities of the test path in the model. The weighted priority will be used for determining the test execution order. In the following, we delve into more details related to each phase of the process.

4.1 Requirements Modeling

The process starts with the analysis and structuring of the informal requirements into a Requirements Model. The requirements diagrams of the Systems Modeling Language (SysML) [14] are used for this purpose. Requirements are organized hierarchically in a tree-like structure, starting from top-level abstract requirements down to concrete testable requirements. Each requirement element contains a *name* field which specifies the name of the requirement, an *id* field, and a *text* field. For the purpose of statistical testing, requirements are also given a `priority` value (see Figure 2). The priority value is a property describing the importance of the requirement. During the modeling process the requirements are traced to different parts of the models to point out how each requirement is addressed by the models. By doing this we ensure the traceability of requirements and that priority information is propagated to other model artifacts.

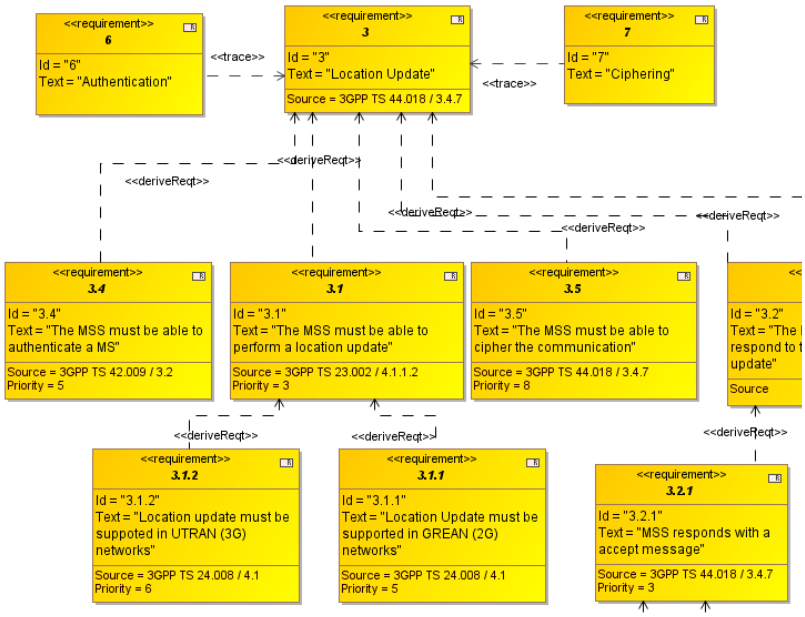


Fig. 2. Requirement Diagram with priorities

4.2 System Modeling

In this phase, the SUT is specified using UML. In our modeling process, we consider that several perspectives of the SUT are required in order to enable a successful test derivation process later on. A *use case diagram* is used to capture the main functionality of the system. *Sequence diagrams* are used to show how the system communicates with external components (in terms of sequence of messages) when carrying out different functionality described in the use case diagram. A class diagram is used to specify a *domain model* showing what domain components exist and how they are interrelated through interfaces. A *behavioral model* describes the behavior of the system using state machines. *Data models* are used to describe the message types exchanged between different domain components. Finally, *domain configuration models* are used to represent specific test configurations using object diagrams. Each use case is given a probability value which indicates the chance of the use case being executed (see Figure 3).

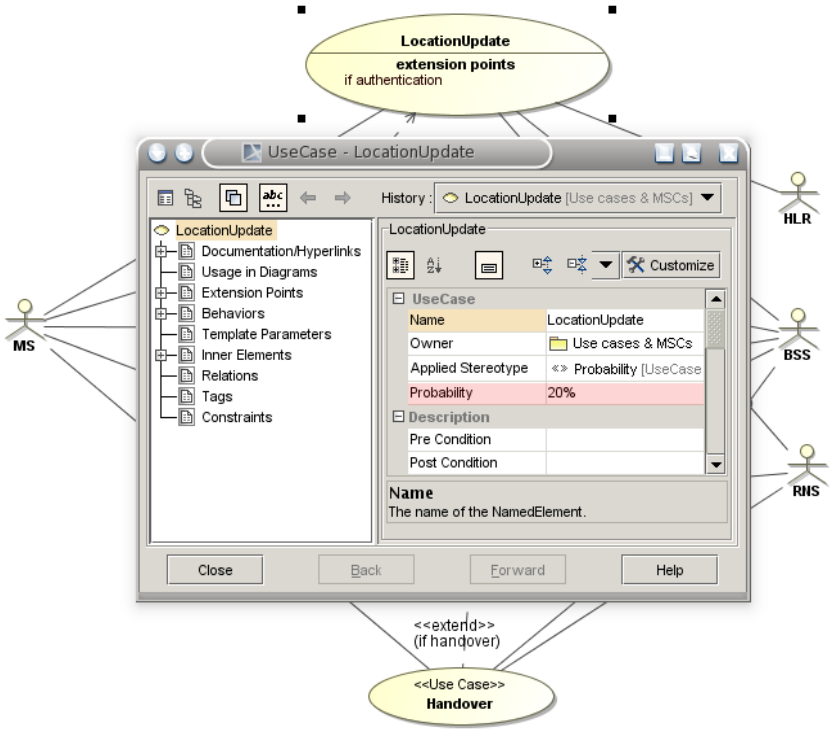


Fig. 3. Use case diagram with probability

The state model describing the expected behavior of the system is the pivotal artifact for test generation. According to the MATERA approach, leaf requirements are linked to transitions in the state machine to enable requirements traceability and requirements coverage during test generation. Thus, the priority of each requirement will be asso-

ciated to the corresponding transition. Similarly, use case probabilities are manually linked to the state model, as use cases are related with one or several starting points in the state machine diagram (see Figure 4). This enables the test generation tool to determine the weighted probability of certain paths through the state model. Before the tests are generated, the consistency of the UML models is checked using custom defined Object Constraint Language (OCL) rules [15].

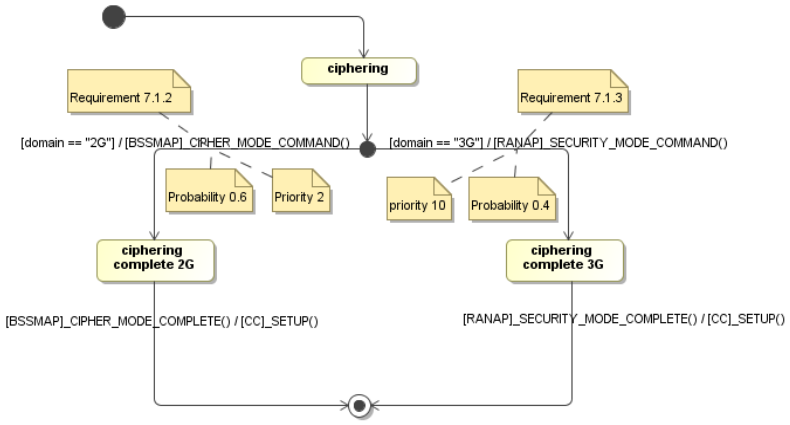


Fig. 4. UML state machine diagram

4.3 Test Case Generation

In the MATERA approach, the UML models are translated into a representation understood by a test generation tool, namely Qtronic [16], using the transformation described in [17]. During the translation, the priority and probability values are propagated to the new model representation. Test cases are generated by the tool based on the selected structural coverage criteria (e.g., state, transition, and requirement coverage, respectively), without taking into account priority and probability annotations.

4.4 Test Case Ordering

After the test cases have been generated, the test generation tool can determine the generation order of test cases based on the annotated probability and priority values. For each generated test case, a weighted probability is calculated based on the algorithm implemented by the test generation tool described in [18]. The weighted probability is calculated from both the use case probability and the requirement priority and determines the sequence in which test cases are ordered (see Figure 6). Test cases are finally rendered into executable test scripts using an adapter for concertizing test cases into executable scripts.

4.5 Test Execution

Test scripts are executed against the SUT using a test executor tool. The test scripts are executed in the order determined by the test generation tool. If only a part of the test suite can be executed, e.g. due to restricted testing time, ordering tests according to probability and priority ensures that the most important tests are executed. The execution of test scripts is monitored and the results are stored in log files. The log files contain information about the test execution, e.g. messages sent and received by the SUT, tested and untested requirements, used resources, etc. The log files together with the test scripts serve as a source for the test results analysis.

4.6 Test Log Analysis

By parsing logs and scripts and comparing these against each other it is possible extract statistical data from the test run. The extracted data describe requirements that have been successfully tested, requirements that have been left uncovered, and during testing of which requirements that failures have occurred.

The analysis of the test execution is presented in a HTML report (see Figure 5) generated by the MATERA tool-set. The report consists of two sections, one for General Test Execution Statistics and one for Requirements Information. The General Test Executions Statistics section contains information about the number of test cases that passed and failed. The Requirements Information section contains information about the requirement coverage. Finally, the test cases are presented in a Traceability Matrix.

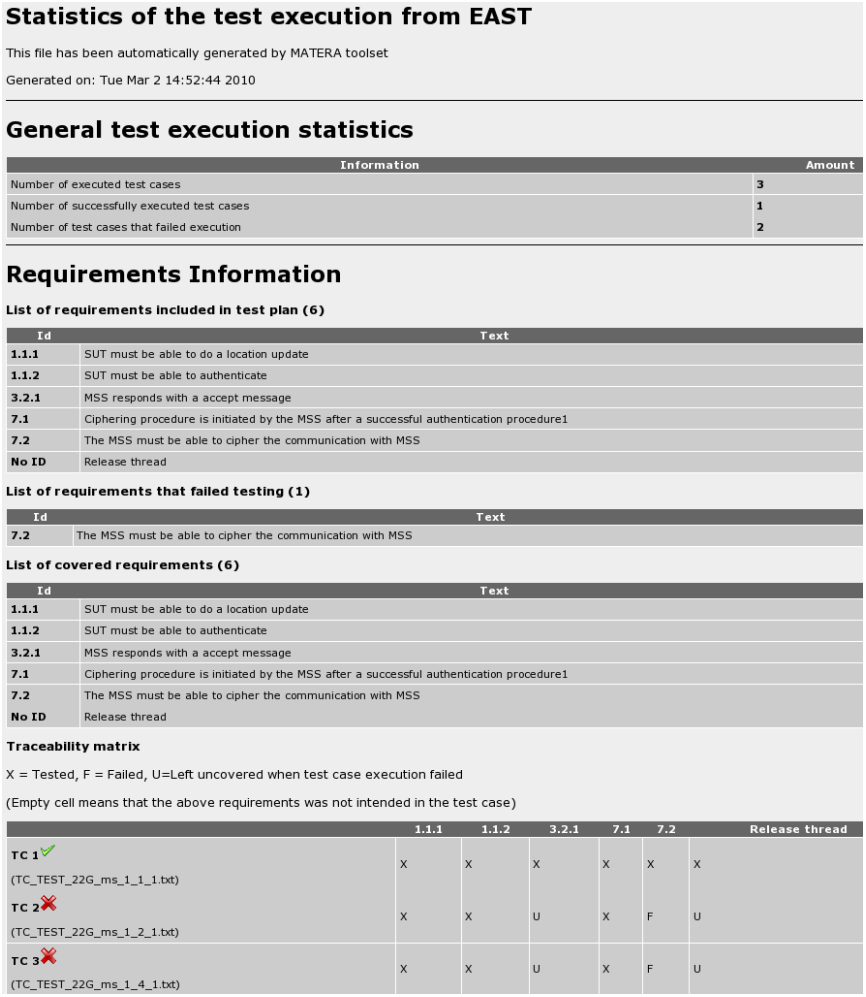
4.7 Feedback Loop

In the feedback loop, the statistical information gathered in the test log analysis is used to update priority of requirements that failed or were left uncovered during testing. The feedback loop is implemented as a part of the MATERA tool-set and allows the modeler to read in the analyzed statistics and update priority values for requirements in the UML models without user intervention.

The feedback loop is the main actor for targeting the test execution towards the parts of the system that had most failures. This is done by incrementally increasing the priority of the failed and uncovered requirements, such that they will counterbalance the effect that the probabilities of the use cases have on the ordering of tests. As testing progresses and the process is iterated several times, the importance (priority) of requirements will change according to how well they have been tested. Providing a feedback loop which updates the requirement importance automatically, will result in that the failed and uncovered requirements are included in the test cases that are ordered first in the test execution queue.

However, if requirement importance is changed due to external factors that cannot be derived from statistics, the tester can choose to manually change the priority of requirements directly in the models at any time.

The feedback module is executed from the MATERA menu in MagicDraw. When initialized, the module collects test data from a user specified folder holding test logs and test scripts from the last test execution. Based on these statistics, the priority values for requirements that need to be tested more thoroughly in a subsequent test iteration are



Statistics of the test execution from EAST

This file has been automatically generated by MATERA toolset

Generated on: Tue Mar 2 14:52:44 2010

General test execution statistics

Information	Amount
Number of executed test cases	3
Number of successfully executed test cases	1
Number of test cases that failed execution	2

Requirements Information

List of requirements included in test plan (6)

Id	Text
1.1.1	SUT must be able to do a location update
1.1.2	SUT must be able to authenticate
3.2.1	MSS responds with a accept message
7.1	Ciphering procedure is initiated by the MSS after a successful authentication procedure1
7.2	The MSS must be able to cipher the communication with MSS
No ID	Release thread

List of requirements that failed testing (1)

Id	Text
7.2	The MSS must be able to cipher the communication with MSS

List of covered requirements (6)

Id	Text
1.1.1	SUT must be able to do a location update
1.1.2	SUT must be able to authenticate
3.2.1	MSS responds with a accept message
7.1	Ciphering procedure is initiated by the MSS after a successful authentication procedure1
7.2	The MSS must be able to cipher the communication with MSS
No ID	Release thread

Traceability matrix

X = Tested, F = Failed, U=Left uncovered when test case execution failed

(Empty cell means that the above requirements was not intended in the test case)

	1.1.1	1.1.2	3.2.1	7.1	7.2	Release thread
TC 1 (TC_TEST_22G_ms_1_1_1.txt)	X	X	X	X	X	X
TC 2 (TC_TEST_22G_ms_1_2_1.txt)	X	X	U	X	F	U
TC 3 (TC_TEST_22G_ms_1_4_1.txt)	X	X	U	X	F	U

Fig. 5. Statistical Report

incremented with a predefined coefficient and automatically updated in the requirement models.

5 Tool Support

In our current approach we use No Magic's MagicDraw [19] modeling tool for creating and validating the UML models. The Graphical User Interface (GUI) of the MATERA tool-set has been implemented as a plug-in for MagicDraw. The purpose of the MATERA tool-set is to extend the capabilities of MagicDraw for specifying system models and using them as input for automatic test generation.

For automatic test case generation we use Conformiq's Qtronic [16]. Qtronic is an Eclipse based tool to automate the design of functional tests. Qtronic generates tests and

executable test scripts from abstract system models based on selected coverage criteria. An example of a test case sequence ordered by probability is shown in Figure 6. The models for Qtronic are expressed using the Qtronic Modeling Language (QML). QML is a mixture of UML State Machines and a super set of Java, used as action language. The UML state machines are used to describe the behavior of the SUT and QML is used to represent data and coordinate the test generation. By using a custom Scripting Backend (adapter), Qtronic generates executable test scripts for the Nethawk's EAST test executor framework [20].

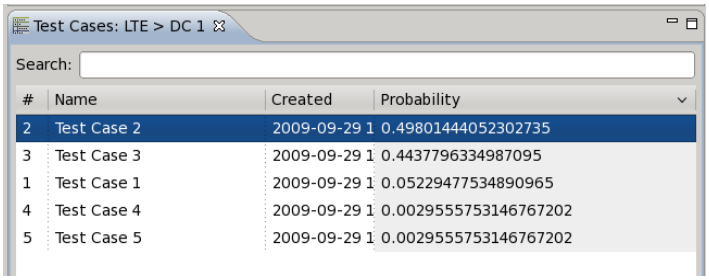


Fig. 6. Test case sequence ordered by weighted probability in Qtronic

The EAST Scripting Backend in Qtronic is the main actor for rendering the test scripts. When the abstract test cases are selected for execution, they are rendered to test scripts, loaded into the EAST test executor, and executed against the SUT. The test executor produces logs from the test case execution, which are used as source for the statistical analysis in the MATERA tool-set.

6 Conclusions

In this paper, we have presented a model-based testing approach in which statistical information is included in the system models and used for ordering of test cases. The approach benefits from a highly integrated tool chain and a high degree of automation. To handle complexity, the system is described from different perspectives using a different UML model for each perspective. Statistical information is described in use case and requirement diagrams, via priority and probability annotations. Traceability of requirements is preserved in each step of the testing process and can be gathered as statistics for later test cycles.

During test generation, test cases are ordered based on the statistical information contained in the models. After each test run, statistical information is gathered and fed back to the models in a feedback loop. The statistical information serves as basis for updating the information contained in the models to prioritize tests for those parts of the system where failures are discovered.

Future work will be to extract additional information from test logs. Since the test logs contain detailed information about messages sent and received from the SUT, this information could be extracted and presented to the user. For example the HTML test

report could be extended to include sequence diagrams for each test case. The tester could then examine failed tests in more detail, e.g. see what messages has been sent and received and what values were used, to manually adjust priorities and probabilities in the model. It could also facilitate the debugging of possible errors in the model.

References

1. Weber, R.J.: Statistical Software Testing with Parallel Modeling: A Case Study, Los Alamitos, CA, USA, IEEE Computer Society (2004) 35–44
2. Mills, H.D., Poore, J.H.: Bringing Software Under Statistical Quality Control. Quality Progress (nov 1988) 52–56
3. Whittaker, J.A., Poore, J.H.: Markov analysis of software specifications. ACM Trans. Softw. Eng. Methodol. (1) (1993) 93–106
4. Utting, M., Pretschner, A., Legeard, B.: A Taxonomy of Model-Based Testing. Technical report (April 2006)
5. Object Management Group (OMG): OMG Unified Modeling Language (UML), Infrastructure, V2.1.2. Technical report (November 2007)
6. Prowell, S.J.: JUMBL: A Tool for Model-Based Statistical Testing. In: HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9, Washington, DC, USA, IEEE Computer Society (2003)
7. Popovic, M., Basicevic, I., Velikic, I., Tatic, J.: A Model-Based Statistical Usage Testing of Communication Protocols. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS) (2006) 377–386
8. P.G., S., Mohanty, H.: Prioritization of Scenarios Based on UML Activity Diagrams. First International Conference on Computational Intelligence, Communication Systems and Networks (2009) 271–276
9. Böhr, F.: Model Based Statistical Testing and Durations. In: 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, IEEE Computer Society's Conference Publishing Services (CPS) (March 2010) 344–351
10. Bauer, T., Bohr, F., Landmann, D., Beletski, T., Eschbach, R., Poore, J.: From Requirements to Statistical Testing of Embedded Systems. In: SEAS '07: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems, Washington, DC, USA, IEEE Computer Society (2007)
11. All4Tec: MaTeLo http://www.all4tec.net.
12. Dulz, W., Zhen, F.: MaTeLo - Statistical Usage Testing by Annotated Sequence Diagrams, Markov Chains and TTCN-3. International Conference on Quality Software (2003) 336
13. Abbors, F., Bäcklund, A., Truscan, D.: MATERA - An Integrated Framework for Model-Based Testing. In: 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems (ECBS 2010), IEEE Computer Society's Conference Publishing Services (CPS) (March 2010) 321–328
14. Object Management Group (OMG): Systems Modeling Language (SysML), Version 1.1. Technical report (November 2008)
15. Abbors, J.: Increasing the Quality of UML Models Used for Automatic Test Generation. Master's thesis, Åbo Akademi University (2009)
16. Conformiq: Conformiq Qtronic (2009) http://www.conformiq.com.
17. Abbors, F., Pääjärvi, T., Teittinen, R., Truscan, D., Lilius, J.: Transformational Support for Model-Based Testing–from UML to QML. Model-based Testing in Practice 55
18. Conformiq: Conformiq Qtronic User Manual. (2009) 131–134 http://www.conformiq.com/downloads/Qtronic2xManual.pdf.
19. No Magic Inc: No Magic Magicdraw (2009) http://www.magicdraw.com/.
20. Nethawk: Nethawk EAST test executor (2008) https://www.nethawk.fi/.