Characterization of Model-based Software Testing Approaches

Arilo Claudio Dias Neto[a]

Rajesh Subramanyan[b]

Marlon Vieira[b]

Guilherme Horta Travassos[a]

PESC/COPPE/UFRJ[a]

SIEMENS CORPORATE RESEARCH[b]

Accomplished by September-December 2006

Published August-2007

Summary

Figure References

Table References

1 Introduction

Software Testing is a type of Quality Assurance technique consisting of the software product dynamic analyses. In other words, software engineers execute the software aiming at to make it to fail. It contributes for future defect detection by allowing the software engineers to look for the errors that made the software to fail. Besides, it could contribute with the confidence improvement that the software product is correct (adapted from ROCHA *et al*., 2001).

In the software development process, most of the defects are human based, and despite the using of better development methods, tools or professionals, defects remain present in the software products (HOWDEN, 1987). This scenario highlights the importance of testing for the software development, since testing is one of the last possibilities to evaluate the software product before its deployment to the final user (PRESSMAN, 2005). Testing must be accomplished in different levels (unit, integration, system, acceptance), in accordance with the software development process activities.

According to Beizer (1990) and Juristo *et al*. (2004), Software Testing can be considered the most expensive activity into the software development process, and thus, it needs an efficient planning to avoid loss of resources and behind schedule. Myers (1979) said that one of the factors that influence testing cost is the total of designed test cases, because for each test case, resources (such as material and humans) must be allocated. Therefore, to improve the use of such resources, it could be interesting to define some criteria regarding the testing coverage and test cases generation:

- **Testing Coverage Criteria:** The Software Testing Coverage Criteria allow the identification of the software parts that must be executed to assure the software quality and to indicate when the software has been enough tested (RAPPS and WEYUKER, 1982). In other words, it allows the identification of the percentage of the software that has been evaluated by a set of test cases (ROCHA *et al*., 2001).
- **Test Cases Generation Criteria:** rules and guidelines used to generate a set of test cases “T” adequate for a software product (ROCHA *et al*., 2001).

Over there these concerns related to the testing planning, it is necessary to simplify or to automate the testing execution or its regression (test cases re-execution after the software changing process). The generation of test cases can impact testing coverage.

Into this context, Model-based Testing (MBT) appears to be a feasible approach to control the software quality, reducing the costs related to testing process, because test cases can be generated from the software artifacts produced throughout the software development process. Exploring MBT makes software engineers able to accomplish testing activities in parallel with the development of software.

Model-based Testing consists of a technique for the automatic generation of test cases using models extracted from software artifacts (i.e. system requirements) (DALAL *et al*., 1999). For its using, it's necessary that some sort of formalized software model definition (such as formal methods, finite state machines, UML diagrams, etc.) has been used to describe the software or system behavior.

Despite the fact that MBT could be confound with Test Case Generation, MBT uses models developed during any software development phase to identify (or generate) the set of test cases. Test Case Generation is just one of the tasks of the software testing process, and it could be accomplished even not using a formalized software model.

The Figure 1 describes specific activities of Model-based testing:

- **Build the model**
- **Test Case Generation**
  - **Generate expected inputs**
  - **Generate expected outputs**
- **Run testing**
- **Compare actual outputs with expected outputs**
- **Decide on further actions** (whether to modify the model, generate more test cases, stop testing, or estimate the software reliability (quality))

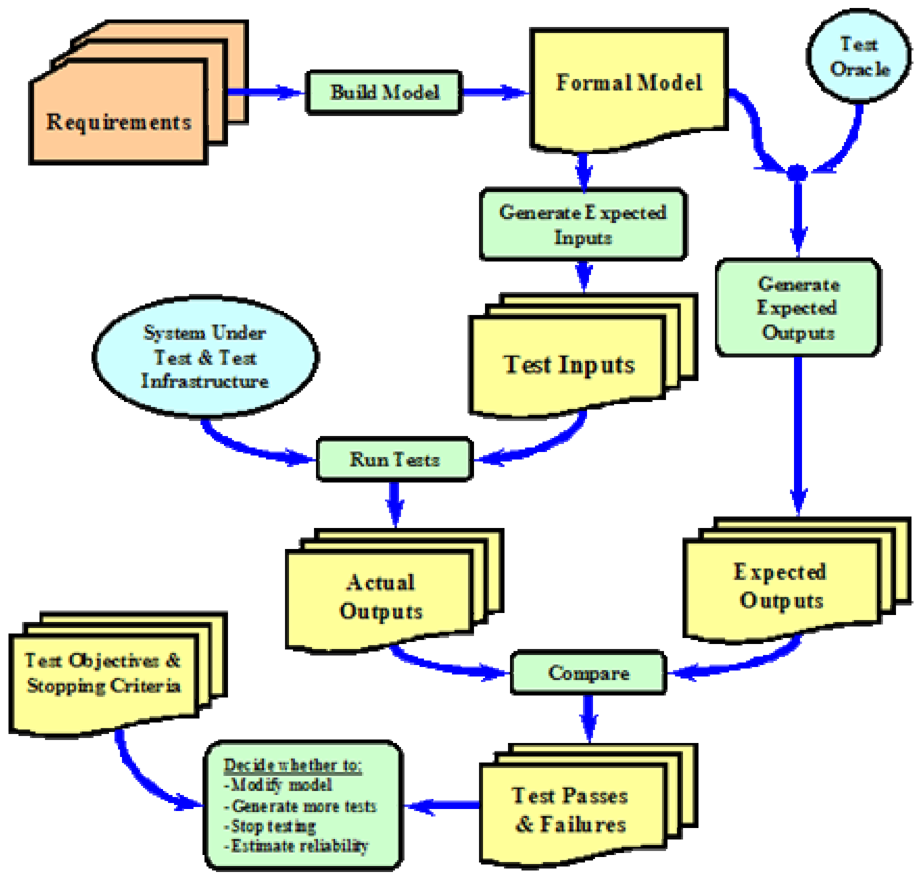

Figure 1. Model-based Testing Activities (PRASANNA *et al.*, 2005)

The MBT strategy usually includes different levels of abstraction, a behavior model, the relationship between models and code, test case generation technology, the importance of selection criteria for test cases, and a discussion of what can be and cannot be automated when it comes to testing (PRETSCHNER, 2005).

Some Model-based Testing approaches have been proposed in the technical literature. They use a lot of different software models, describing different software characteristics, to specify the software. It makes the identification and using of MBT approaches into software development processes a hard task, mainly considering that most of the software projects are using the UML as basic notation for the models. Therefore, the purpose of this work is to identify and characterize the available MBT approaches by accomplishing a secondary study (systematic review). After this, it is expected to be possible to describe Model-based Testing state-of-art, revealing the different perspectives used in the field and highlighting some perspectives for future research regarding model based testing.

2 Systematic Review Protocol regarding Model-Based Testing

Most of the scientific research has begun by an *ad hoc* literature review. However, if this review doesn't be complete and fair, it does not have some scientific value. This is the main reason to apply a systematic review. Systematic review is a method to identify, evaluate and interpret the pertinent research about a particular research question (KITCHENHAM, 2004). Moreover, there are other specific reasons that justify the use of systematic review:

- To summarize evidences about a specific theory or technology;
- To identify gaps in certain research areas, contributing for identification of investigations areas in the research field;
- To provide grounds for new research activities.

Into the context of this work, the main purposes on the execution of the systematic review execution are to provide grounds for new research activities and to resume the evidences regarding Model-based Software Testing. Therefore, it has been defined a review protocol to guide the literature review execution. This review protocol is described from section 2.1 to section 2.9. Further information regarding systematic reviews, its process and template can be found in (BIOLCHINI *et al*., 2005).

2.1 Goal of this Work

To accomplish a systematic review with the purpose of ***characterize*** the model-based software testing approaches usually used in software development projects described in the technical literature.

2.2 Research Question

2.2.1 **P0:** What Model-based Software Testing Approaches have been described in the technical literature and what their main characteristics?

- **Population:** Research on Software Testing.
- **Intervention:** Model-based Software Testing Approaches.
- **Comparison:** not applied.
- **Effects:** Characterization of Model-based Software Testing Approaches.
- **Problem:** To identify Model-based Software Testing Approaches and their characteristics.
- **Application:** Scientific research on Model-based Software Testing Area.

2.3 Selection of Sources

The search must be accomplished in digital library or technical databases. Must be analyzed only the papers published after 1990. This year has been identified because it landmarks the publication of UML. Although we search approaches that use different types of software models for test case generation, we will be focusing, after this systematic review, on approaches that use UML models to extract test cases.

2.3.1 Keywords

- **Population:** Software;
- **Intervention:** Model based Testing, Model driven Testing, UML based Testing, UML driven Testing, Requirement based Testing, Requirement driven Testing, Finite State Machine based Testing, Specification Based Testing, Specification Driven Testing;
- **Outcome:** approach, method, methodology, technique.

2.3.2 Language of Primary Studies

English.

2.3.3 Identification of Source

Methods of Source Selection:

The sources of studies were accessed by *web*. In the context of this systematic review didn't applied manual search.

Listing of Sources:

- IEEEXplorer
- ACM Digital Library
- Compendex EI
- INSPEC
- Web of Science
- The Software Quality Engineering Laboratory (SQUALL): Technical Reports (Carleton University): http://squall.sce.carleton.ca/pubs_tech_rep.html. Will be used the following technical reports:
    - A State-based Approach to Integration Testing for Object-Oriented Programs
    - A UML-Based Approach to System Testing.
    - Improving State-Based Coverage Criteria Using Data Flow Information.
    - Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles.
    - Towards Automated Support for Deriving Test Data from UML Statecharts.
- Online Papers About Model-Based Testing available at: http://www.geocities.com/model_based_testing/online_papers.htm (September 18, 2006). Will be used the following papers:
    - ABDURAZIK, A.; OFFUTT, J.; "Using UML Collaboration Diagrams for Static Checking and Test Generation"; UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings, Springer, 2000, 1939, 383-3.
    - ANDREWS, A.; OFFUTT, J.; ALEXANDER., R.; "Testing web applications by modeling with FSMs"; Software Systems and Modeling, 2005, 4(2).
    - MEYER, Steve; SANDFOSS, Ray; "Applying Use-Case Methodology to SRE and System Testing"; STAR West Conference, Oct. 1998, pp. 1-16.
- Kamran Ghani's Website (PhD Student at York University). Some papers related to Model-Based Testing available at: http://www-users.cs.york.ac.uk/~kamran/papers.htm#R2 (September 18, 2006). Will be used the following papers:

  - SCHEETZ, M.; von MAYHAUSER, A.; FRANCE, R.; DAHLMAN, E.; HOWE, A.E.; “Generating Test Cases from an OO Model with an AI Planning System”, *ISSRE '99: Proceedings of the 10th International Symposium on Software Reliability Engineering, IEEE Computer Society,* 1999, 250.
  - OFFUTT, Jeff; LIU, Shaoying; ABDURAZIK, Aynur, AMMANN, Paul; “Generating Test Data from State-Based Specifications”, Journal of Software Testing, Verification and Reliability, John Wiley & Sons, Ltd, No.13, 2003, pp. 25-53.
  - CRICHTON, Charles; CAVARRA, Alessandra; DAVIES, Jim; “Using UML for Automatic Test Generation” Proceedings of Automated Software Engineering (ASE), 2001.
- Informatik 2006 - http://www.informatik2006.de/272.html
  - NEBUT, C., FLEUREY, F.; “Automatic Test Generation: A Use Case Driven Approach”, IEEE Trans. Software Engineering, IEEE Press, 2006, 32, 140-155 (quoted by one paper).
  - BERNARD, E., BOUQUET, F., CHARBONNIER, A., LEGEARD, B., PEUREUX, F., UTTING, M., TORREBORRE, E.; “Model-Based Testing from UML Models”, In proceedings: Informatik, 2006 (printed copy).
  - ERNITS, J., KULL, A., RAINED, K., VAIN, J.; “Generating Test Sequences from UML Sequence Diagrams and State Diagrams”, In proceeding: Informatik, 2006 (printed copy).

2.3.4 Type of Primary Studies

Industrial Experience Report, Theoretical, Proof of Concept, Empirical Studies, and Experimental Studies.

2.3.5 Primary Studies Inclusion and Exclusion Criteria

- The papers must be available in the *Web (*digital libraries or technical databases);
- The papers must be written in English;
- The papers must describe model-based software testing approaches.
- The papers must have published from 1990.

After the definition of these criteria, the identified papers must be classified using one of the following categories:

[A] Papers describing Model-based software Testing Approaches regarding models extracted from software requirements (Functional Testing). These models **must use UML diagrams** as the mechanism to describe the software behavior.

[B] Papers describing Model-based software Testing Approaches regarding models extracted from software requirements (Functional Testing). These models **must NOT use UML diagrams** as the mechanism to describe the software behavior.

[C] Papers describing Model-based software Testing Approaches regarding models extracted from the software internal structure (Structural Testing based on architecture, components, interfaces, units). These models **must use UML diagrams** as the mechanism to describe the software internal structure.

[D] Papers describing Model-based software Testing Approaches regarding models extracted from the software internal structure (Structural Testing based on architecture, components, interfaces, units). These models **must NOT use UML diagrams** as the mechanism to describe the software internal structure.

[E] Any other papers that has been collected during the protocol execution but not related to the description of a MBT approach.

The Figure 2 describes the categorization of papers from the defined inclusion/exclusion criteria.

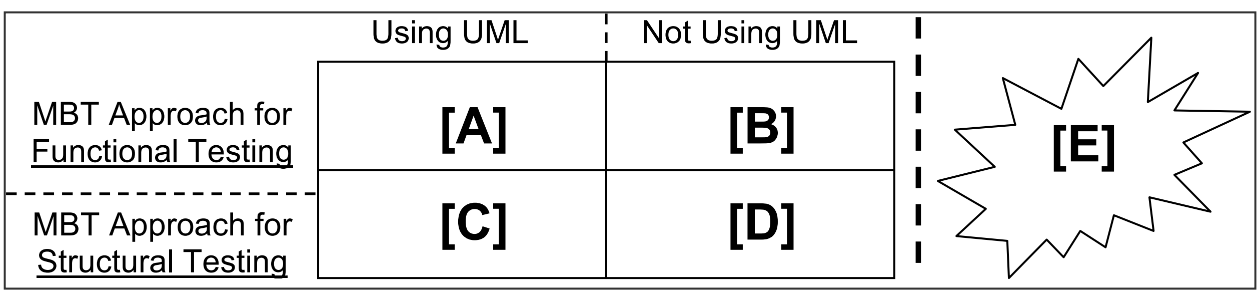

Figure 2. **Papers Categorization**

After the characterization process, the papers categorized within the [A] or [C] options (MBT Approaches using UML diagrams) will be the Priority Level 1, the subset of papers categorized within the [B] or [D] options will be the Priority Level 2, and the remained papers will compose the Priority Level 3. These papers must be selected for future reading and analysis. Otherwise, the paper categorized within the [E] option must be excluded. The definition of the Priority Levels will be done during this work.

2.3.6 Process to Select Primary Studies

One researcher will apply the search strategy to identify the primary studies. The identified papers will be filtered by the researcher that must firstly read the abstract to produce an initial categorization of the paper. After that, the full text must be read, checking whether the inclusion/exclusion criteria have been satisfied. In case of any conflict, a second researcher will make the verification. Besides all the papers have been read and categorized, a third researcher will double check the selections. After the process, the researchers must reach an agreement about what papers shall be selected per each category.

2.4 Primary Studies Quality Evaluation

We do not have defined any checklist to evaluate the quality of primary studies. The strategy is to consider the quality concerned with the sources selection of these studies and the application of the inclusion/exclusion criteria by the researchers.

2.5 Information Extraction Strategy

For each selected MBT approach after the filtering process, the researcher must extract the following information from the paper:

- Title;
- Authors;
- Source;
- Abstract;
- Type of Study: Industrial Experience Report, Theoretical, Proof of Concept, Empirical Study or Experimental Study (controlled or *quasi*-experiment).
- Category: [A], [B], [C], [D] or [E];
- MBT approach Description;

- Model used to describe the software behavior;
- Testing Level: acceptance, system (functional, behavior, configuration, security, stress, performance testing), integration, unit and regression (ROCHA et al., 2001).
- Software Domain: Execution Platform or Software Development Paradigm: Embedded System, Real-time System, Web Application, Client-server System, Object-Oriented Software, Model-based Development, etc.
- Testing Coverage Criteria;
- Test Cases Generation Criteria;
- Testing Design and Generation Technology;
- Tools: Tools used to testing generation and automation;
- Proportion of Automated Steps (X/Y): Number of automated steps (X) per Number of Steps;
- Complexity of each non-automated steps: Necessary (Initial Behavior Model Building); Low (Only choices of a specific criteria or option); Medium (Intermediate Model Building or Test Data Definition); High (Translation from on model into other model or other task); or Not Applied (if approach doesn't describe steps or if there is not non-automated steps);
- Intermediate Model used between behavior model and test cases;
- Non-functional requirement used to generate test cases: if applied, must be defined the category of non-functional requirement according ISO 9126 quality attributes;
- Inputs used to generate test cases and the complexity for interpretation them;
- Output generated by the approach and the complexity for interpretation them;
- External software used to support any steps of the approach (modeling, generation, or execution);
- Limitation of the approach: what are the limitations of the approach? What task is it not able to do? Ex: only OO software developed using the JAVA language, or the test cases are generated from a specific and particular model developed during the work (external people cannot use it).

- Restriction to use the approach: what are the restrictions to use the approach on field? For example, there is no tool to support, or the algorithm used to generate test cases doesn't allow a specific type of diagrams, or some other constraints.

The information must be organized in accordance of the Table 1:

Table 1. Fields used to characterize one MBT approach

Title
Authors
Source
Abstract
Type of Study
Category
Approach Description
Behavior Model
Testing Level
Software Domain
Tests Coverage Criteria
Test Cases Generation Criteria
Tests Design and Generation Technology
Tools to Test Case Generation or Execution
Automated Steps
Complexity of each non-automated steps
Intermediate Model
Non-functional requirements used to generate test cases
Inputs
Complexity of inputs interpretation
Outputs
Complexity of outputs interpretation
External software
Limitation of the approach
Restriction to use the approach on the field

The data must be recorded using the JabRef Tool (References Manager Tool) (http://jabref.sourceforge.net/). This tool:

- Imports papers references from all digital libraries used in this work after the search execution (in BibTex format);
- Allows the creation of fields to help the characterization of papers;
- Allows the creation of reports to export information about papers;
- Makes easy the citation of papers during papers writing in MS Word or LaTex
- Allows the filtering of papers to extract some information from them;

In this work, it has been used to (1) support the papers extraction from digital libraries, and (2) Support the papers characterization (several fields have been created). A captured screen of JabRef is presented on Figure 3.

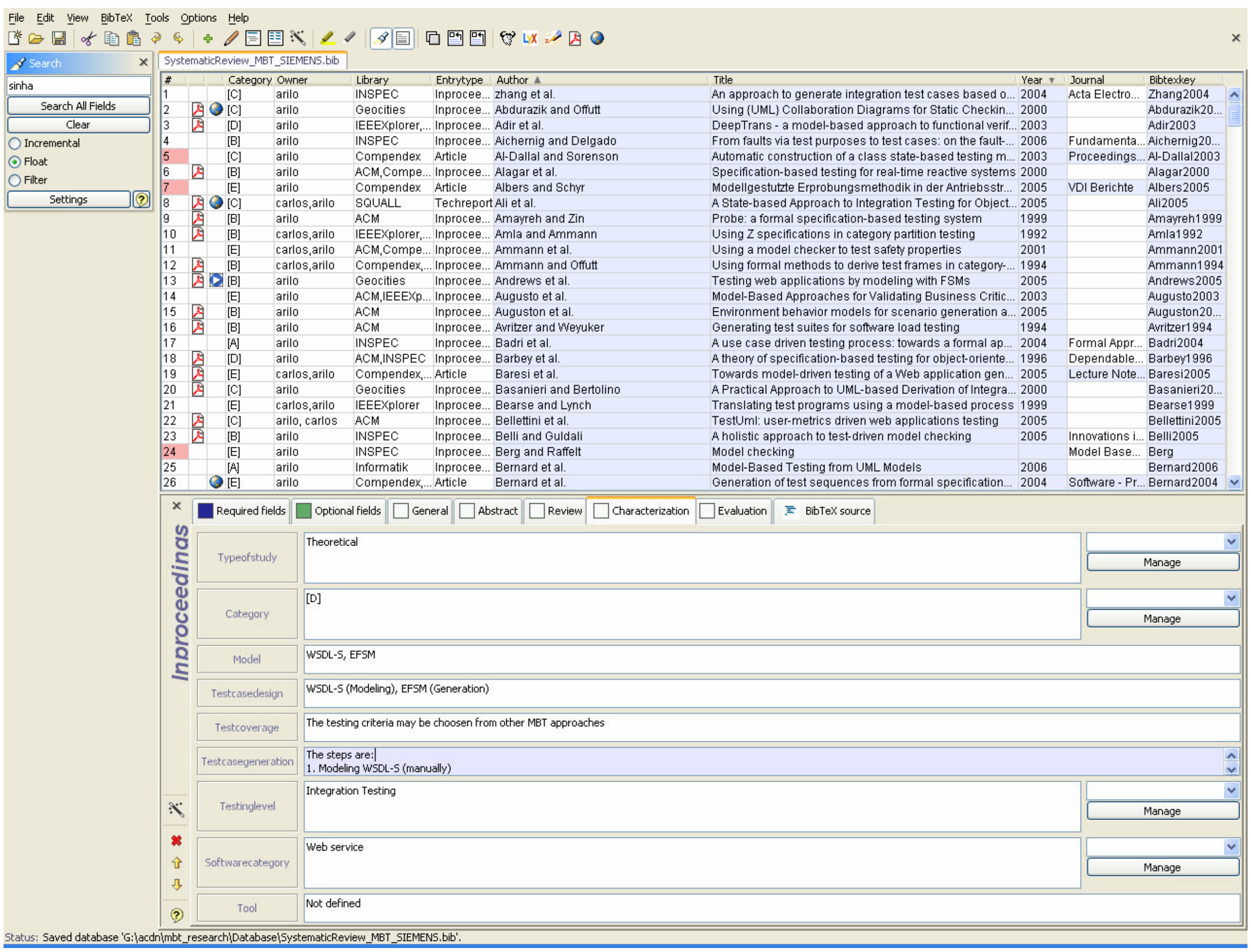

Figure 3. Main Screen on JabRef Tool

2.6 Summary of the Results

The results must be organized in tables. Must be accomplished analysis for:

- Identifying the MBT approaches in different testing level.
- Observing different models or languages used by the MBT approaches for test cases generation regarding the testing level;
- Identifying which approaches use a Intermediate Model for test cases generation, since this may represents more cost for the testing process;
- Observing what software domains that MBT approaches are more applied;
- Observing the way the MBT approaches define testing coverage and test case generation criteria (see definition of these terms in section 1) considering the used models;
- Observing how MBT approaches support the execution of the generated test cases (if the tasks are executed manually or automatically, for example), and;
- Observing positive and negative aspects in the MBT approaches;
- Identifying future research perspectives regarding model-based software testing.

2.7 Search

Some search strings have been defined for the identification of primary studies on the selected sources described in section 2.3.3. For more search precision, it is necessary to fit the strings in accordance with the syntax of each used search machine in this systematic review.

To evaluate the results of the search strings, the following set of papers can be used as control, because they represent important information concerned with MBT research:

(LUND and STØLEN, 2006), (HARTMANN *et al.*, 2004), (LINZHANG, 2004), (PRETSCHNER *et al.*, 2004), (BERTOLINO and GNESI, 2003), (BEYER *et al.*, 2003), (TAHAT *et al.*, 2001), (HARTMANN *et al.*, 2000), (DALAL *et al.*, 1999)

2.7.1 Search Strings

P0: IEEEXplorer

- (approach or method or methodology or technique) and ((specification based test) or (specification driven test) or (model based test) or (model driven test) or (use case based test) or (use case driven test) or (uml based test) or (uml driven test) or (requirement based test) or (requirement driven test) or (finite state machine based test) or (finite state machine driven test)) and (software) and (pyr >= 1990 and pyr<=2006)

PS: We could observe that if we use the term “test” or “testing” the output is not different for this search engine.

P0: Compendex EI

- (approach or method or methodology or technique) and (("model based test") or ("model based testing") or ("model driven test") or ("model driven testing") or ("specification based test") or ("specification based testing") or ("specification driven test") or ("specification driven testing") or ("use case based test") or ("use case based testing") or ("use case driven test") or ("use case driven testing") or ("uml based test") or ("uml based testing") or ("uml driven test") or ("uml driven testing") or ("requirement based test") or ("requirement based testing") or ("requirement driven test") or ("requirement driven testing") or ("finite state machine based test") or ("finite state machine based testing") or ("finite state machine driven test") or ("finite state machine driven testing")) and (software)

PS: This search engine has a field to determine the range of year to run the search. The range defined has been 1990 until 2006.

P0: INSPEC

- (approach or method or methodology or technique) and (("model based test") or ("model based testing") or ("model driven test") or ("model driven testing") or ("specification based test") or ("specification based testing") or ("specification driven test") or ("specification driven testing") or ("use case based test") or ("use case based testing") or ("use case driven test") or ("use case driven testing") or ("uml based test") or ("uml based testing") or ("uml driven test") or ("uml driven testing") or ("requirement based test") or ("requirement based testing") or ("requirement driven test") or ("requirement driven testing") or ("finite state machine based test") or ("finite state machine based testing") or ("finite state machine driven test") or ("finite state machine driven testing")) and (software)

PS: This search engine has databases to range of years. The databases selected were from 1990 until 2006.

P0: ACM Digital Library

- +“model based test” (it is equal +“model based testing”) +software
- +“model driven test” (it is equal “model driven testing”) +software
- +“uml based test” (it is equal “uml based testing”) + software
- +“uml driven test” (it is equal “uml driven testing”) +software
- +“use case based test” (it is equal “use case based testing”) + software
- +“use case driven test” (it is equal “use case driven testing”) +software
- +“requirement based test” (it is equal “requirement based testing”) +software
- +“requirement driven test” (it is equal “requirement driven testing”) +software
- +“specification based test” (it is equal “specification based testing”) +software
- +“specification driven test” (it is equal “specification driven testing”) +software
- +“finite state machine based test” (it is equal “finite state machine based testing”) +software
- +“finite state machine driven test” (it is equal “finite state machine driven testing”) +software

PS1: This search engine doesn’t have search fields to determine the range of years. The selection of the papers by date must be done manually.

PS2: This search engine doesn't have mechanisms to save the results of the search. Therefore, it has been accomplished a manual filtering and after this the results were inserted in the JabRef Tool.

P0: Web of Science

- TS=(("model based test" OR "model based testing" OR "model driven test" OR "model driven testing" OR "specification based test" OR "specification based testing" OR "specification driven test" OR "specification driven testing" OR "use case based test" OR "use case based testing" OR "use case driven test" OR "use case driven testing" OR "uml based test" OR "uml based testing" OR "uml driven test" OR "uml driven testing" OR "finite state machine based test" OR "finite state machine based testing" OR "finite state machine driven test" OR "finite state machine driven testing") AND (software) AND (approach OR method OR methodology OR technique))

PS1: This search engine has a field to determine the range of year to run the search. The range defined was 1990 until 2006.

After the searches using each one of the digital libraries, the following number of papers has been found (Table 2, Figure 4, and Figure 5):

Table 2. Classification of Identified Papers

Digital Library	Amount of Papers	Classification Criteria						Selected Papers		
		[A]	[B]	[C]	[D]	[E]	NC	1 PL	2 PL	3 PL
IEEEXplorer	89	4	37	3	14	31	0	7	10	41
COMPENDEX EI	100	4	29	3	16	36	12	7	12	33
INSPEC	181	7	48	3	29	55	41	10	15	60
ACM	221	12	58	3	39	92	17	15	23	74
Web of Science	33	1	8	0	3	12	9	1	2	9
SQUALL	5	1	0	4	0	0	0	5	0	0
GEOCITIES Website	4	1	1	2	0	0	0	3	1	0
Kamran Ghani's Website	3	1	0	2	0	0	0	3	0	0
Informatik 2006	3	2	0	1	0	0	0	3	0	0
After filtering (removing repeated papers)	**406**	**26**	**93**	**21**	**62**	**147**	**57**	47	31	124

NC = Not classified. The papers were not found or they are not available.
PL = Priority Level.

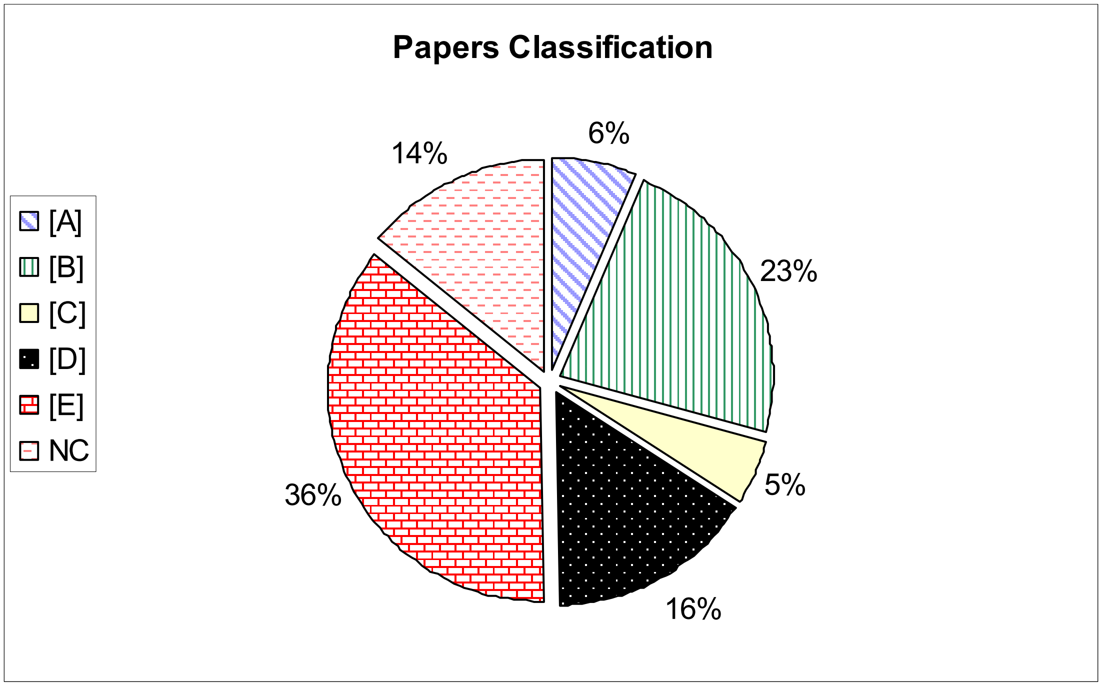

Figure 4. **The papers classification in accordance with the defined categories.**

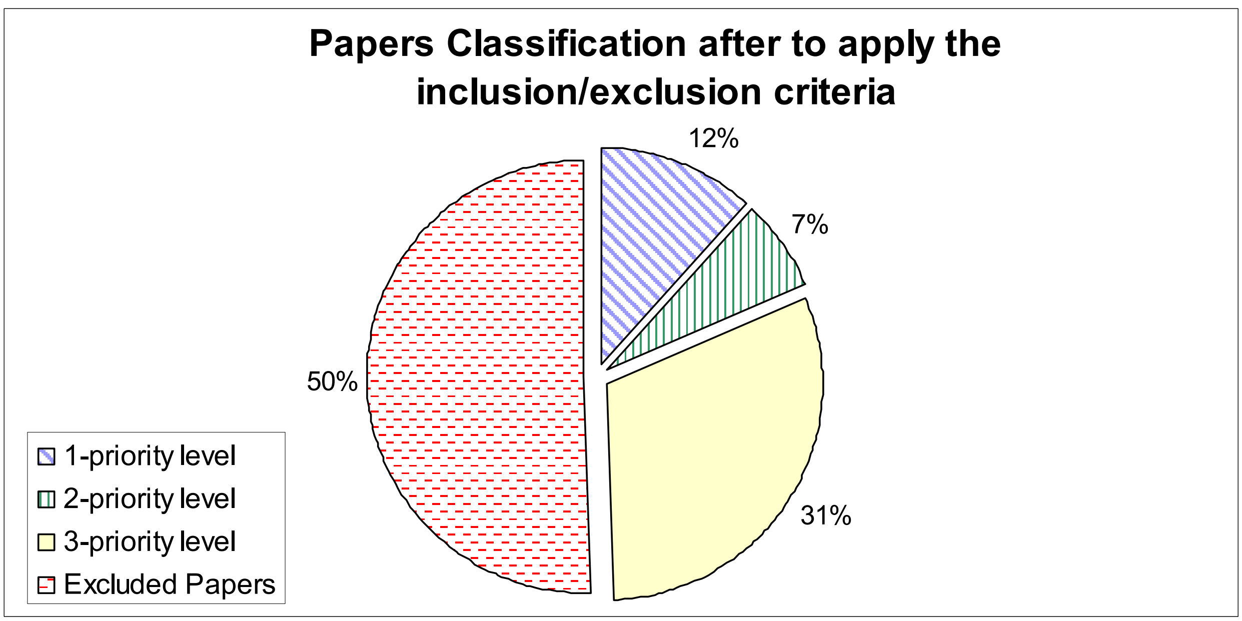

Figure 5. The papers classification after to apply the inclusion/exclusion criteria.

The list with all papers identified in this systematic review is presented in the Appendix A.

2.8 Execution

After the developing of the search protocol, the systematic review was executed. The execution has been accomplished for all identified research string in each digital library, but the results have been analyzed as a whole, because one paper could be found in different digital libraries.

A total of 406 papers have been found. A researcher began the filtering process. During this process, 204 papers have been excluded (because either their scope was not related to this work, or they were repeated, or they are not available). After to apply the inclusion/exclusion criteria, we selected 202 papers to read, extract and analyze their information: 47 papers in the 1-priority level group, 31 papers in the 2-priority level group, and 124 papers in the 3-priority level group. The description about each Priority Level is presented in the section 2.9 (“Analysis”).

2.9 Analysis

To analyze the identified papers, some steps will be accomplished:

1. The inclusion/exclusion criteria (section 2.3.5) will be applied;
    a. Only papers in the categories A, B, C, or D, and papers available to download will be analyzed. Papers in the category “E” or not available will be excluded of this research.
2. The selected papers have been separated in priority levels, according with some criteria defined during this work. These criteria will be presented in this section (section 2.9.1).
    a. Firstly, the papers in the Priority Level will be analyzed;
    b. Secondly, the papers in the Priority Level 2 will be analyzed;
    c. Finally, the papers in the Priority Level 3 will be analyzed;
    d. After that, every papers will be analyzed together;

3. For each priority level group, will be analyzed quantitative and qualitative information about the papers, according with the criteria defined and that will be showed in this section (section 2.9.2).

2.9.1 Priority Levels (PL)

During this work, three priority levels have been defined to support the papers analyses. The criteria used to define what papers compose each priority level group are:

- **PRIORITY LEVEL 1:** Only papers describing Model-based Testing approaches using UML, that is, every paper in the categories "A" and "C". The papers are (Table 3):

Table 3. Papers in the Priority Level 1.

Approach	Title	Authors
Abdurazik2000	Using UML Collaboration Diagrams for Static Checking and Test Generation	A. Abdurazik, J. Offutt
Ali2005	A State-based Approach to Integration Testing for Object-Oriented Programs	S. Ali, M.J. Rehman, L.C. Briand, H. Asghar, Z. Zafar, A. Nadeem
Basanieri2000	A Practical Approach to UML-based Derivation of Integration Tests	F. Basanieri, A. Bertolino
Bellettini2005	TestUml: user-metrics driven web applications testing	C. Bellettini, A. Marchetto, A. Trentini
Bernard2006	Model-Based Testing from UML Models	E. Bernard, F. Bouquet, A. Charbonnier, B. Legeard, F. Peureux, M. Utting, E. Torreborre
Bertolino2003	Integration of "components" to test software components	A. Bertolino, E. Marchetti, A. Polini
Bertolino2005	Introducing a Reasonably Complete and Coherent Approach for Model-based Testing	A. Bertolino, E. Marchetti, H. Muccini
Beyer2003	Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains	M. Beyer, W. Dulz, F. Zhen
Botaschanjan2004	Testing agile requirements models	J. Botaschanjan, M. Pister, B. Rumpe
Briand2002	A UML-Based Approach to System Testing	L.C. Briand, Y. Labiche
Briand2002a	Automating impact analysis and regression test selection based on UML designs	L.C. Briand, Y. Labiche, G. Soccar
Briand2002b	Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles	L.C. Briand, Y. Labiche, Y. Wang
Briand2004a	Towards Automated Support for Deriving Test Data from UML Statecharts	L.C. Briand, Y. Labiche, J. Cui
Briand2004b	Improving State-Based Coverage Criteria Using Data Flow Information	L.C. Briand, Y. Labiche, Q. Lin
Briand2006	Automated, contract-based user testing of commercial-off-the-shelf components	L.C. Briand, Y. Labiche, M. Sówka
Carpenter1999	Verification of requirements for safety-critical software	P. B. Carpenter
Cavarra2003	A method for the automatic generation of test suites from object models	A. Cavarra, C. Crichton, J. Davies

Chen2002	Specification-based regression test selection with risk analysis	Y. Chen, R. L. Probert, D. P. Sims
Chevalley2001	Automated Generation of Statistical Test Cases from UML State Diagrams	P. Chevalley and P. T. Fosse
Crichton2001	Using UML for Automatic Test Generation	C. Crichton, A. Cavarra, J. Davies
Deng2004	Model-based testing and maintenance	D. Deng, P.C. Sheu, T. Wang
Garousi2006	Traffic-aware stress testing of distributed systems based on UML models	V. Garousi, L. C. Briand, Y. Labiche
Gnesi2004	Formal test-case generation for UML statecharts	S. Gnesi, D. Latella, M. Massink
Gross2005	Model-Based Built-In Tests	H-G. Gross, I. Schieferdecker, G. Din
Hartman2004	The AGEDIS tools for model based testing	A. Hartman, K. Nagin
Hartmann2000	UML-based integration testing	J. Hartmann, C. Imoberdorf, M. Meisinger
Kansomkeat2003	Automated-generating test case using UML statechart diagrams	S. Kansomkeat, W. Rivepiboon
Kim99	Test cases generation from UML state diagrams	Y.G. Kim, H.S. Hong, D.H. Bae, S.D. Cha
Linzhang2004	Generating test cases from UML activity diagram based on Gray-box method	W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, Z. Guoliang
Liuying1999	Test selection from UML Statecharts	L. Liuying, Q. Zhichang
Lucio2005	A methodology and a framework for model-based testing	L. Lucio, L. Pedro, D. Buchs
Lund2006	Deriving tests from UML 2.0 sequence diagrams with neg and assert	M.S. Lund, K. Stølen
Meyer1998	Applying Use-Case Methodology to SRE and System Testing	S. Meyer, R. Sandfoss
Mingsong2006	Automatic test case generation for UML activity diagrams	C. Mingsong, Q. Xiaokang, L. Xuandong
Murthy2006	Test ready UML statechart models	P.V.R. Murthy, P.C. Anitha, M. Mahesh, R. Subramanyan
Nebut2006	Automatic Test Generation: A Use Case Driven Approach	C. Nebut, F. Fleurey
Offutt1999b	Generating tests from UML specifications	J. Offutt, A. Abdurazik
Offutt2003	Generating Test Data from State-Based Specifications	J. Offutt, S. Liu, A. Abdurazik, P. Ammann
Olimpiew2005	Model-based testing for applications derived from software product lines	E.M. Olimpiew, H. Gomaa
Riebisch2002	UML-based statistical test case generation	M. Riebisch, I. Philippow, M. Gotze
Rumpe2003	Model-based testing of object-oriented systems	B. Rumpe
Scheetz1999	Generating Test Cases from an OO Model with an AI Planning System	M. Scheetz, A. Mayrhauser, R. France, E. Dahlman, A.E. Howe
Sokenou2006	Generating Test Sequences from UML Sequence Diagrams and State Diagrams	D. Sokenou
Traore2003	A transition-based strategy for object-oriented software testing	I. Traoré
Vieira2006	Automation of GUI testing using a model-driven approach	M. Vieira, J. Leduc, B. Hasling,R.Subramanyan,

		J. Kazmeier
Wu2003	UML-Based Integration Testing for Component-Based Software	Y. Wu, M-H. Chen, J. Offutt
Zhen2004	Model-based testing with UML applied to a roaming algorithm for Bluetooth devices	R.D. Zhen, J. Grabowski, H. Neukirchen, H. Pals

- **PRIORITY LEVEL 2:** From each selected paper (A, B, C, or D) it's necessary to look for the papers in the categories "B" or "D" that are referenced by some selected paper. The citation number must be greater than 2 (>2) to compose the PL 2 (22 papers), it's necessary to build a tree connecting every papers referenced among them (Figure 6). Each node is a paper, and an arrow indicates that the paper in the arrow's beginning references the paper in the arrow's end. The three is described by a UML Activity Diagram, and the swim lanes represent the years (from 1990 until 2006). Other 9 papers published between 2004 and 2006 will be included subjectively, because these papers cannot have an expressive number of citations. The papers that compose the PL 2 are at total 31 papers, but some of them describe repeated approach, then just one paper by approach has been analyzed (Table 4):

Table 4. Papers in the Priority Level 2.

Approach	Title	Authors
Ammann1994 (Amla1992)	Using formal methods to derive test frames in category-partition testing (Using Z specifications in category partition testing)	P. Ammann, J. Offutt (N. Amla, P. Ammann)
Andrews2005	Testing web applications by modeling with FSMs	A. Andrews, J. Offutt, R. Alexander
Barbey1996	A theory of specification-based testing for object-oriented software	S. Barbey, D. Buchs, C. Péraire
Bousquet1999a	Lutess: a specification-driven testing environment for synchronous software	L. du Bousquet, F. Ouabdesselam, J.-L. Richier, N. Zuanon
Chang1999 (Chang1996)	Structural specification-based testing: automated support and experimental evaluation (Structural specification-based testing with ADL)	J. Chang, D. J. Richardson (J. Chang, D.J. Richardson, S. Sankar)
Chen2005	An approach to integration testing based on data flow specifications	Y. Chen, S. Liu, F. Nagoya
Dalal1999 (Dalal1998)	Model-based testing in practice Model-based testing of a highly programmable system	S.R. Dalal, A. Jain, N. Karunanithi, J.M Leaton, C.M. Lott,

		G.C Patton, B.M Horowitz (S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott)
Friedman2002	Projected state machine coverage for software testing	G. Friedman, A. Hartman, K. Nagin, T. Shiran
Gargantini1999	Using model checking to generate tests from requirements specifications	A. Gargantini, C. Heitmeyer
Hong2000	A test sequence selection method for statecharts	H.S. Hong, Y.G. Kim, S.D. Cha, D.H. Bae, H. Ural
Legeard2004	Controlling test case explosion in test generation from B formal models	B. Legeard, F. Peureux, M. Utting
Mandrioli1995	Generating test cases for real-time systems from logic specifications	D. Mandrioli, S. Morasca, A. Morzenti
Offut1999a	Generating test data from SOFL specifications	J. Offutt, S. Liu
Paradkar2004a	Plannable test selection criteria for FSMs extracted from operational specifications	A. Paradkar
Parissis1996	Specification-based testing of synchronous software	I. Parissis, F. Ouabdesselam
Pretschner2001 (or Pretschner2001a)	Model based testing in evolutionary software development (Model-based testing for real : the inhouse card case study)	A. Pretschner, H. Lotzbeyer, J. Philipps (A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel)
Richardson1992	Specification-based test oracles for reactive systems	D.J. Richardson, S.L. Aha, T.O. O'Malley
Richardson1996	Software testing at the architectural level	D.J. Richardson, A.L. Wolf
Satpathy2005	ProTest: An Automatic Test Environment for B Specification	M. Satpathy, M. Leuschel, M. Butler
Sinha2006a	Model-based functional conformance testing of web services operating on persistent data	A. Sinha, A. Paradkar
Stobie2005a	Model Based Testing in Practice at Microsoft	K. Stobie
Stocks1993 (or Stocks1993a or Stocks1996)	Test template framework: a specification-based testing case study (Test templates: a specification-based testing framework or A Framework for Specification-Based Testing)	P. Stocks, D. Carrington
Tahat2001	Requirement-based automated black-box test generation	L.H. Tahat, B. Vaysburg, B. Korel, A.J. Bader
Tan2004	Specification-based testing with linear temporal logic	L. Tan, O. Sokolsky, I. Lee
Xu2006a	State-based testing of integration aspects	W. Xu, D. Xu

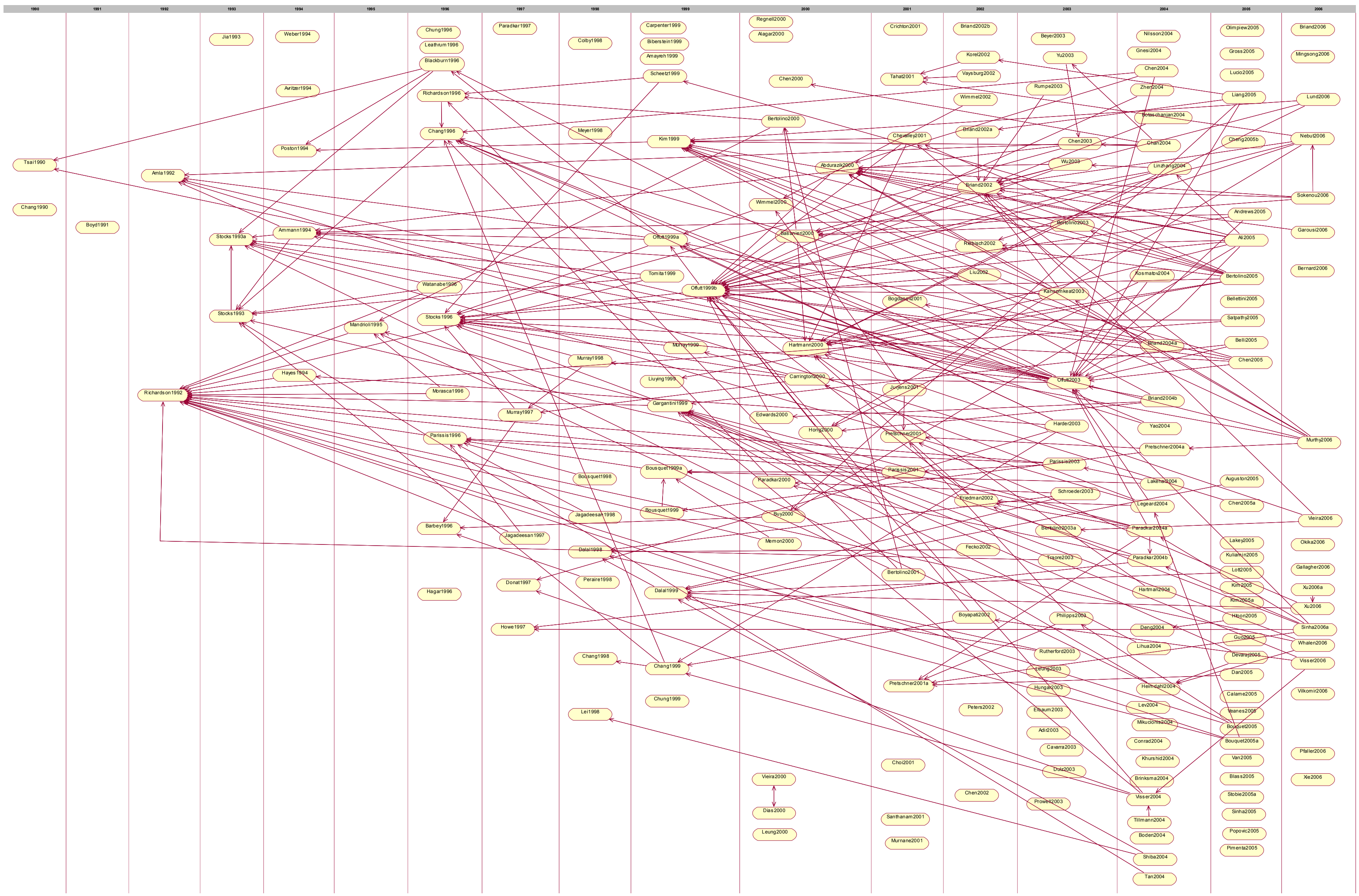

Figure 6. **Tree describing the relation among the papers A, B, C and D.**

- **PRIORITY LEVEL 3:** Every papers in the category "B" or "D" that are not included in the PL 2. At total, 124 papers are included on the Priority Level 3 (because 11 papers are repeated in the PL2, but will be counted as just 5).

2.9.2 Quantitative and Qualitative Analysis

The analysis will be accomplished in two ways: **(1) Quantitative Analysis**, describing statistical information about the selected papers (for example: How many approaches use a specific model to describe the software behavior?); and **(2) Qualitative Analysis**, describing subjective information extracted from each or a group of selected papers about specific characteristics (for example: After the characterization, the approaches X and Y have some specifics positive aspects that make easy to apply them on the field, but have also some negative aspects that can make difficult to use them in some organizations).

- **Quantitative Analysis:**

To describe the quantitative results, some information must be extracted from the papers analysis. This information can be related to:

- Testing Level that the approaches are applied;
- Software Categories that the approaches are applied;
- Level of automation of the approaches;
- Models used to described the software behavior;
- Usage of Tools to support the steps;
- Usage of Intermediate model during the test case generation process;
- Usage of Non-functional requirement to describe the software behavior;
- Visibility of the approach in the Model-based Testing community;

These data may be crossed to define specific scenarios about research regarding Model-based Testing approaches described on the technical literature. This information may present scenarios on Software Engineering not covered by Model-based Testing approaches.

- **Qualitative Analysis:**

To describe the qualitative results, some subjective information must be extracted the papers analysis regarding specific characteristics about one or a group of approaches. This information can be related to:

- Complexity of Cost and effort to apply the approach on the field;
- Skills required to apply the approach on the field;
- Inputs or pre-requirements necessaries to use the approach;
- Limitations (application context, testing level, tools);
- Quality of the outputs generated by the approach

To accomplish the qualitative analysis, just a subset of papers may be used. These papers are selected after the papers characterization by the researcher using two criteria: (1) Number of times that the paper was referenced by other papers in the set of identified papers; (2) Decision of the researcher about the quality of the paper;

3 Results Analysis

The analysis has been accomplished at two levels: Quantitative and Qualitative, for paper in Priority Levels 1 and 2. The papers in the Priority Level 3 haven't been analyzed yet.

3.1 Quantitative Analysis

According to Dalal (1998), Model-based Testing depends on three key elements: the <u>model</u> used for the software behavior description, the <u>test-generation algorithm</u> (criteria), and the <u>tools</u> that generate supporting infrastructure for the tests (including expected outputs). In this section, we discuss about these and other characteristics that influence MBT approaches' quality, like testing level, software domain, and automated steps.

3.1.1 Testing Level

MBT approaches have been characterized based on the testing level that they are applied. This information is important to show the scope in that researches regarding MBT approaches have been developed.

Tests may be applied to very small units (Unit Testing), collections of units (Integration Testing), or entire systems (System Testing – functional and non-functional). MBT can assist test activities at all levels and in different ways. Moreover, Regression Testing may be accomplished easily, because if the software changes, the model (or specification) also changes and the new test cases can be generated from the automatic generation process.

Therefore, these approaches have been characterized based on:

- The use of intermediate model to generate test cases. Intermediate models are models generated from a model describing the software (that is used as input by a MBT approach) to generate test cases. Usually, intermediate model reduces the complexity of the original model, by making it easy to use an algorithm to generate test case. Intermediate model also increases the cost of applying a particular MBT approach, since sometimes it is maybe necessary to translate manually from one behavioral model into an intermediate model.

- The use of tool to support some step in the test case generation process. Tool may reduce cost and time to generate test cases, since some steps may be automated, and to make it easier to use the approach.

Table 5 describes the results about these characteristics:

Table 5. Results based on the Testing Level Analysis

Testing Level	# approaches	Intermediate Model		Support tool	
		Use	Not use	Use	Not cited
System	48	40%	60%	66%	34%
Integration	17	64.7%	35.3%	41.2%	58.8%
Unit/Component	8	50%	50%	62.5%	37.5%
Regression	4	25%	75%	50%	50%
TOTAL	72 (*)	45.6%	54.4%	59.5%	40.5%
(*) some approaches are applied for more than one testing level					

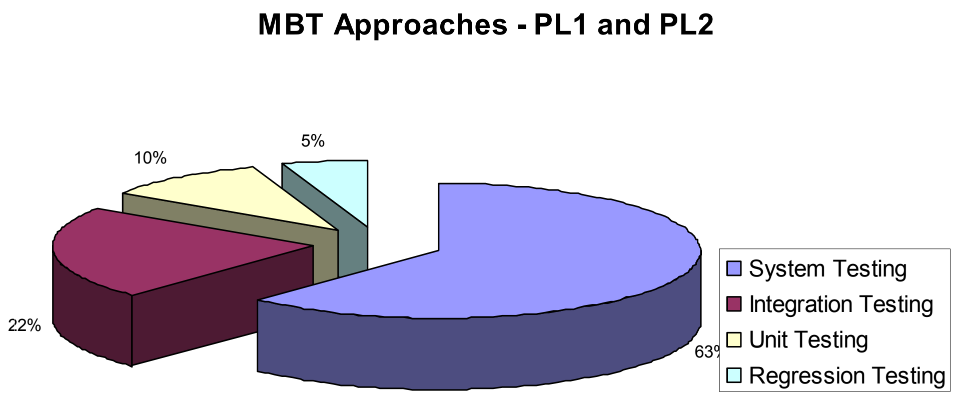

Figure 7. Analysis of MBT approaches by Testing Levels

MBT was originally intended for System Level Testing. This aspect makes high the number of MBT approaches applied for this testing level, as showing in Figure 7. 63% of approaches surveyed were used for System Level Testing. Examples of approaches: (ABDURAZIK and OFFUTT, 2000), (BRIAND and LABICHE, 2002), (MINGSONG *et al.*, 2006), (OFFUTT and ABDURAZIK, 1999), (OFFUTT *et al.*, 2003), and (STOBIE, 2005). Subsequently, MBT approaches were applied for other levels, like Integration Testing (22%) [E.g.: (HARTMANN *et al.*, 2000), (BASANIERI and BERTOLINO, 2000), (WU *et* *al*., 2003)], Regression Testing (5%) and Unit Testing (10%) [E.g.: (CHANG *et al*., 1996), (KIM *et al*., 1999), (BRIAND *et al*., 2006)]. Unit testing is not a usual abstraction level for MBT. The purpose of Unit Testing is to tests small module, functions, or classes after the implementation. There are a lot of available approaches to support structural testing from the source code.

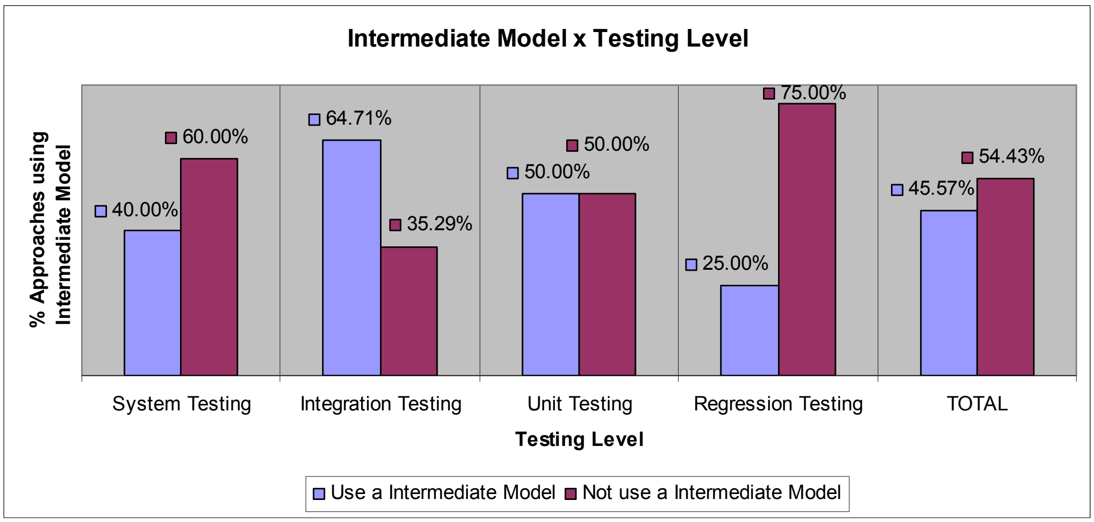

Figure 8. Analysis of MBT approaches using Intermediate Model by Testing Levels

Analyzing the usage of *Intermediate Model* (Figure 8), we observe that Intermediate Models are applied more for Integration Testing (64%) than System Testing (60%). One possible reason is that Design or Architectural Models usually don't describe control-flow or data-flow information. It is therefore necessary to translate this model into other model in order to extract paths and data to be used during the tests. In System Testing approaches, usually there is a model generated from the system functionalities that describe the steps to be followed and the data to be input.

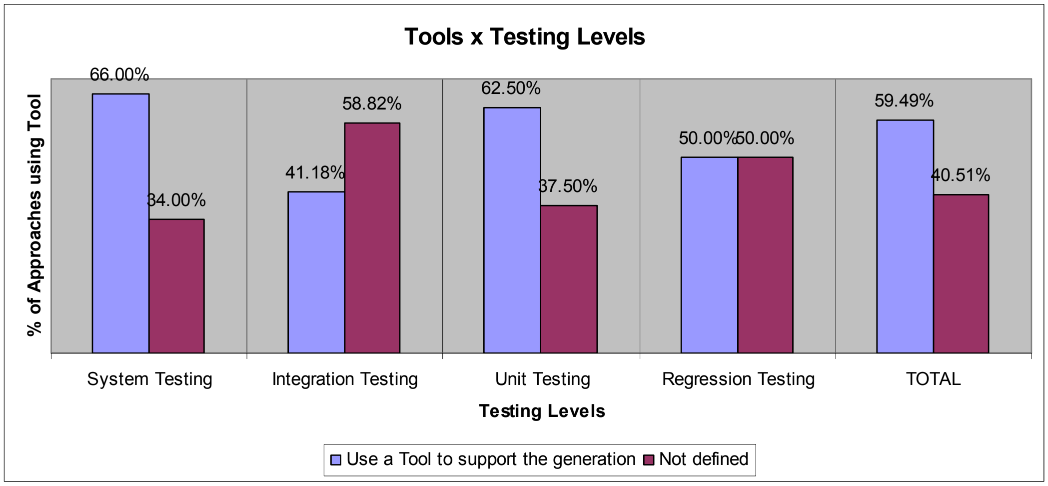

Figure 9. Analysis of MBT approaches using Tools by Testing Level

The estimation of the number of *tools to supporting test case generation process* is difficult, because there are a high number of tools under deployment. However, we observe that there are more MBT tools for System Testing than for Integration Testing (Figure 9). The reasons are unknown.

Several proprietary and freeware tools have been developed. To count the amount of MBT tools available is an unfeasible task. The URL www.opensourcetesting.org lists open-source testing tools for different tasks, including test case generation.

3.1.2 MBT Approaches by Software Domain

Usually, MBT approaches are applied for specific application domain. This defines the scope of each approach and when they can be used.

Table 6 lists the software domains and the number of approaches applied per testing level:

The majority of the selected approaches use UML, an *Object-oriented* modeling language. This information explains the high number of approaches applied for *Object-oriented Software*.

The majority of non-UML based MBT approaches are applied for *Reactive Systems* and *Safety-critical Systems*. Examples are (LEGEARD *et al*., 2004), (MANDRIOLI *et al*., 1995), (PARISSIS and OUABDESSELAM, 1996), (PRETSCHNER *et al*., 2001), and

(RICHARDSON *et al*., 1992). These application domains need high coverage and quality of software, and therefore need very good testing methods such as MBT.

Table 6. Number of MBT approaches for Software Domains x Testing Levels

Software Domains / Testing Levels		Testing Levels			
		System Testing	Integration Testing	Unit Testing	Regression Testing
Software Domain	Object-oriented software	17	10	4	2
	Not defined	13	3	2	0
	Safety-Critical Software	5	0	1	0
	Reactive systems	5	1	0	0
	Distributed Systems	2	0	0	1
	Web Application	2	0	0	0
	Component-based Software	0	2	0	0
	Embedded System	2	0	0	1
	Concurrent Program	1	0	0	0
	Aspect-oriented Software	0	1	0	0
	Java programs	1	0	0	0
	Software Product Line	1	0	0	0
	Web Service	0	1	0	0
	COTS	0	0	1	0
	Any domain	1	0	0	1

Moreover, MBT has not been used sufficiently for software domains like COTS, Product Line, Aspect-oriented Software, Web Services, and Web Application. Apply MBT to these software domains could be potential topics of future research. Potential questions are: what are the limitations of the current Model-based Integration Testing for Web Service (or other software domain not frequently applied for MBT) and how to avoid them? How to apply MBT for system testing in Web Applications (or other software domain not frequently applied for MBT)?

3.1.3 Non-functional Requirements

Software Requirements may be: Functional or Non-functional.

Functional Requirements define the scenarios and functionalities that the software must provide during its execution. Non-functional Requirements (NFR) are essential to define architectural aspects and constraints during the software development. Both need to be tested. Generation of test cases from non-functional requirements description is sometimes not performed. We describe bellow MBT approaches used to non-functional test case generation.

The categories of NFR have been defined using ISO 9126 (2001). This standard defines quality attributes for software in 6 groups (*Functionality*, *Reliability*, *Usability*, *Maintainability*, *Portability*, and *Efficiency*). However, *Functionality* is composed of other sub-categories (including functional requirements). Then, only the sub-category *Security* related to the group *Functionality* has been used to classify non-functional requirement in this work. According with ISO-9126 (2001), the groups are:

- Efficiency: *A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.*
- Reliability: *set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.*
- Usability: *A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.*
- Maintainability: *A set of attributes that bear on the effort needed to make specified modifications.*
- Portability: *A set of attributes that bear on the ability of software to be transferred from one environment to another.*
- Security: *An attribute that bear on the ability of software to maintain its behavior and integrity in specific situations or context defined during its development.*

Table 7. Number of approaches using NFR classified according with ISO 9126 (2001).

NFR (according with ISO 9126)	# Approaches		
	Priority Level 1	Priority Level 2	TOTAL
Efficiency	4	4	8
Usability	0	0	0
Reliability	1	0	1
Maintainability	0	0	0
Portability	0	0	0
Security	0	4	4

From Table 7, we observe that the majority of MBT approaches that address the evaluation of NFR are for *Efficiency*. At total, 8 approaches use efficiency description for testing: (OFFUTT and ABDURAZIK, 1999), (RICHARDSON *et al*., 1992), (RUMPE, 2003), (ZHEN *et al*., 2004), (TAN *et al*., 2004), (GAROUSI *et al*., 2006), (MANDRIOLI *et al*., 1995), and (PRETSCHNER *et al*. 2001a). All 8 approaches use information about the response-time (performance) during test case generation. Analyzing the usage of NFR description during the testing case generation for different software domains, we observe that *Efficiency* requirements are used by OO Software, Distributed System, Reactive System, Critical System, or Embedded Systems. Other software domains don't address NFR for test case generation.

Moreover, 4 approaches use *Security* requirements during test case generation, all of them in Priority Level 2: (RICHARDSON *et al*., 1992), (TAN *et al*., 2004), (PARISSIS and OUABDESSELAM, 1996), and (du BOUSQUET *et al*., 1999). These requirements are usually used by *Reactive Systems*. This software domain has well-defined approaches to described securities properties. However, these application domains have intersection, and specific software could be classified on all these domains together.

Table 8. Number of approaches using types of NFR by Software Domain

Domains x NFR Type	Efficiency	Usability	Reliability	Maintainability	Portability	Security
Object-oriented Software	3	0	0	0	0	0
Distributed System	1		1	0	0	0
Reactive System	1	0	0	0	0	3
Critical System	1	0	0	0	0	0
Embedded System	1	0	0	0	0	0
Not defined	1	0	0	0	0	1

Several types of NFR, like usability, maintainability, portability, have not been tested using MBT. These aspects could be explored in future research.

3.1.4 Models Used to Describe the Software Structure or Behavior

The model used to describe the software structure or behavior is an important element in the study of MBT. The model defines the limitation of each approach based on the information that it can represents about the software structure or behavior. Sometimes the model is used for a specific application domain and cannot be used in another domain.

Table 9 lists behavior models and the number of approaches using each model:

Table 9. Number of approaches for different Behavior Model

Behavior Model	# Approaches		
	Priority Level 1	Priority Level 2	TOTAL
Statechart Diagram	26	1	27
Class Diagram	19	0	19
Sequence Diagram	19	0	19
Use Case Diagram	11	0	11
OCL	11	0	11
(Extended) Finite State Machine	3	7	10
Activity Diagram	9	0	9
Collaboration Diagram	8	0	8
Object Diagram	7	0	7
Graph	6	1	7
Z Specification	0	4	4
UML 2.0 Test Profile	2	0	2
Markov Chain Usage Model	2	0	2
XML Model	1	1	2
Abstract State Machine	1	1	2
CDFD	0	2	2
LUSTRE Model	0	2	2
UML 2.0 Package Diagram	1	0	1
UML 2.0 Interaction Overview Diagram	1	0	1
UML 2.0 Context Diagram	1	0	1
TTCN-3	1	0	1
Transition Test Sequence	1	0	1
Test Model (graphs + constraints)	1	0	1
State COllaboration TEst Model	1	0	1
Operational Profile	1	0	1
IOLTS (input enabled labeled transition systems over i/o-pair)	1	0	1
Intermediate Format	1	0	1
Extended Context Free Grammar Model	1	0	1
CSPE Constraints	1	0	1
Feature model	1	0	1
Component-based software architecture model	1	0	1
Simulation Model	1	0	1

AETGSpec	0	1	1
SCR Specifications	0	1	1
TRIO Specification	0	1	1
DNF	0	1	1
S-Module	0	1	1
I-Module	0	1	1
SALT Model	0	1	1
System Structure Diagrams (SDDs), State Transition Diagram (STD), Message Sequence Charts (MSCs), Data Type Diagram (DTD)	0	1	1
Prolog Specification	0	1	1
SDL Model	0	1	1
Linear Temporal Logic (LTL)	0	1	1
CO-OPN/2 Specification	0	1	1
ADL Specification	0	1	1
Chemical Abstract Machine (CHAM)	0	1	1
WSDL-S	0	1	1
State Model in AOP	0	1	1

As said before, the majority of the selected approaches uses UML for test case generation, hence the high number of approaches applied for Object-oriented software. For MBT approaches using UML, 27 approaches use *Statechart* [(BRIAND *et al*., 2004b), (HARTMAN and NAGIN, 2004), (HARTMANN et al., 2000), (OFFUTT and ABDURAZIK, 1999), (OFFUTT *et al*., 2003), (TRAORE, 2004), and others papers], 19 approaches use *Class Diagram* [(BASINIERI and BERTOLINO, 2000), (BERTOLINO *et al*., 2003), (BRIAND and LABICHE, 2002), (BRIAND and LABICHE, 2006), (VIEIRA *et al*., 2006), and others papers], and 19 papers use *Sequence Diagram* [(BERTOLINO *et al*., 2003), (BRIAND and LABICHE, 2006), (RUMPE, 2003), (WU *et al*., 2003), and others papers]. These are the most used models.

For MBT approaches not using UML, 7 approaches use *Finite State Machine* [(ANDREWS *et al*., 2005), (KIM *et al*., 1999), PARADKAR, 2004), (STOBIE, 2005), (TAHAT *et al*., 2001), and others papers] and 4 approaches use *Z Specification* [(AMLA and AMMANN, 1992), (AMMANN and OFFUTT, 1994), (LEGEARD *et al*., 2004), (STOCKS and CARRINGTON, 1996)]. These are the most used models for this context.

There is not enough information to decide if one model is more useful than another model. This definition depends on a lot of variables (or characteristics), like the software domains, algorithm used for tests generation, and testing level. However, knowing the information is useful to filter approaches for comparison.

More information about Models used to describe the software structure and behavior and their limitations is described on the Qualitative Analysis section (3.2).

3.1.5 Coverage and Test Case Generation Criteria

The main differences among MBT approaches are the criteria used to define the testing coverage (subset of test cases generated from a specific model) and test generation (steps to be accomplished).

The **Testing Coverage Criteria** defines the subset of test cases generated by the approach. These criteria depend on the model used to describe the software behavior. They may also determine the limitation of the approach, and the type and format of test cases that can be generated. The types of criteria used are control-flow (like transforming one model in a graph), and data-flow (defining the possible inputs for a specific field).

Popular testing coverage criteria used in control-flow which are based on graph theory are: all-states (all-nodes), all-transactions, simple-paths (single-paths), all-paths, n-Path Coverage, all-transaction-pair, and loop-coverage. In data-flow, common testing criteria are: boundary value coverage, equivalence class, category-partition, one-value/All-values coverage, all-definition, all-use, and all def-use paths coverage.

Details on what approaches use data-flow or control-flow criteria are presented on the *Quantitative Analysis* section (3.2).

The **Test Case Generation Criteria** defines the steps to be performed to generate test cases from a software behavior model. Each approach has its specific steps with different complexity or automation level. The steps are: (1) Modeling of Software Behavior; (2) Applying Testing Coverage Criteria to select the set of test cases; (3) Generation of test cases and expected results; (4) Execution of tests and comparison of obtained results against the expected results.

In the ideal case, test cases are fully generated automatically from the software behavior model, there are no manual steps, thus reducing time, effort, cost and also increasing testing coverage. However, some approaches have intermediate non-automated steps in its testing process, with different complexity levels. The discussion about the complexity of non-automated steps is presented in the next section (3.1.6).

Other important issues regarding the steps in a test case generation process is the automatic support provided by tools. Tools make feasible the application of a MBT in a project, automating some steps and supporting the performing of non-automated steps and the integration of data between the steps. The discussion about the using of tool by MBT approaches was described previously on section 3.1.1.

3.1.6 Level of Automation

One of the more important aspects in a MBT approach is the level of automation of its tasks. Automation means less cost, time, and effort to generate test cases. Usually, the testing process of MBT approaches is composed of automated and non-automated steps. It is necessary to analyze the complexity of non-automated steps in a MBT approach. A single manual step in a significantly automated approach can be harder than several manual in other approach.

A non-automated step may have different complexity level depending upon modeling of software behavior (every approach need to execute this task), choice of testing criteria, modeling of intermediate model, or translation from one model into other model.

The classification of non-automated steps for each MBT approach is listed in the Appendix B. Table 10 lists the summary of non-automated steps classification for complexity levels:

Table 10. Number of non-automated steps for different complexity level

Step Complexity Level	# Approaches		
	Priority Level 1	Priority Level 2	TOTAL
Necessary (Initial Behavior Model Modeling)	42	23	65
Low (Only choices)	8	6	14
Medium (Intermediate Model Modeling or Test Data Definition)	13	3	16
High (Translation or other task)	8	0	8
Not Applied	2	1	3

All approaches have at least one non-automated step: The initial modeling of software behavior. For this reason, this step has been classified as “Necessary”. But in some approaches the initial modeling is a hard task, involving translation between models, data modeling, etc.

Other types of non-automated step may represent extra effort, time, and cost in addition to the application of the MBT approach. Manual steps to select a *choice* have been classified as *Low Complexity* because the tester needs just to select one option (testing criteria to be applied, test case to be executed or filtered, etc…). This manual step was found in 14 approaches. Examples are: (du BOUSQUET *et al*., 1999; MANDRIOLI *et al*., 1995; MINGSONG *et al.*, 2006), and (TAHAT *et al*., 2001). Manual steps that require intermediate-model modeling or test data definition by hand have been classified as *Medium Complexity*. They were found in 16 approaches. Examples are: (ANDREWS *et al*., 2005), (BASANIERI and BERTOLINO, 2000), (OFFUT *et al*., 2003), and (Vieira *et al*., 2006). Several approaches use intermediate model during the test case generation process, but in the most of cases are generated automatically. Manual generation of intermediate model is a hard task. In several UML based MBT approaches (13 approaches), intermediate models are need for test data definition. This is because UML diagrams describe usually just the software structure, and not the inputs.

Translation from one model into another model is a hard step. It involves the application of rules, constraints to derive one model and can make the test generation process very difficult or even impossible. This step has been classified as *High Complexity* and was found in 8 approaches (all in Priority Level 1 approaches). Examples are: (BRIAND *et al*., 2004), (OFFUT *et al*., 2003), and (RUMPE, 2003).

3.1.7 Analysis of Papers References

The level of acceptance of any approach by the Software Engineering community is an important factor in our analysis. The number of citations for each paper may indicate the paper/approach visibility or quality in the community. Papers are likely to be cited more often by other authors when describing interest concepts or results regarding a research field.

Table 11 lists the number of citation for each paper in both priority levels:

Table 11. Number of citation for papers in the priority levels 1 and 2

Approach	# citations	# papers in PL 1 and 2 cited by it	Priority Level
Offutt1999b	24	2	1
Richardson1992	17	0	2
Stocks1996 (1)	13	0	2
Offutt2003	12	10	1
Briand2002	12	3	1
Hartmann2000	11	1	1
Kim1999	10	1	1
Gargantini1999	9	3	2
Abdurazik2000	9	1	1
Stocks1993a (1)	8	0	2
Chang1996	7	1	2
Dalal1999	7	1	2
Stocks1993 (1)	7	1	2
Parissis1996	6	1	2
Offut1999a	5	4	2
Ammann1994	5	3	2
Basanieri2000	4	3	1
Mandrioli1995	4	1	2
Amla1992	4	0	2
Dalal1998	4	0	2
Pretschner2001 (2)	4	0	2
Pretschner2001a (2)	4	0	2
Chang1999	3	3	2
Friedman2002	3	2	2
Riebisch2002	3	2	1
Bousquet1999a	3	1	2
Richardson1996	3	1	2
Barbey1996	3	0	2
Hong2000	3	0	2
Tahat2001	3	0	2
Chevalley2001	2	3	1
Kansomkeat2003	1	5	1
Nebut2006	1	5	1

[1] These three papers describe the same approach, and they were analyzed as one paper.
[2] These two papers describe the same approach, and they were analyzed as one paper.

Paradkar2004a (3)	1	5	2
Linzhang2004	1	4	1
Wu2003	1	3	1
Bertolino2003	1	2	1
Scheetz1999	1	2	1
Briand2002a	1	1	1
Liuying1999	1	0	1
Ali2005	0	9	1
Bertolino2005	0	9	1
Murthy2006	0	7	1
Sinha2006a (3)	0	6	2
Briand2004a	0	5	1
Chen2005 (3)	0	5	2
Briand2004b	0	4	1
Sokenou2006	0	4	1
Lund2006	0	3	1
Vieira2006	0	3	1
Andrews2005 (3)	0	3	2
Satpathy2005 (3)	0	2	2
Botaschanjan2004	0	1	1
Garousi2006	0	1	1
Hartman2004	0	1	1
Rumpe2003	0	1	1
Traore2003	0	1	1
Zhen2004	0	1	1
Tan2004 (3)	0	1	2
Htoon2005 (3)	0	1	2
Xu2006a (3)	0	1	2
Bellettini2005	0	0	1
Bernard2006	0	0	1
Beyer2003	0	0	1
Briand2002b	0	0	1
Briand2006	0	0	1
Carpenter1999	0	0	1
Cavarra2003	0	0	1
Chen2002	0	0	1
Crichton2001	0	0	1
Deng2004	0	0	1
Gnesi2004	0	0	1
Gross2005	0	0	1

[3] These ten papers were selected subjectively by the researchers.

Lucio2005	0	0	1
Meyer1998	0	0	1
Mingsong2006	0	0	1
Olimpiew2005	0	0	1
Stobie2005a (3)	0	0	2
Belletini2005	0	0	1
Okika2006 (3)	0	0	2

One of the most cited papers is from (OFFUT and ABDURAZIK, 1999). This is recognized as the first published MBT approach using UML. The approach uses statechart for Model-based System Testing. In the Top 10, 6 papers use UML, they are: (OFFUT and ABDURAZIK, 1999), (OFFUT *et al.*, 2003), (BRIAND and LABICHE, 2002), (HARTMANN *et al.*, 2000), (KIM *et al.*, 1999), (ABDURAZIK and OFFUT, 2000), and 4 papers use non-UML approaches (2 of them describe the same approach), they are: (RICHARDSON *et al.*, 1992), (STOCKS and CARRINGTON, 1996), (GARGANTINI and HEITMEYER, 1999), and (STOCKS and CARRINGTON, 1993a).

Papers with high visibility and popular among research groups provide valuable information on new directions in MBT research and provide the state-of-art. These approaches need to be compared closely with emerging approaches.

We have also identified interesting pre-1990 papers. These papers describe fundamentals concepts that have been used in later MBT approaches. The papers are:

- **C. V. RAMAMOORTHY, S. F. HO, and W. T. CHEN, "On the automated generation of program test data", IEEE Transactions on Software Engineering, SE-2(4):293–300, Dec. 1976.**

  This paper described the seminar approach for TEST DATA generation. In this approach, given a program graph, a set of paths are identified to satisfy some given testing criteria. It generates test data from FORTRAN programs. This paper is referenced by several MBT approaches to indicate the origin of automated tests generation.

- **T.S. CHOW, "Testing software design modeled by finite-state machines", IEEE Trans. Software Eng., vol. SE-4, pp. 178-187, Mar, 1978.**

  This paper is the seminar approach for Control-flow based Testing. It proposes a method of testing the correctness of control structures that can be modeled by a *finite-state machine. Test results derived from the design are evaluated against the specification. The method is based on a result in automata theory and can be applied to software testing. This paper is referenced by several MBT approaches to indicate the origin of automated tests generation based on control-flow.*

- **D. L. BIRD and C. U. MUNOZ, "Automatic generation of random self-checking test cases", IBM Systems Journal, 22(3):229–245, 1983.**

  This paper presents a Random test case generation techniques [2] generate test cases by randomly selecting input values. This technique may be applied to building generators for testing several different types of program, including PL/I compilers, a graphics display manager, and sodmerge routines. The authors emphasize that although concepts of test case generation are similar for different types of software, each test case generator is a specific tool that must be coded separately. This information is referenced a lot of times by MBT approaches.

- **S. RAPPS, S. and E.J. WEYUKER, "Data Flow analysis techniques for test data selection", In: International Conference on Software Engineering, pp. 272-278, Tokyo, Sep, 1982.**

  This paper defines a family of program test data selection criteria derived from data flow analysis techniques similar to those used in compiler optimization. It is argued that currently used path selection criteria, which examine only the control flow of a program, are inadequate for testing. This procedure associates with each point in a program at which a variable is defined, those points at which the value is used. Several test data selection criteria, differing in the type and number of these associations, are defined and compared. This paper is referenced by several MBT approaches, usually, when they are using some test-data criteria defined and compared in this paper.

- **T. J. OSTRAND and M. J. BALCER, "The Category-Partition Method for Specifying and Generating Functional Tests", Communications of the ACM, 31(6), June 1988, pp. 676-686.**

  This paper describes by the first time the Category-Partition Method for Testing Functional Generation. It is method for creating functional test suites in which a *test engineer analyzes the system specification, writes a series of formal test specifications (categories and partitions of values for each categories) to produce test descriptions. This method is used by a lot of MBT approaches for test data definition.*

Appendix B is a summary of all approach analyzed during this work. More detailed information (subjective information, summary, limitations and restrictions) about each approach is available on JabRef Tool's database.

3.2 Qualitative Analysis

This section describes the qualitative analysis, that is, subjective information on limitations, cost, effort and complexity, input, pre-requirements or skill required to use them, and quality of the outputs generated for various MBT approaches.

The Qualitative Analysis has been accomplished for 15 papers selected using the following criteria:

- The first 12 papers (removing repeated approaches) have the highest number of citation;
- 3 papers of select interest from 2004 to 2006 have been analyzed.

The papers are described on Table 12:

Table 12. Papers selected for Qualitative Analysis

Approach	Title	Authors
Abdurazik2000	Using UML Collaboration Diagrams for Static Checking and Test Generation	A. Abdurazik, J. Offutt
Briand2002	A UML-Based Approach to System Testing	L.C. Briand, Y. Labiche
Chang1996	Structural specification-based testing with ADL	J. Chang, D.J. Richardson, S. Sankar
Dalal1999	Model-based testing in practice	S.R. Dalal, A. Jain, N. Karunanithi, J.M Leaton, C.M. Lott, G.C Patton, B.M Horowitz
Gargantini1999	Using model checking to generate tests from requirements specifications	A. Gargantini, C. Heitmeyer
Hartmann2000	UML-based integration testing	A. Hartman, K. Nagin
Kim1999	Test cases generation from UML state diagrams	Y.G. Kim, H.S. Hong, D.H. Bae, S.D. Cha

Mingsong2006 [4]	Automatic test case generation for UML activity diagrams	C. Mingsong, Q. Xiaokang, L. Xuandong
Offutt1999b	Generating tests from UML specifications	J. Offutt, A. Abdurazik
Offutt2003	Generating Test Data from State-Based Specifications	J. Offutt, S. Liu, A. Abdurazik, P. Ammann
Parissis1996	Specification-based testing of synchronous software	I. Parissis, F. Ouabdesselam
Richardson1992	Specification-based test oracles for reactive systems	D.J. Richardson, S.L. Aha, T.O. O'Malley
Stobie2005a [4]	Model Based Testing in Practice at Microsoft	K. Stobie
Stocks1996	A Framework for Specification-Based Testing	P. Stocks, D. Carrington
Vieira2006 [4]	Automation of GUI testing using a model-driven approach	M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, J. Kazmeier

3.2.1 Limitation of Models, Tools, and Input and Output formats

The three key elements in a MBT approach are: the **model** used for the software behavior description, the **tools** that generate supporting infrastructure for the tests (including expected outputs), and the **test-generation algorithm** (criteria). In this section, we discuss qualitatively about the first two – model and tool, their characteristics, and limitations, for each of the selected approach.

The models must describe the software structure or behavior, and then it defines what is possible to test using a MBT approach. We can define two types of models in a MBT approach: (1) an initial model, that describes the software specification from the software development process, and (2) an intermediate model, that it is used to support the test case generation.

The intermediate model is used to define just the relevant information for the test generation process, excluding information and characteristics not related to this process. If it is performed automatically, there is no additional cost, effort, time, or complexity for this step. Otherwise, it may represent more cost, effort, time, complexity for the process, and introduce a source of error to the MBT approach. Ideally, tests are generated without intermediate model.

The correctness of a behavioral model is essential to applying the MBT approach. If the model is wrong, the generated tests will be invalid, e.g. they checking nothing. To solve this problem, several model-checking approaches have been defined and

[4] This paper is one of the papers that have been included for qualitative analysis subset subjectively by the researchers.

integrated with MBT approaches. Examples in the 1st and 2nd Priority Level are: (GARGANTINI and HEITMEYE, 1999), (ABDURAZIK and OFFUT, 2000), (BRIAND and LABICHE, 2002), (TAN *et al*., 2004), and (SATPATHY *et al*., 2005). Table 13 identifies which of the selected papers use model-checker during test case generation.

The second key element is the type of tools that is used to support the test case generation process. In an ideal case, a single tool must support all steps starting for software behavior modeling (or importing inputs from other external tool) until the generation of expected results, automating all the main steps of the approach. The quality of a MBT tool depends on the level of automation (LA) of the test generation process, and the input (IN) used and output (OUT) generated by the tool. The format/notation used as input or output determines the feasibility of extending the tool. For example, tools using some of known notations, like XMI (standard to describe UML Specification), XMI, B language, or TSL, are extended easier than tools using other unknown format/notation.

Sometimes an approach may require the use of external tools (ET) for steps such as generation of a software behavior model or test scripts to be executed in a specific platform. This restricts the use of the approach. Some external tools are proprietary, and require a license fee.

Table 13 shows the quality of tools used for each MBT approach. The symbol ✓ indicates that the tool characteristics analyzed are good for a MBT approach, because use a defined format and generate test cases automatically; the symbol ⚠ indicates that the tool characteristics don't make easy the test generation process, but they also don't confuse this process, for example because some steps are not-automated or the format of input or output is not defined; and the symbol ✗ indicates that the tool don't have good characteristics for MBT, for example if all steps are not-automated or there is not tool to support this MBT approach.

Table 13. Qualitative analysis about Models, Tools, Input and Outputs

Approach	Model Analysis		MBT Tool Analysis	
	Limitation of Behavior Model	Using Model Checker	Attributes	Status
Abdurazik2000	UML Collaboration Diagrams. This model describes just the relation among classes in a specific sequence in object-oriented software, but there is not information about the timing	Using	There is no tool defined to support this approach	✗

	and parallelism between events. It describes the control-flow, and the test data are defined by the instance of classes (objects). Information about non-functional requirement (like timing, parallelism between events) cannot be described.			
Briand2002	6 UML Diagrams. The use of 6 models allows a more precise modeling, but the effort to model and be consistent is hard. This approach uses a strategy to check the models before the test generation.	Using	• **LA (Level of Automation):** all-steps • **IN (Input):** generated by itself • **OUT (Output):** use a format defined by OMG • **ET (External Software):** not used	✔
Chang1996	ADL Specification. This language defines just assertions (a Boolean expression that must evaluate to true at the termination of function execution) about software (from calls and requests) and doesn't allow to model complex behavior. The test data must be defined externally using a specific description format. This model is applied just for Unit testing a=in specific software categories.	Not using	• **LA:** all-steps • **IN:** generated by itself • **OUT:** test drivers in C format • **ET:** not used	✔
Dalal1999	AETGSpec. This model is useful to describe just the data-flow (no control-flow) and it is not expressive to describe complex constructs. According with the authors, training is necessary for testers to be able to use it.	Not using	• **LA:** all-steps • **IN:** generated by itself • **OUT:** test cases in a table format • **ET:** not used	✔
Gargantini1999	SCR Specification is based on a user-friendly tabular notation and offers several automated techniques for detecting errors in software requirements specifications, including an automated consistency checker to detect missing cases and other application-independent errors. An SCR requirements specification describes both control-flow and data-flow, and test cases are generated like a counterexample of sequences. Non-determinist transitions may be modeled. However, non-functional requirements cannot be described.	Using	• **LA:** all-steps • **IN:** imported from a model checker (SMV or SPIN) • **OUT:** test sequences in text file format • **ET:** Model checker (SMV or SPIN)	✔
Hartmann2000	UML State Diagram. This model is useful to describe the software internal behavior and the set of states that one class can be (control-flow). The tests are applied to check if software behaves as described in the requirements for each events or transactions. To describe data-flow, categories of states are defined using category-partition method, from other diagram.	Not using	• **LA:** all-steps • **IN:** imported from XMI format • **OUT:** test cases in TSL format • **ET:** Rational Rose (proprietary tool)	⚠
Kim1999	UML State Diagram. This model is useful to describe the software internal behavior and the set of states that one class can be (control-flow) and events and transactions (data-flow). This approach is applied to test the static behavior of the software (classes, but without generalization and inheritance among classes), since dynamic behavior is not supported.	Not using	There is no tool defined to support this approach	✘
Mingsong2006	UML 2.0 Activity Diagram. This model is useful to describe the limited control-flow for testing (no timing aspects, for example), but the data-flow is not described. In this approach, the test data are generated randomly to support test cases generation.	Not using	• **LA:** all steps • **IN:** generated by itself using eclipse plug-in • **OUT:** test case in Java code format	✔

			• **ET:** not used	
Offutt1999b	UML State Diagram. This model is useful to describe the software internal behavior and the set of states that one class can be (control-flow). The tests are applied to check if software behaves as described in the requirements for each events or transactions. To describe test data, instances of objects are used.	Not using	• **LA:** all-steps • **IN:** imported from XMI format • **OUT:** test cases in ASCII text file format • **ET:** Rational Rose (proprietary tool)	⚠
Offutt2003	UML State Diagram. This model is useful to describe the software internal behavior and the set of states that one class can be (control-flow). Test data are defined like pre-condition or pos-condition for each state.	Not using	• **LA:** all-steps • **IN:** generated by itself • **OUT:** Test data in LUSTRE format • **ET:** not used	✗
Parissis1996	Environment Specification and the safety properties in LUSTRE language. A LUSTRE program is structured into nodes: a node is a subprogram describing a relation between its input and output variables. Once declared, a node may be used in any expression, as a basic operator. LUSTRE allows the modeling of security properties to evaluate reactive software. Just control-flow is used to generate tests. The purpose of it to generate TEST DATA.	Using	• **LA:** test script isn't generated automatically • **IN:** models translated manually from external tools models • **OUT:** test scripts • **ET:** SCR tool and Rational Rose (proprietary tools)	✓
Richardson1992	System Specification In Z format. It is based on set theory and predicate calculus, uses fairly standard mathematical notation and supports standard set and predicate operations. Systems are described in Z using a state-based model (state, event, operation). Training is necessary to use this notation to describe software behavior using Z.	Not using	There is no tool defined to support this approach	✗
Stobie2005a	FSM and ASML. This model is useful to describe the control-flow for testing and for model checking, but the data-flow is not described. Moreover the authors say that the cost to develop these models is a little high, and information about non-functional requirement (like timing, parallelism between events) cannot be described.	Using	**There are two tools:** • **LA:** all steps • **IN:** generated by AsmL tool • **OUT:** test case in XML format • **ET:** not used	✓
Stocks1996	Test Template In Z format. It is based on set theory and predicate calculus, uses fairly standard mathematical notation and supports standard set and predicate operations. Systems are described in Z using a state-based model (state, event, operation). Training is necessary to use this notation to describe software behavior using Z. The software behavior described using this model is limited and it is used just to generate test data (no control-flow).	Not using	There is no tool defined to support this approach	✗
Vieira2006	UML Use Case, Activity Diagram, and Class Diagrams. Activity Diagrams are used to define the control-flow for each use case, but are defined notes to link activity and class diagrams (where is defined test data using category-partition). However, information about non-functional requirement (like timing, parallelism between events) cannot be described.	Not using	• **LA:** all steps • **IN:** generated by itself using eclipse plug-in • **OUT:** test case in TSL format • **ET:** not used	✓

An important question regarding behavior models is the limitation imposed by the language used to describe it. Some languages don’t allow the description of control-flow and data-flow in the same model (or group of models). The language may limit the definition of the testing coverage.

Other question is related to the language’s power to describe the software behavior. Some approaches don’t allow the description of complex models to be used during test cases generation, decreasing the testing quality. UML diagrams are easy to be used because we are familiar with them, but other notations, like Finite State Machine (FSM), and B Language are powerful and may also be used for software modeling.

3.2.2 Complexity, effort, and cost to apply MBT approaches

MBT approaches are, usually, developed by a research group to be applied in a specific context. After that, it may be used in other general contexts or on the field. To apply a new approach on the field is a hard task, because it involves training, adaptation, and consequently time, resources, and cost.

A MBT approach is very useful for systems that change continually (or in constant evolution) as regenerate test cases make the testers’ task easier.

The ideal way to evaluate an MBT approach is through Experimental Study, but this may not be feasible. Alternatively, complexity of MBT approaches is analyzed subjectively based on the skill, complexity, effort, and cost necessary to use it in a real project.

According to Dalal (1999), many approaches expect the tester to be 1/3 developer, 1/3 system engineer, and 1/3 tester. Unfortunately, such combination of skills is rare and the budget to hire may not exist, technology that does not adequately take into account the competence of a majority of its users is not useful.

Table 14 describes usage complexity for each MBT approach. The symbol ▼ indicates that the characteristic evaluated (skill, effort, or cost) has a complexity low-level, and make easy the use of the MBT approach. The symbol ▲ indicates that the characteristic evaluated has a complexity high-level, and make difficult or unfeasible to use one MBT approach. And, finally, the symbol indicates that the characteristic evaluated has a complexity intermediate-level, and it's necessary to spend an additional time to avoid these problems, but its use is still feasible.

Table 14. Qualitative analysis about the Complexity to use MBT approaches

Approach	Skill Necessary	Effort to apply	Costs
Abdurazik2000	• UML Modeling	• Software modeling using UML collaboration diagram • All steps has a strategy for automation	• There is no tool to support, then, all steps must be performed manually
Briand2002	• UML and OCL Modeling	• Software modeling using some UML Diagrams • All steps has a strategy for automation	• There is a tool to automate all steps • Models used during the development process may be used during tests
Chang1996	• ADL Modeling	• Software modeling using ADL Specifications, Boolean conditions and test data descriptions • This approach supports the test execution for C Programs	• There is a tool to automate all steps
Dalal1999	• AETGSpec Modeling • Equivalence Class Method	• Software (Test data) modeling using AETGSpec	• There is a tool to automate all steps
Gargantini1999	• SCR Specifications Modeling	• Software modeling using SCR Specification • The approach assumes that the customers have formulated a set of systems properties previously • A translation from SCR Model into SMV or SPIN model using a model checker	• The incompleteness of the generated test sequences • Currently, the approach constructs only ONE test sequence for each branch. • There is a tool to automate all steps
Hartmann2000	• UML Modeling • Category-partition method	• Software modeling using some UML Diagrams • All steps has a strategy for automation • Test case generation produces test cases in TSL format, but this approach also allows the test execution for components based on COM/DCOM and CORBA middleware.	• Rational Rose (proprietary tool) is used for software modeling
Kim1999		• Software modeling using UML	• There is no tool to

	• UML Modeling	state diagram • All steps has a strategy for automation	support, then, all steps must be performed manually
Mingsong2006	• UML Modeling • Java Programming	• Software modeling using UML activity diagram • Almost steps are non-automated • Code instrumentation is necessary to apply this MBT approach	• There is a tool to automate all steps • This approach is must be applied after the implementation, then the models cannot be used during the development
Offutt1999b	• UML Modeling	• Software modeling using UML state diagram • All steps has a strategy for automation	• Rational Rose (proprietary tool) is used for software modeling • The results obtained on a empirical study show the effectiveness of this approach for test coverage
Offutt2003	• UML Modeling	• Software modeling using UML state diagram • Almost all steps have a strategy for automation. The test scripts generation is not automated.	• The results obtained on a case study show the effectiveness of this approach for test case generation, faults found and test coverage
Parissis1996	• LUSTRE Modeling	• According with the authors, software modeling using LUSTRE is a hard task • All steps has a strategy for automation	• There is a tool to automate all steps
Richardson1992	• Reactive System Modeling (including temporal requirements) • Z Language	• Two steps in the test generation process must be performed manually • There is no information about how to perform the non-automated steps	• There is no tool to support, then, all steps must be performed manually
Stobie2005a	• Finite State Machine (FSM) Modeling • Abstract State Machine Language (ASML) Modeling	• Modeling the software behavior using ASML • All steps has a strategy for automation	• There is a tool to automate all steps • Models used during the development process may be used during tests
Stocks1996	• Z Language Modeling	• Input and output space modeling using Z notation • The steps for test case	• There is no tool to support, then, all steps must be

		generation are not described	performed manually
Vieira2006	• UML Modeling • Category-partition method	• Software modeling using a specific UML Modeling Tool and test generator • All steps has a strategy for automation	• There is a tool to automate all steps • Models used during the development process may be used during tests

All approaches need a basic understanding of Modeling Language. MBT approaches are easier for Software Engineering professionals having UML training from university. Other languages, cited in the Table 14 may need some basic training.

The effort to use each an approach depends on the complexity of all manual steps (initial modeling or generation). Most approaches need modeling (low complexity). However, some don't describe how to perform the non-automated steps, or describe the steps as hard.

The cost to apply each approach defines its impacts (for the schedule or budget) during the software development process. For example, one tool decrease the cost to apply one approach, but the using of a proprietary tool could make this approach less practical. Moreover, sometimes the benefits to use one approach are low because the output generated is not precise or incomplete, and consequently, the cost is high.

3.2.3 Changing the paradigm of MBT and the testing criteria

The Testing Coverage Criteria is divided in two types, and both of them are essential to generate test cases. The types are: **control-flow** and **data-flow**. We can observe that just some approaches use both types to generate test cases, and others just one type. Table 15 describes the analysis about the type of testing coverage criteria used by MBT approaches. The types can be: Data-flow, Control-flow, or Both (control-flow and data-flow).

Both (data-flow and control-flow) are important during the test generation process. However, one other characteristic extracted from these approaches is that previously (this behavior can be observed on the approaches published approximately until 1999), the focus was on the data-flow based testing coverage, that is, either the approach uses data-flow, like (CHANG *et al.*, 1996), (PARISSIS and OUABDESSELAM, 1996), (DALAL *et al.*, 1999) or use both (data-flow and control-flow). The reason for that is that the languages/technique used for software modeling were previously limited and focused on black-box technique, that consider software as a closed function when the data as inputted and a specific results are expected. Moreover, the systems were considered simple functions or a small module.

Table 15. Qualitative analysis about the type of testing coverage criteria

Approach	Publication Year	Testing Coverage Type
Richardson1992	1992	**Both:** data-flow (equivalence class) and control-flow (paths)
Chang1996	1996	**Data-flow:** from test data description
Parissis1996	1996	**Data-flow:** it is used a random test data generator
Stocks1996	1996	**Both:** data-flow and control-flow
Dalal1999	1999	**Data-flow:** equivalence class
Gargantini1999	1999	**Both:** data-flow (data boundary) and control-flow (branch coverage)
Kim1999	1999	**Both:** data-flow and control-flow
Offutt1999b	1999	**Control-flow** (this approach is used for TEST DATA generation)
Abdurazik2000	2000	**Both:** data-flow and control-flow
Hartmann2000	2000	**Both:** data-flow and control-flow
Briand2002	2002	**Control-flow:** path extracted from activity diagrams
Offutt2003	2003	**Both:** data-flow and control-flow
Stobie2005a	2005	**Control-flow:** path extracted from activity diagrams
Mingsong2006	2006	**Control-flow:** path extracted from activity diagrams
Vieira2006	2006	**Both:** data-flow (category-partition) and control-flow (activity diagram)

More recently, the focus changed, and most approaches are working, frequently, with the structure of the system (control-flow based testing coverage) for test case generation, like (BRIAND and LABICHE, 2002), (MINGSONG *et al.*, 2006), (STOBIE, 2005). This happened because the languages/techniques developed recently are able to describe structural information about the software behavior, making easy the analysis of software control-flow. Moreover, the system's complexity is increasing each year, and it's necessary to evaluate not just small modules, but full systems. If we analyze all paper included in the 1st and 2nd Priority Level, this behavior can be visualized easier than looking at just the selected 15 papers.

This fact created a problem, because currently we need to use both (data and control) criteria to generate test cases. Therefore, the main question to be answered by the approaches is: How to use both? Which model to use to represents each one? Can I use only one model to describe all (because if I have more models to describe, the cost and effort to apply this approach increase)?

3.2.4 Positive and Negative aspects from MBT approaches

Each MBT approach has specific characteristics that make it different from other approaches and cannot be used to define if one approach is better than other approach or not. In different contexts, each characteristic may be interesting or not. For example, if one approach uses the model *X* and other approach uses the model *Y* for test case generation, it is impossible to state that one approach is better than the other one. These specifics characteristics were described and analyzed during this section.

In this last section, we present some positive and negative aspects from each MBT approach. This information cannot be used to compare them, but may help us during the selection or classification of one approach to be used in a specific project with specific characteristics.

Table 16 described the positive and negative aspects for each MBT approach select for qualitative analysis.

Table 16. Positive and Negative aspects of MBT approaches

Approaches	Positive Aspects	Negative Aspects
Abdurazik2000	- All process may be executed automatically; - This approach introduces an algorithm for code instrumentation of a program that is implemented from collaboration diagrams; - This approach included techniques to be used for statically and dynamically;	- There is not tool built to support this approach; - The authors describe the intention to integrate eventual built tool to support this approach with Rational Rose, a proprietary tool; - This approach doesn't generate test cases, but only test data; - There is not information about a case study using this approach; - Collaboration diagram doesn't define timing information and parallelism between events;
Briand2002	- All process is executed automatically using TOTEM; - The authors discuss about the use of non-functional requirement to generate test cases as future task; - The models used are extracted from artifacts produced during the software development process (analysis phase); - This research is in progress, and new approaches may be developed for MBT;	- The approach hasn't been evaluated on the field;
Chang1996	- All process may be executed automatically; - There is a strategy to selection the tests (filtering);	- Applied just for C programs; - There is no information about how to define the testing coverage criteria; - There is no information about how to define the test data; - The authors describes that this technique

		complements code-based techniques, but the idea of MBT is to start testing before the definition of the code, using a model describing the software behavior;
Dalal1999	- The tight coupling of the tests to the requirements; - The ease with testers can write the data model; - The ability to regenerate tests rapidly in response to changes; - There are a lot of hints about MBT in general; - There are a lot of information about future research regarding MBT;	- It works just with test data. The control flow is not considered; - There is no an oracle to store the test cases. - The demand for development skills from testers to use this approach; - This approach is applied for system of low complexity (test's response to a stimulus is relatively low);
Gargantini1999	- This approach uses model checking to generate test sequences from counterexample. It's an interesting solution; - This approach uses control and data coverage to generate test sequences; - The process is automated using the developed tool;	- The papers describes that this approach generates test suite to cover all error. It's impossible to aim that; - The authors said that the method assumes that the customers have formulated a set of systems properties (it is not a normal step during the requirements specification); - The incompleteness of the test sequences; - The method assumes the correctness of both the operational specification and properties;
Hartmann2000	- The process is automated using the developed tool; - This approach defines test cases from the data-flow and control-flow using different criteria; - This approach generates test scripts in TSL format (may be executed in several testing execution tools); - This approach is able to execute the test for components based on COM- and CORBA.	- This approach is not applied for asynchronous communication, because it requires the modeling of these queued messages and events. - The internal data conditions of these state machines (meaning the global state machine variables) influencing the transition behavior are not supported. - Concurrent states are also not supported as yet. - This approach uses Rational Rose (proprietary tool).
Kim1999	- This approach uses control flow and data flow to generate test case, differently of some approaches that use only one of these (either generate test data or generate paths); - All steps may be executed automatically. The steps are described on the paper;	- Applied to test the software static behavior. - There is no information about the type of inputs and outputs used by that; - There is no information about the efficiency of this approach in a real project;
Mingsong2006	- The process is automated using the developed tool; - This approach is able to generate test cases and EXECUTE the tests for Java programs supporting all testing process, including testing analysis;	- The approach is applied only for JAVA programs; - There is not a case study to describe the use of the approach in a real project; - The test data are generated randomly (no using a test data generation criteria). Just the control-flow uses testing criteria; - Code instrumentation in Java programs is necessary to apply this approach;
Offutt1999b	- The process is fully automated using the developed tool; - This is the first MBT approach based on UML identified on the technical literature; - Empirical results from using UMLTest to evaluate the testing criteria were given. The results indicate that highly effective tests can be automatically generated for system level testing;	- This approach uses Rational Rose (proprietary tool); - The current tool requires all variables to be Boolean. Other types are not supported; - Test data are defined by parameters of events, without use a data-flow strategy;
Offutt2003	- There is a tool to support this approach; - This approach has 4 testing criteria level	- There are 2 hard non-automated steps to be performed using this approach;

	to be applied; - Empirical results shows the effectiveness of this approach about number of test cases, faults found, faults missed, coverage for each testing criteria;	- Test data are defined by parameters of events, without use a data-flow strategy; - The SpecTest tool parses specifications into a general format (the specification graph) using algebraic predicate representation, and currently there is one restriction that it processes only one mode class for SCR specifications at a time, and one statechart class for UML specifications.
Parissis1996	- It works with Non-functional requirements (security properties) to generate test data; - It allows for fully automated test data generation consistent with the environment assumptions; - Test data generation ignores test data which are of no interest for the testing process; - Provide a formal framework for testing safety-critical software;	- It works just with test data, no control flow; - Specifying the software environment by means of invariant properties is a rather difficult task. It's necessary to learn how to use LUSTRE before to use this approach;
Richardson1992	- The use of Testing Oracle may allow exhaustive testing, because the subset of output is defined; - Regression Testing may be accomplished because the Testing Oracle is defined; - Temporal Requirements are used to generate test oracles, because this information is essential for reactive systems;	- There is no information about the input and output used by the approach; - There is no information about tools to support; - This approach can hardly ever be automated because it uses a non executable specification language;
Stobie2005a	- There are two tools to support the test case generation; - There are 6 options for testing coverage. The tester may choose which one to use in your project; - The tester may use Assertions (constraints) for model checking - The outputs are generated in XML format; - The quality of the approach is described on the test by the increasing of the coverage and number of test cases, reducing of time and effort in comparison with manual test design;	- The cost of developing a model is high (the authors said); - The learning curves to use this approach is a little high; - The tools used are perceived as too expensive or too complex by most testers; - This approach uses just Microsoft's Technologies; - This approach work just with control flow (no test data);
Stocks1996	- The papers describes about the importance of control and data criteria to generate test cases. The most of times, just one of that is used by the approaches; - The Test Templates are generated like a tree, from a valid input to other inputs changing the initial state;	- There is no information about the steps to be accomplished to generate test cases; - There is no information about tool to support; - There is no information about the model used for input and about the outputs;
Vieira2006	- The process is automated using the developed tool; - Almost models used are extracted from artifacts produced during the software development process (analysis phase); - This approach defines test cases from the data-flow and control-flow using different criteria; - This approach generates test scripts in TSL format (may be executed in several testing execution tools); - The paper describes the positive results (reduction of effort, test cases, cost and time) about the use of this approach in a project.	- This approach is applied just for GUI Testing; - The modeling of category and values using class diagrams is necessary to apply the category-partition method;

4 Towards new research regarding MBT

After the evaluation of MBT approaches, some insights were observed that could be future research topics in MBT. These observations can be divided into:

1. Important aspects to be considered during the development or application of a new MBT approach. These aspects are discussed in section 4.1.
2. Potential future research topics in MBT. Three topics for new research are introduced, motivated, discussed, and some examples are presented. Subsequently, for each topic, preliminary guidelines about how to develop the research and what are the expected results are presented. This information is showed in section 4.2.

4.1 Observations for application or development of a MBT approach

The observations extracted from 74 papers describe important aspects to be considered during the application or development of a MBT approach. They are enclosed in boxes in this section. The first box discusses observations to be considered during the development of a new MBT approach, and the second box discusses observations related to the selection of a MBT approach for a project.

4.1.1 The model to describe the software behavior is a key point

The behavior model must be developed carefully. Its limitation and the restrictions must be respected during the modeling of the software under test. Moreover, the correctness of a model is fundamental to start the test case generation process. If the model is wrong, the generated tests will be invalid.

Some approaches present a model-checker integrated with the test case generation process. This is an interesting solution, but usually the checker evaluates just the syntax of the model, not the semantic. Errors from modeling may still exist.

Another aspect is the language/notation used for modeling. UML is more common than other languages. The expressive differences among the languages determine the effectiveness of test case generation process. Important advantages of UML are:

- UML is an international standard for object-oriented software modeling defined by OMG (Object Management Group);
- UML have been used more that other languages;
- Software Engineering professionals have UML basic training in the university. This makes it easy to apply UML based MBT approaches;
- There are a lot of tools available to support software modeling using UML.

On Intermediate models, we observe that the translation from a model into another model may introduce a new source of error, and the strategy to support the translation must be defined. If this task is done automatically, the possibility of mistakes is lower than if it is done by hand.

While choosing the behavior model to be used in a MBT approach, we have to analyze its limitations, and what it can or cannot represent. This information may limit the use of this MBT approach.

Check if the behavior model allows modeling of our system and if there are intermediate models to be developed. Assure the correctness of the model before to perform the test case generation using a checker or another solution.

4.1.2 Testing coverage must address control-flow and data-flow criteria

Testing coverage criteria define the set of test cases selected for testing. Test cases can be generated from two sources: control-flow (structure or path extracted from the software) or data-flow (test data to be input during the test execution) criteria. In the case of testing applied for large system is essential to combine both criteria to increase the testing coverage. Usually, a test execution strategy is composed of test procedures (describing the sequence to be followed from the control-flow) and test cases (describing the test data to be input from the data-flow).

If just one of these two criteria were used, the MBT approach needs either test data or test procedures to be defined manually.

While defining the testing coverage criteria, we should try to combine control-flow and data-flow in test case generation to increase the coverage. Otherwise, the approach has limitation.

Check if the set of test cases generated by these criteria supplied by this approach is either insufficient, excessive, or enough to test our system.

4.1.3 The level of automation must be high

Level of automation is one important factor to define the feasibility to use a MBT approach. Usually, one MBT approach has only one non-automated step: the modeling of software behavior used as input to test case generation process. Other non-automated steps increase the effort, cost, and complexity to use the approach.

Non-automated steps have different complexity levels. The definition of the level depends on the language/notation used for modeling, and algorithms or choices required to perform a step. If a MBT approach has a high level of automation, the complexity, effort, and cost for application are low.

Try to develop a MBT approach with high level of automation. We need to create (or use) algorithms to support each steps, and describe guidelines to perform them. If there are non-automated steps, we should try to reduce their complexity level.

Check if the automation level is high, and if we are prepared to perform the non-automated steps that compose the test case generation process. Sometimes the complexity of these steps is high and requires special resource or skills.

4.1.4 Tool supporting the automation of the steps is fundamental

Tools must support the execution of automatic steps. However, if there is not a tool available to support the steps, this task may be hard and unfeasible (for example, to apply an algorithm to generate test cases manually is not trivial). A tool must supply the integration between non-automated (usually modeling) and automated (usually generation) steps.

The importance of the integration between automated and manual steps can be observed in the following example: in one approach, the test case generation if performed automatically by an available tool. However, the model used as input for the testing process is described on a text or XML file developed manually because there is not tool to support the modeling. The cost and effort of this task can damage the use of a MBT approach.

Proprietary tools often are stable and feature-rich, but may require a license fee limiting its use in some organization.

Construct a tool to make easier the application of this MBT approach. An important aspect is to supply supporting for the integration between manual and automatic steps. Moreover, analyze the necessity to use proprietary tools to perform any steps in the testing process. These tools have, usually, a short life time and may be less usability.

Check if the MBT approach supplies tools to support the testing process, preferably all steps. Verify if there is no additional cost to use these tools, e.g. license fee. If these tools don’t support all steps, analyze the effort, cost and complexity to perform this task manually.

4.1.5 The skill and complexity to use any approach must be understood for all stakeholders

The skill required for testing is changing year-by-year. Previously, testers needed to learn just testing techniques to perform testing activities. Currently, testers need to know testing techniques, modeling languages and sometimes programming languages.

Each MBT approach requires from the tester expertise about the notation used for software modeling, and sometimes expertise about testing criteria, testing metrics, or languages to generate test scripts.

The first step to apply one MBT approach *in loco* must be an explanation for all stakeholders about the process for test case generation, the inputs and outputs, testing criteria, and level of automation. This is fundamental because one MBT approach may involve different roles in the software development process that may impact the current activities. For example: the model used as input for test case generation may be generated by the designer, and there are some constraints required on the model to apply the MBT approach. Then, the designer need to know how to do this task and what the importance of his/her is task for the test case generation process.

Construct basic training material during the development of the approach to be used to support training of stakeholders and its implantation on the field. To include a MBT as part of the software development process may to help to decrease effort, cost, and time for a project, but requires a commitment of all project team, not just the testing team.

Analyze the skill required to apply a MBT approach, and if there are qualified professionals to work with this approach in our organization. Else, analyze the cost of training, if is necessary to involve other stakeholders (analysts, designers, programmers) and if these stakeholders can and are disposed to perform this new tasks. Expressive change in the current tasks may make difficult the implantation of a MBT approach. The ideal solution is to introduce low changes in the current tasks.

4.1.6 The order of steps to be followed while using a MBT approach

Each approach defines some steps for test case generation process. It’s necessary to know about them, their pre-requirement, pos-condition, and constraints. The steps must be performed in the same sequence as defined originally by the authors. Otherwise, wrong outputs may be generated.

Create easy steps, with well-defined inputs and outputs. Moreover, try to automate them by algorithms or tools.

Analyze the steps that compose the approach. Check if our software project generates the requirements to start this approach, and observe constraints for each intermediate step. Usually, MBT approaches have a strict process to be followed that may require changes in the current software testing process in a software organization.

4.1.7 Technology transfer

MBT Technologies transfer from academic environment to be implemented in industrial projects needs to be analyzed carefully before to be performed. Several issues must be analyzed, which are presented in this section. Risks and impacts must also be prevented. The researchers must support the transfer by experimental and controlled studies to adapt the approach to the industrial environments characteristics.

Execute experimental studies in different levels to evaluate the performance of a MBT approach in different scenarios, like *in virtuo*, *in vitro*, *in vivo* experiments (TRAVASSOS and BARROS, 2003). These studies help in the characterization, quality analysis, and observation of limitations and context of a MBT approach. Moreover, develop training material (like slides, demos, or tools manual) to support this task.

Analyze the existence of experimental results evaluating a MBT approach, or publications based on the application on the field. This information may help to reduce risks to apply a new approach in an organization. Apply this approach firstly for small and controlled projects before to use it in large and complex projects. This is useful to familiarize stakeholders with the new testing process, and may allow using the current and new testing approaches together for comparison. Ask for support of the researcher to transfer an approach to real projects. To apply a new MBT approach in a software project without supporting from its developers is an impracticable task.

4.2 Future research topics in Model-based Testing

Three open questions from current research are suggested for future work.

In this section, we discuss there, their motivation, feasibility, examples, and expected results. The questions are:

1. What are the contexts (testing levels, software domains, etc) in which Model-based Testing strategies have not been applied? And why?
    - What are the limitations about the model-based testing approaches described on the technical literature? How to avoid them?
2. How are non-functional requirements descriptions used for test cases generation?
    - How can a model describe software in order to efficiently generate test cases for non-functional requirements evaluation?
    - How to deal with different types of non-functional requirements?
3. What are the defects types not detected by test cases generated from specific model?
    - How to establish the relation between defects and their cause (phase, model, or artifact)?
    - Which types of defects continue to occur after the software deployment that was not detected by MBT (e.g. Domain application error, Inconsistency, Omission)?
    - Which types of defects are more efficiently detected by MBT?

4.2.1 What contexts (testing levels, software categories) have Model-based Testing strategies not been applied? Why?

Traditional testing techniques (not based on models) are published on the technical literature for several software domains and testing levels. However, MBT approaches have not been frequently applied for some contexts (Figure 10).

This research must investigate these contexts that MBT approaches have not been applied and analyze what are the reasons/characteristics/limitations about these context that prevent the using of MBT approaches.

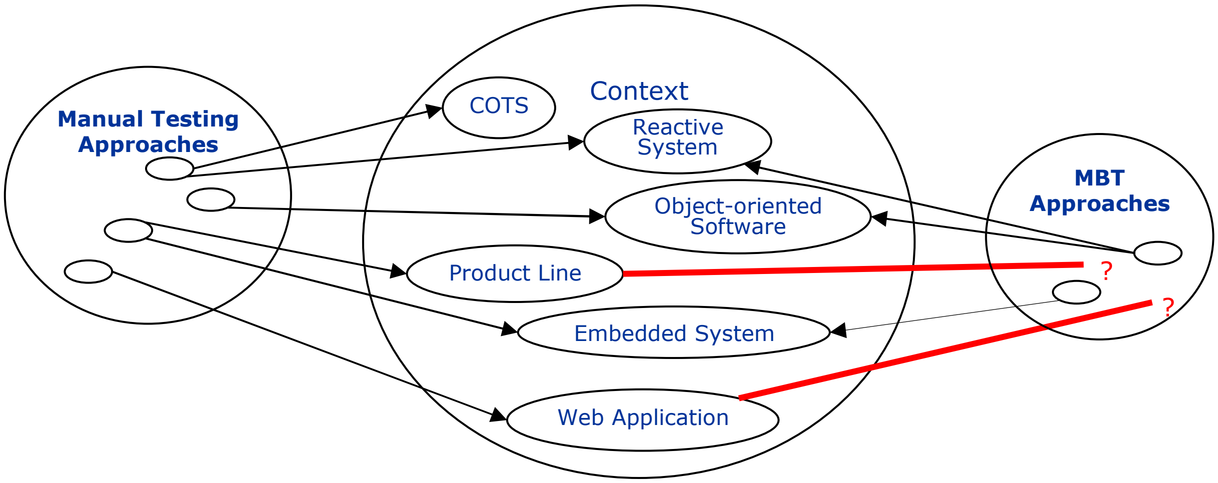

Figure 10. **MBT Approaches have not been applied for specific context.**

After this analysis, we may be able to create solution to test these software domains using a MBT strategy in different levels.

From these questions, we have defined an abstract about this research.

> ➢ ABSTRACT:
>
> *This research analyzes the contexts in which MBT has not been frequently applied. Subsequently, we may develop a strategy to define (or adapt) one approach to be applied in one of these contexts based on its specific characteristics. To perform that, we need to know what characteristics are important in a MBT approach, and how to relate the MBT characteristics with the specific characteristics of the software domain and the testing level that the approach to be developed will be applied.*

We have extracted some issues from the obtained results that may be used as starting point for the research that evaluates MBT for different contexts. The issues are presented below:

> ➢ ISSUES:
>
> - *How to create a single MBT approach for use at different testing levels during the entire software development process?*
> - *How to reduce the test case set without decreasing the testing coverage (i.e. eliminating redundant test cases)?*

- *What are the main characteristics of software domains that MBT approaches have not been frequently applied to? These domains include Product Line Software, COTS, Web Service, and Web application). How to test applications in these software domains using a MBT approach?*
- *What models may be used to describe the specific characteristics for the above domains? What are testing coverage criteria to be applied for applications in each of the above domains?*
- *Why tool usage is for MBT based Integration Testing low? And how increase MBT tools for Integration Testing?*

These issues have similar purposes. Therefore, the guidelines towards a solution for them could be generalized in some steps:

➢ GUIDELINES:

- *Observe the mains characteristics for each context where MBT has not been frequently applied yet. For example:*
  - *What are the specific characteristics of Web Application?*
  - *How to model a Web Application?*
  - *How to test a Web Application?*
- *Propose or identify an interesting method or approach to test an application in this context. For example:*
  - *Identification of approach(es) to test web application;*
- *Define how to use an approach based on model to test an application in this context. For example:*
  - *Creation or adaptation of a Web Application Testing Technique for MBT.*
- *Execution of Experimental Studies to evaluate the effectiveness of this solution for different characteristics, like effort, time, coverage, cost.*
  - *Controlled Experimental and On the Field Studies;*

4.2.2 How are non-functional requirements descriptions used for test cases generation?

Software Requirements Specification document consists of Functional (Figure 11A) and Non-functional (Figure 11B) requirements (IEEE 830, 1998). Functional requirements describe the software components or modules to be developed during the software design and coding phases (Figure 11C) and to be used to test case generation for system or integration level testing (Figure 11D). Non-functional requirements are essentials to define architectural aspects and constraints to be addressed during the software development (Figure 11E). Both need to be modeled and tested.

However, the majority of MBT approaches use only functional requirements during test case generation process. Non-functional requirement descriptions have not been frequently used for test case generation (Figure 11F). This information was analyzed in the section 3.1.2.

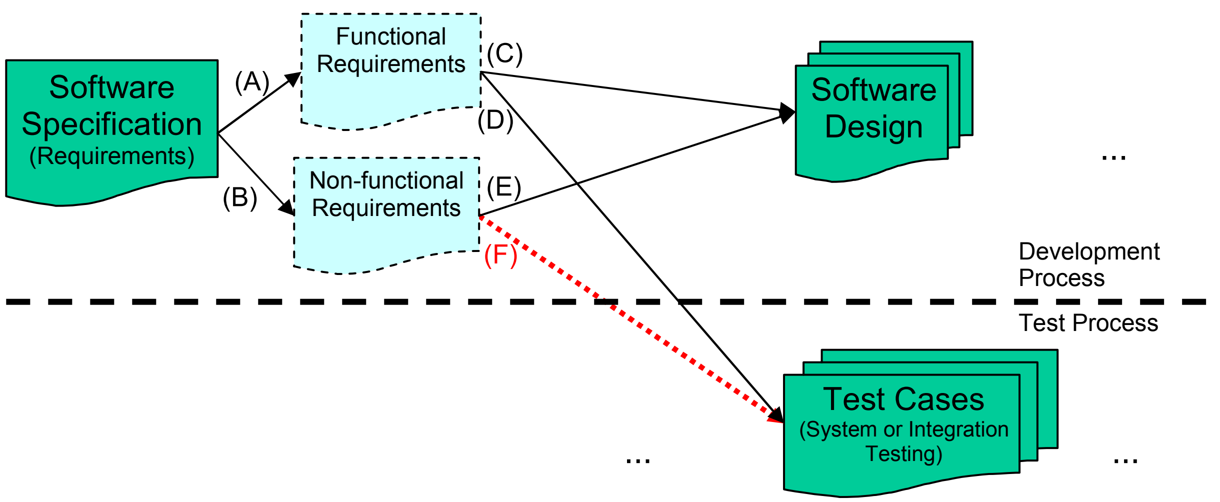

Figure 11. Usability of Functional and Non-functional Requirements in the software development and MBT process.

We propose to investigate test in different types of non-functional requirements by MBT. The main question is how to generate test cases from non-functional requirements description?

This research need to define a classification of non-functional requirements, using ISO-9126 (2001), for example, and propose a strategy to testing different types of non-functional requirements for different contexts (non-functional requirements have different behavior for different contexts).

From these questions, we have defined an abstract about this research.

> ➢ ABSTRACT:
>
> *This research discusses about the developing of an approach for test cases generation for different type of non-functional requirements for different software domain (according with some standard, like ISO or IEEE. E.g.: usability, efficiency, performance, security, etc).*

Some issues based on to use non-functional requirements descriptions for MBT have been extracted. These issues are presented below:

> ➢ ISSUES:
>
> - *How to classify non-functional requirement for a specific software domain?*
> - *How to model quality attributes for application in a specific software domain?*
> - *Which model to use for non-functional requirement behavior modeling?*
> - *How to measure the testing coverage about non-functional requirement?*
> - *Which type of non-functional requirement can be tested by MBT approaches? (Eventually, some attributes cannot be modeled).*

Using ISO-9126 (2001) as standard to classify non-functional requirement (as described on the section 3.1.2), we have defined some examples of non-functional requirements for different software domain, as described below:

> ➢ EXAMPLES OF NON-FUNCTIONAL REQUIREMENTS (ISO 9126)
>
> - *Usability*
>   - *Are the defined styles being applied in a web application?*
>   - *What was the defined screen configuration? (480x600; 800x600; 1024x768). Is the correct configuration being used?*
> - *Reliability*
>   - *Is the software running 24h a day, 7 days a week?*
>   - *Has the software a mechanism to treat fault tolerance?*
> - *Efficiency*

Does the software response to one action in the time defined on the requirements?

- *Portability*
  - *Can the software (web) be used on MS Explorer and Firefox?*
- *Security*
  - *Is the software prepared to avoid attacks like "sql-injection"?*
  - *One web application must not transport information in the URL (using GET. Post must be used). Is the software in accordance with this constraint?*
- *Maintainability*
  - *Model-based Testing Approaches support the software maintenance (regression testing), but how to evaluate its maintainability?*

These issues are integrated with the purpose of make able the using of non-functional requirements for a MBT solution. Therefore, the guidelines towards this solution could be described in some steps:

➢ GUIDELINES:

- *Define the context that non-functional requirements descriptions will be used for MBT. For example:*
  - *Which software domain? Web Application, Embedded System, etc.*
  - *Which testing level? System Testing, Integration Testing, etc.*
- How to classify non-functional requirements in the selected software domain? For example:
  - *How to classify types of non-functional requirements for Web Application?*
- How to test non-functional requirement for the selected software domain? For example:
  - *How to test different types of non-functional requirement for Web Application?*
- Which model to use for non-functional requirements modeling? For example:
  - *How to model a specific type of non-functional requirement for Web Application?*
- *Propose a method to test types of non-functional requirements for a software domain. For example:*
  - *Define a MBT approach to test different types of non-functional requirements for*

web application;

- *Execution of Experimental Studies to evaluate the effectiveness of this solution based on different characteristics, like effort, time, coverage, and cost.*
  - *Controlled Experimental and On the Field Studies;*

SIEMENS had developed a *Usability* approach to checking if one web application has been developed according to the SIEMENS' styles guide using ***WebChecker***.

4.2.3 What are the defects types not detected by the test cases generated from a specific behavior model?

Tests can show failures in the software, but not the absence of failures. The defects could remain in the software after the tests.

Software development is a human-task and consequently errors may be made during this process (HOWDEN, 1987). Testing try to show fails in a product to support the elimination of the defects that start these fails. However, to reach 100% of testing coverage is a hard and impracticable task, since we had to test the universe of all possible inputs (Figure 12).

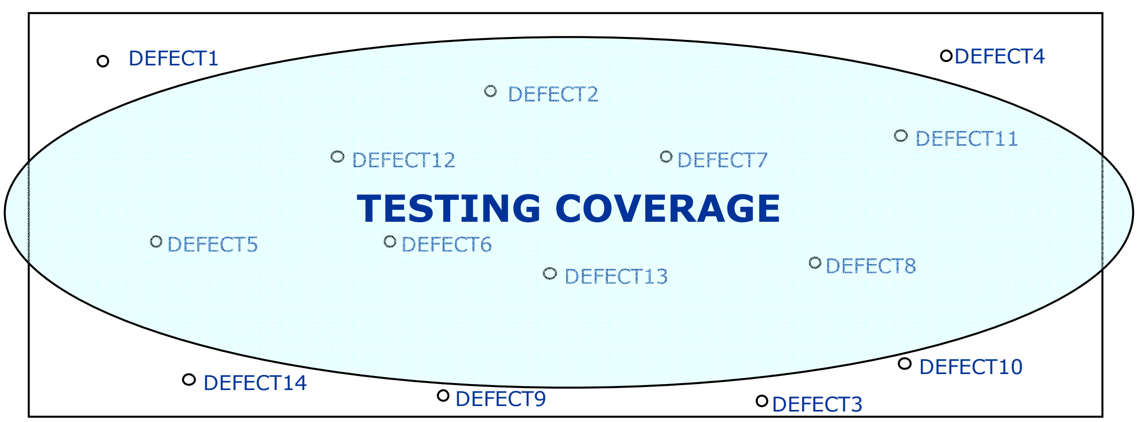

Figure 12. Behavior of testing coverage criteria applied to try to discover defects in software.

Testing coverage criteria aim to define strategies to assure an acceptable quality level for software without to reach 100% of coverage. However, eventually, some types of defects don't get to be discovered by a testing approach. This problem is more frequently in approaches based on models, since that they are dependent on the model used to describe the software behavior, and sometimes these models have limited testing coverage.

We want to discover types of defects not detected by MBT approaches and to propose improvements on the testing process to avoid these defect types in future projects. This solution may be using a MBT solution or not. The goal is to increase testing coverage, and also eventually decrease effort, time, and cost to use it.

To perform this task, we need to know results based on projects that MBT approaches have been applied. To do this, we should apply a causal-analysis and resolution approach (CARD, 2005) to establish the relation between defects and their cause (e.g.: the phase, model or artifact that the defect was discovered)? Subsequently, new strategies may be defined to avoid this type of defects in future projects, using a MBT approach or not.

From these questions, we have defined an abstract about this research.

- ABSTRACT:

This research discusses about the improvement of a MBT approach or just a testing strategy using information extracted from software projects. We need to apply a causal-analysis and resolution approach to establish the relation between defects and their cause in an organization and to evaluate the application of a MBT approach in a real project. Examples of information to be analyzed from the projects results are:

- *In which phase these defects have been included on the software artifact.*
- *What type of information (or behavior) the models are not able to represent.*
- *Which types of defects are not found by MBT approaches.*

From this information, we can purpose a new strategy to improve the testing process and to avoid these types of defects not found by MBT approaches in future projects.

Issues based on this research topic have been extracted. These issues are presented below:

- ISSUES:

- *What were the types of defects detected using a MBT approach?*

- *What were the types of defects not detected using a MBT approach, but detected after the deployment?*
- *How to extract metrics from the software development process about the MBT approach used during it?*
- *How is the defects distribution per phase or per type of defects?*
- *What are the origins of these defects? (Limitations of the model? Testing criteria? Tool? Testing process? Modeling error?)*
- *How to avoid these types of defects in future projects using a MBT approach or other strategy to support the testing?*

These issues involve a strategy to support the evaluation and improvement of MBT approaches. Other purpose is to support the technology transfer from academic environment to industry. The guidelines towards this solution could be described in some steps:

➢ GUIDELINES:

- *Define metrics to be extracted from a software development projects about MBT approaches. For example:*
  - *What is the testing coverage?*
  - *How many defects were detected using the MBT approach?*
  - *How many defects were detected not using the MBT approach?*
- *Collect information about MBT approaches in software projects to be analyzed. For example:*
  - *Are there empirical results describing metrics based on the quality and application of MBT approaches available on the technical literature?*
- *Develop a causal-analysis and resolution approach to know the origin of these defects in different approaches. For example:*
  - *How to define the origin of a defect from the project data?*
  - *How to propose a resolution for an identified problem to avoid it in future projects?*
- *Analyze the types of defect usually not identified by MBT approaches and propose a*

solution (based on model or not) to avoid them. For example:

- *How to identify <u>Software Domain</u> error using a MBT approach?*
- *How to identify <u>Inconsistence</u> error between parts of the Software Requirement using a MBT approach?*
- *How to identify <u>Logic</u> error using a MBT approach?*

- *Execution of Experimental Studies to evaluate the improvement of the new developed strategy.*
  - *Controlled Experimental and On the Field Studies;*

5 Conclusions

5.1.1 Summary

This work presented a survey for characterization of MBT approaches described on the technical literature using a Systematic Review protocol.

This survey was developed to capture the state-of-art of MBT approaches toward future research regarding MBT. It described the mains information related to several MBT approaches, and analyzed them to construct a body of knowledge of MBT. The expected results of this work are:

- The identification of MBT approaches, and research groups working with this field. This information will be very useful during the progress of this work.
- The identification of possible future research topics to be explored.

In the next sections, the contributions of this work are presented, some limitations are discussed, and future works are described.

5.1.2 Contributions

In this section, the mains contributions obtained during this work are presented:

SYSTEMATIC REVIEW. The execution of a systematic review, mainly the planning phase, when used as a tool on a bibliographical review allows a more complete specification about what has to be searched on technical sources, facilitating the search and improving the researcher's focus during the read and analysis of the publications. For this topic, the contributions are:

- Definition of a Systematic Review Protocol to guide the planning, execution and analysis of the survey. Moreover, this protocol allows the repetition or re-execution of this search by other researchers in different moments and source (digital libraries). If this protocol were followed, the results can be integrated, like a meta-analysis.

- The using of JabRef Tool to support the papers characterization/filtering/analysis and creation of a database with information about each paper collected. This database is available with the authors of this work.

MBT APPROACHES ANALYSIS. The main contribution of this work is related to the MBT field. More than 400 MBT papers have been found during this work (50% has been excluded because they are not related to propose of this work and 20% has been already analyzed) and the results have been presented in this document. For this topic, the contributions are:

- The obtained results showing analysis for the main characteristics about MBT approaches (like behavior model, testing coverage, automation, and limitations);
- The identification of some contexts that MBT has not been frequently applied;
- Description of observation based on MBT, supporting researcher, during the development of new MBT approaches, and practitioners, during the selection of a MBT to be applied in a software project.
- Motivations for new research regarding MBT. These proposals have been described on the section 4.

5.1.3 Limitations

Some limitations are presents in this work and they are related to the planning of survey, execution, and analysis. The identified limitations are:

- Only 9 sources of papers have been selected in this Systematic Review. We cannot assure that the identified papers represent a high coverage of papers regarding MBT approaches. The analysis performed cannot be generalized for all MBT approaches, but just for the context of the identified papers. For example, probably a lot papers published after 2004 have not been found because they are not available on the sources used during this work.
- In 406 papers identified during this work, 57 are not available on Internet. These paper had to be excluded of the characterization and analysis phases because was impossible to extract their information.

- The set of identified paper was large. Then, we have created three Priority Levels to define the analysis order for the papers. However, just the Priority Levels 1 and 2 have already been analyzed (20% of all papers). The papers in the Priority Level 3 must be analyzed in other moment to complete this survey.

5.1.4 Future Work

Some tasks can be suggested as future activities with the purpose of to continue this work. They are:

- Perform the characterization and analysis of the papers in Priority Level 3. These papers have not been analyzed yet;
- Look for other papers sources to be used to identify new papers regarding MBT;
- Explore new research topics for future work;
- Select one (or more) of the topics proposed in the section 4 of this work and explore it (or them) aiming a new MBT research for Master or PhD degree.

6 Bibliography References

ABDURAZIK, A. & OFFUT, J. (2000), "Using UML Collaboration Diagrams for Static Checking and Test Generation", Proceedings of UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Springer, pp. 383--395.

ALI, S.; ur REHMAN, M.J.; BRIAND, L.C.; ASGHAR, H.; ZAFAR, Z.; NADEEM, A. (2005), "A State-based Approach to Integration Testing for Object-Oriented Programs", Technical report, Carleton University.

AMLA, N. & AMMANN, P. (1992), "Using Z specifications in category partition testing", Proceedings of the COMPASS 7th Annual Conference on Computer Assurance (COMPASS '92), Gaithersburg, MD, pp. 15--18.

AMMANN, P. & OFFUT, J. (1994), "Using formal methods to derive test frames in category-partition testing", Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS '94) - Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security, pp. 69--79.

ANDREWS, A.; OFFUT, J. & ALEXANDER, R. (2005), "Testing web applications by modeling with FSMs", Systems and Modeling, 4(3):326-345, July 2005.

BARBEY, S.; BUCHS, D. & PÉRAIRE, C. (1996), "A theory of specification-based testing for object-oriented software", Proceedings of the Second EDCC2 (European Dependable Computing Conference), Taormina, Italy, October 1996.

BASANIERI, F. & BERTOLINO, A. (2000), "A Practical Approach to UML-based Derivation of Integration Tests", Proceedings of 4th International Software Quality Week Europe.

BEIZER, B. (1990), Software testing techniques, 2^{nd} ed., Van Nostrand Reinhold Co., New York, NY.

BELLENTINI, C.; MARCHETTO, A. & TRENTINI, A. (2005), "TestUml: user-metrics driven web applications testing", Proceedings of the 2005 ACM Symposium on

Applied Computing (SAC'05), New Mexico, March, 2005, pp. 1694-1698, ISBN 1-58113-964-0.

BERNARD, E.; BOUQUET, F.; CHARBONNIER, A.; LEGEARD, B.; PEUREUX, F.; UTTING, M. & TORREBORRE, E. (2006), “Model-Based Testing from UML Models”, Model-based Testing Workshop (MBT'2006), INFORMATIK'06, volume P-94 of LNI, Lecture Notes in Informatics, Dresden, Germany, pp. 223--230, October 2006. Note: ISBN 978-3-88579-188-1.

BERTOLINO, A.; GNESI, S. (2003), “Use case-based testing of product lines”. Proceedings of the ESEC / SIGSOFT FSE, pp. 355-358.

BERTOLINO, A.; MARCHETTI, E. & MUCCINI, H. (2005), “Introducing a Reasonably Complete and Coherent Approach for Model-based Testing”, Electronic Notes in Theoretical Computer Science 116(SPECISS), 85 - 97.

BERTOLINO, A.; MARCHETTI, E. & POLINI, A. (2003), “Integration of ‘components’ to test software components”, Electronic Notes in Theoretical Computer Science 82(6), 49 - 59.

BEYER, M.; DULZ, W. & ZHEN, F. (2003), “Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains”, Proceedings of 12th Asian Test Symposium (ATS 2003), pp. 102--105.

BIOLCHINI, J.; MIAN, P.G.; NATALI, A.C.; & TRAVASSOS, G.H. (2005), “Systematic Review in Software Engineering: Relevance and Utility”, Technical Report ES-679/05, PESC-COPPE/UFRJ. Available at http://www.cos.ufrj.br.

BIRD, D. L. & MUNOZ, C. U. (1983), “Automatic generation of random self-checking test cases”, IBM Systems Journal, 22(3):229–245.

BOTASCHANJAN, J.; PISTER, M. & RUMPE, B. (2004), 'Testing agile requirements models', Journal of Zhejiang University: Science 5(5), 587 - 593.

du BOUSQUET, L.; OUABDESSELAM, F.; RICHIER, J. & ZUANON, N. (1999), “Lutess: a specification-driven testing environment for synchronous software”, Proceedings of the 1999 International Conference on Software Engineering (ICSE1999), pp. 267--276.

(a) BRIAND, L.; LABICHE, Y. & CUI, J. (2004), "Towards Automated Support for Deriving Test Data from UML Statecharts", (SCE-03-13, Version 2), Technical report, Carleton University.

(b) BRIAND, L.; LABICHE, Y. & LIN, Q. (2004), "Improving State-Based Coverage Criteria Using Data Flow Information", Technical report, Carleton University.

BRIAND, L.; LABICHE, Y. & SOCCAR, G. (2002), "Automating impact analysis and regression test selection based on UML designs", Proceedings of International Conference on Software Maintenance (ICSM).

BRIAND, L.C.; LABICHE, Y. (2002), "A UML-Based Approach to System Testing", Technical report, Carleton University.

BRIAND, L.C.; LABICHE, Y. & SÓWKA, M.M. (2006), "Automated, contract-based user testing of commercial-off-the-shelf components", Proceeding of the 28th international conference on Software engineering (ICSE '06), ACM Press.

BRIAND, L.C.; LABICHE, Y.; WANG, Y. (2002), "Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles – An Investigation of Graph-based Class Integration Test Order Strategies", Technical report, Carleton University.

CARD, D. (2005), "Defect Analysis: Basic Techniques for Management and Learning", Advances in Computers, vol. 65, chapter 7, pp. 259-295.

CARPENTER, P.B. (1999), "Verification of requirements for safety-critical software", Proceedings of the 1999 annual ACM SIGAda international conference on Ada (SIGAda '99), ACM Press.

CAVARRA, A.; CRICHTON, C. & DAVIES, J. (2003), "A method for the automatic generation of test suites from object models", Proceedings of the 2003 ACM symposium on Applied computing.

CHANG, J. & RICHARDSON, D.J. (1999), "Structural specification-based testing: automated support and experimental evaluation", Proceedings of the 7th European software engineering conference, Springer-Verlag, ACM Press, pp. 285-302.

CHANG, J.; RICHARDSON, D.J. & SANKAR, S. (1996), “Structural specification-based testing with ADL”, Proceedings of the 1996 International Symposium on Software Testing and Analysis ISSTA, ACM Press.

CHEN, Y.; LIU, S. & NAGOYA, F. (2005), “An approach to integration testing based on data flow specifications”, Lecture Notes in Computer Science 3407, 235 - 249.

CHEN, Y.; PROBERT, R.L. & SIMS, D.P. (2002), “Specification-based regression test selection with risk analysis”, Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, IBM Press.

CHEVALLEY, P. & FOSSE, P.T. (2001), “Automated Generation of Statistical Test Cases from UML State Diagrams”, COMPSAC '01: Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, IEEE Computer Society, Washington, DC, USA, pp. 205--214.

CHOW, T.S. (1978), “Testing software design modeled by finite-state machines”, IEEE Trans. Software Eng., vol SE-4, pp. 178-187, Mar.

CRICHTON, C.; CAVARRA, A. & DAVIES, J. (2001), “Using UML for Automatic Test Generation”, Proceedings of Automated Software Engineering (ASE).

DALAL, S.; JAIN, A.; KARUNANITHI, N.; LEATON, J. & LOTT, C. (1998), “Model-based testing of a highly programmable system”, Proceedings of The Ninth International Symposium on Software Reliability Engineering, pp. 174--179.

DALAL, S.; JAIN, A.; KARUNANITHI, N.; LEATON, J.; LOTT, C.; PATTON, G. & HOROWITZ, B. (1999), “Model-based testing in practice”, Proceedings of the 1999 International Conference on Software Engineering (ICSE’99), May, pp. 285--294.

DENG, D.; SHEU, P. & WANG, T. (2004), “Model-based testing and maintenance”, Proceedings of IEEE Sixth International Symposium on Multimedia Software Engineering, pp. 278--285.

FRIEDMAN, G.; HARTMAN, A.; NAGIN, K. & SHIRAN, T. (2002), “Projected state machine coverage for software testing”, Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis (ISSTA '02), ACM Press.

GARGANTINI, A. & HEITMEYER, C. (1999), “Using model checking to generate tests from requirements specifications”, Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-7), Springer-Verlag, ACM Press.

GAROUSI, V.; BRIAND, L.C. & LABICHE, Y. (2006), “Traffic-aware stress testing of distributed systems based on UML models”, Proceeding of the 28th international conference on Software engineering (ICSE '06), ACM Press.

GNESI, S.; LATELLA, D. & MASSINK, M. (2004), “Formal test-case generation for UML statecharts”, Proceedings of Ninth IEEE International Conference on Engineering Complex Computer Systems, pp. 75--84.

GROSS, H.; SCHIEFERDECKER, I. & DIN, G. (2005), “Model-Based Built-In Tests”, Electronic Notes in Theoretical Computer Science 111(SPEC ISS), 161 - 182.

HARTMAN, A. & NAGIN, K. (2004), “The AGEDIS tools for model based testing”, ISSTA 2004 - Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, 129 - 132.

HARTMANN, J.; IMOBERDORF, C. & MEISINGER, M. (2000), “UML-based integration testing”, Proceedings of ISSTA’2000, pp. 60-70, Aug.

HARTMANN, A.; NAGIN, K. (2004), “The AGEDIS tools for model based testing”, Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004), 129 – 132.

HONG, H.S.; KIM, Y.G.; CHA, S.D.; BAE, D.H. & URAL, H. (2000), “A test sequence selection method for statecharts”, SOFTWARE TESTING VERIFICATION & RELIABILITY 10(4), 203--227.

HOWDEN, W.E. (1987), “Functional program testing and analysis”. Nova York, NY, McGrall-Hill.

IEEE Recommended Practice for Software Requirements Specifications (1998). ANSI / IEEE Std. 830-1998.

International Organization for Standardization (2001). ISO/IEC Standard 9126: Software Engineering -- Product Quality, part 1.

JURISTO, N.; MORENO, A.M.; VEGAS, S. (2004), "Reviewing 25 years of testing technique experiments". Empirical Software Engineering: An International Journal, 9(1), p. 7-44, March.

KANSOMKEAT, S. & RIVEPIBOON, W. (2003), "Automated-generating test case using UML statechart diagrams", Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology (SAICSIT '03), South African Institute for Computer Scientists and Information Technologists.

KIM, Y.; HONG, H.; BAE, D. & CHA, S. (1999), "Test cases generation from UML state diagrams", Software, IEE Proceedings- [see also Software Engineering, IEE Proceedings] 146(4), 187--192.

KITCHENHAM, B., "Procedures for Performing Systematic Review", Joint Technical Report Software Engineering Group, Department of Computer Science Keele University, United King and Empirical Software Engineering, National ICT Australia Ltd, Australia, 2004.

LEGEARD, B.; PEUREUX, F. & UTTING, M. (2004), "Controlling test case explosion in test generation from B formal models", Software Testing, Verification & Reliability', John Wiley and Sons Ltd.

LINZHANG, W.; JIESONG, Y.; XIAOFENG, Y.; Jun, H.; XUANDONG, L. & GUOLIANG, Z. (2004), "Generating test cases from UML activity diagram based on Gray-box method", Proceeding of 11th Asia-Pacific Software Engineering Conference (APSEC, 2004), pp. 284--291.

LIUYING, L. & ZHICHANG, Q. (1999), "Test selection from UML Statecharts", Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS1999), pp. 273--279.

LUCIO, L.; PEDRO, L. & BUCHS, D. (2005), "A methodology and a framework for model-based testing", Lecture Notes in Computer Science 3475, 57 - 70.

LUND, M.S. & STØLEN, K. (2006), “Deriving tests from UML 2.0 sequence diagrams with neg and assert”, Proceedings of the 2006 international workshop on Automation of software test (AST '06), ACM Press.

MANDRIOLI, D.; MORASCA, S. & MORZENTI, A. (1995), “Generating test cases for real-time systems from logic specifications”, ACM Transactions on Computer Systems (TOCS), ACM Press.

MYERS, G., “The Art of Software Testing”, New York: John Wiley, 1979.

MEYER, S. & SANDFOSS, R. (1998), “Applying Use-Case Methodology to SRE and System Testing”, STAR West Conference, pp. 1-16.

MINGSONG, C.; XIAOKANG, Q. & XUANDONG, L. (2006), “Automatic test case generation for UML activity diagrams”, Proceedings of the 2006 international workshop on Automation of software test (AST '06), ACM Press.

MURTHY, P.V.; ANITHA, P.C.; MAHESH, M. & SUBRAMANYAN, R. (2006), “Test ready UML statechart models”, Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools (SCESM '06).

NEBUT, C. & FLEUREY, F. (2006), “Automatic Test Generation: A Use Case Driven Approach”, IEEE Transaction Software Engineering, IEEE Press, Piscataway, NJ, USA, pp. 140--155.

OFFUTT, A.J. & LIU, S. (1999), “Generating test data from SOFL specifications”, Journal of Systems and Software, 49(1), 49 - 62.

OFFUTT, J. & ABDURAZIK, A. (1999), “Generating tests from UML specifications”, UML’99 Unified Modeling Language.

OFFUTT, J.; LIU, S.; ABDURAZIK, A. & AMMANN, P. (2003), “Generating Test Data from State-Based Specifications”, Journal of Software Testing, Verification and Reliability, John Wiley & Sons, Ltd, pp. 25-53.

OLIMPIEW, E.M. & GOMAA, H. (2005), “Model-based testing for applications derived from software product lines”, Proceedings of the first international workshop on Advances in model-based testing (A-MOST '05).

OSTRAND, T. J. & BALCER, M. J. (1988), “The Category-Partition Method for Specifying and Generating Functional Tests”, Communications of the ACM, 31(6), June, pp. 676-686.

PARADKAR, A. (2004), “Plannable test selection criteria for FSMs extracted from operational specifications”, 15th International Symposium on Software Reliability Engineering (ISSRE 2004), pp. 173--184.

PARISSIS, I. & OUABDESSELAM, F. (1996), “Specification-based testing of synchronous software”, Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering (21), 127 - 134.

PRASANNA, M.; SIVANANDAM, S.N.; VENKATESAN, R.; SUNDARRAJAN, R. (2005), “Survey on Automatic Test Case Generation”, Academic Open Internet Journal, Volume 15, available at http://www.acadjournal.com/2005/v15/part6/p4/.

(a) PRETSCHNER, A.; LOTZBEYER, H. & PHILIPPS, J. (2001), “Model based testing in evolutionary software development”, Proceedings of the International Workshop on Rapid System Prototyping, 155 - 160.

(b) PRETSCHNER, A.; SLOTOSCH, O.; AIGLSTORFER, E. & KRIEBEL, S. (2001), “Model-based testing for real: the inhouse card case study”, Proceedings of International Journal on Software Tools for Technology Transfer (STTT), Springer-Verlag, pp. 140-57.

PRETSCHNER, A.; LOTZBEYER, H.; PHILIPPS, J. (2004), “Model based testing in incremental system development”; Journal of Systems and Software, 70, pp. 315 – 329.

PRETSCHNER, A. (2005), “Model-based testing”, Proceedings of 27th International Conference on Software Engineering, (ICSE 2005), pp. 722-723.

RAMAMOORTHY, C. V.; HO, S. F. & CHEN, W. T. (1976), “On the automated generation of program test data”, IEEE Transactions on Software Engineering, SE-2(4):293–300, Dec.

RAPPS, S. & WEYUKER, E.J. (1982), “Data Flow analysis techniques for test data selection”, In: International Conference on Software Engineering, pp. 272-278, Tokio, Sep.

RICHARDSON, D.J.; LEIF AHA, S. & O'MALLEY, T.O. (1992), “Specification-based test oracles for reactive systems”, Proceedings of International Conference on Software Engineering (ICSE’1992), Vol. 14, 105 - 118.

RICHARDSON, D.J. & WOLF, A.L. (1996), “Software testing at the architectural level”, Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, ACM Press.

RIEBISCH, M.; PHILIPPOW, I. & GOTZE, M. (2002), “UML-based statistical test case generation”, International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, Springer-Verlag.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C., “Qualidade de software – Teoria e prática”, Prentice Hall, São Paulo, 2001.

RUMPE, B. (2003), “Model-based testing of object-oriented systems”, FORMAL METHODS FOR COMPONENTS AND OBJECTS 2852, 380--402.

SATPATHY, M.; LEUSCHEL, M. & BUTLER, M. (2005), “ProTest: An Automatic Test Environment for B Specifications”, Electronic Notes in Theoretical Computer Science 111 (SPEC ISS), 113 - 136.

SCHEETZ, M.; von MAYHAUSER, A.; FRANCE, R.; DAHLMAN, E. & HOWE, A.E. (1999), “Generating Test Cases from an OO Model with an AI Planning System”, Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE '99), IEEE Computer Society, Washington, DC, USA, pp. 250.

SINHA, A. & PARADKAR, A. (2006), “Model-based functional conformance testing of web services operating on persistent data”, Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications (TAV-WEB '06), ACM Press.

SOKENOU, D. (2006), “Generating Test Sequences from UML Sequence Diagrams and State Diagrams”, Informatik Forschung und Entwicklung, pp. 236-240.

STOBIE, K. (2005), “Model Based Testing in Practice at Microsoft”, Electronic Notes in Theoretical Computer Science 111(SPEC ISS), 5-12.

(a) STOCKS, P.A. & CARRINGTON, D.A. (1993), “Test templates: a specification-based testing framework”, Proceedings of the 15th international Conference on Software Engineering (ICSE’1993), IEEE Computer Society Press.

(b) STOCKS, P. & CARRINGTON, D. (1993), “Test template framework: a specification-based testing case study”, Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis (ISSTA '93), ACM Press.

STOCKS, P. & CARRINGTON, D. (1996), “A Framework for Specification-Based Testing”, IEEE Transactions on Software Engineering, IEEE Press.

TAHAT, L.; VAYSBURG, B.; KOREL, B. & BADER, A. (2001), “Requirement-based automated black-box test generation”, 25th Annual International Computer Software and Applications Conference (COMPSAC 2001), pp. 489--495.

TAN, L.; SOKOLSKY, O. & LEE, I. (2004), “Specification-based testing with linear temporal logic”, Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration (IRI 2004), pp. 493--498.

TRAORÉ, I. (2003), “A transition-based strategy for object-oriented software testing”, Proceedings of the 2003 ACM symposium on Applied computing, ACM Press.

TRAVASSOS, G. H.; BARROS, M. O. (2003), “Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering”. In: 2nd Workshop in Workshop Series on Empirical Software Engineering: The Future of Empirical Studies in Software Engineering, Roma. Proceedings of the WSESE03.

VIEIRA, M.; LEDUC, J.; HASLING, B.; SUBRAMANYAN, R. & KAZMEIER, J. (2006), “Automation of GUI testing using a model-driven approach”, Proceedings of the 2006 international workshop on Automation of software test (AST '06), ACM Press.

WU, Y.; CHEN, M. & OFFUTT, J. (2003), “UML-Based Integration Testing for Component-Based Software”, Proceedings of the Second International Conference on COTS-Based Software Systems (ICCBSS '03), Springer-Verlag, London, UK, pp. 251--260.

XU, W. & XU, D. (2006), “State-based testing of integration aspects”, Proceedings of the 2nd workshop on Testing aspect-oriented programs (WTAOP '06), ACM Press.

ZHEN, R.D.; GRABOWSKI, J.; NEUKIRCHEN, H. & PALS, H. (2004), “Model-based testing with UML applied to a roaming algorithm for Bluetooth devices”, Journal of Zhejiang University Science, pp. 1327-35.

Appendix A – List of Identified Papers

Category	Title	Authors	Source	Year
B	A case for test-code generation in model-driven systems	Rutherford and Wolf	GPCE	2003
B	A Choice Relation Framework for Supporting Category-Partition Test Case Generation	Chen *et al.*	IEEE Transactions on Software Engineering	2003
E	A flexible environment to evaluate state-based test techniques	Hierons	ACM SIGSOFT Software Engineering Notes	2004
B	A formal approach to requirements based testing in open systems standards	Leathrum and Liburdy	International Conference on Requirements Engineering	1996
E	A framework and tool support for the systematic testing of model-based specifications	Miller and Strooper	TOSEM	2003
D	A framework for specification-based class testing	Liu *et al.*	ICECCS	2002
D	A Framework for Specification-Based Testing	Stocks and Carrington	IEEE Transactions on Software Engineering	1996
NC	A framework for table driven testing of Java classes	Daley *et al.*	Software—Practice & Experience	2002
D	A Generic Model-Based Test Case Generator	Popovic and Velikic	ECBS	2005
B	A holistic approach to test-driven model checking	Belli and Guldali	International conference on Innovations in Applied Artificial Intelligence	2005
E	A hybrid component-based system development process	Teiniker *et al.*		
A	A method for the automatic generation of test suites from object models	Cavarra *et al.*	ACM symposium on Applied computing	2003
B	A method of generating massive virtual clients and model-based performance test	Kim	QSIC	2005
A	A methodology and a framework for model-based testing	Lucio *et al.*	Lecture Notes in Computer Science	2005
NC	A Methodology of Verification and Testing of Large Software Systems	Lipaev	Programming and Computing Software	2003
E	A model-based statistical usage testing of communication protocols	Popovic *et al.*	International Symposium and Workshop on Engineering of Computer Based Systems	
B	A model-to-implementation mapping tool for automated model-based GUI testing	Paiva *et al.*	ICFEM	2005
B	A new approach to test case generation based on real-time process algebra (RTFA)	Yao and Wang	Canadian Conference on Electrical and Computer Engineering	2004
E	A practical approach to modified condition/decision coverage	Hayhurst and Veerhusen	DASC	2001
C	A Practical Approach to UML-based Derivation of Integration Tests	Basanieri and Bertolino	QWE	2000
E	A reliability estimator for model based software testing	Sayre and Poore	ISSRE	
E	A rigorous method for test templates generation from object-oriented specifications	Periyasamy and Alagar	Software Testing Verification and Reliability	2001
E	A schema language for coordinating construction and composition of partial behavior descriptions	Grieskamp and Kicillof	SCESM	2006
E	A specification driven hierarchical test methodology	Sathianathan and Smith	IEEE International ASIC Conference and Exhibit	
D	A specification-based adaptive test case generation strategy for open operating system standards	Watanabe and Sakamura	ICSE	1996
NC	A specification-based case study from test class framework	Liu and Miao	Journal of Shanghai University	2001
C	A State-based Approach to Integration Testing for Object-Oriented Programs	Ali *et al.*	Technical Report of Carleton University	2005
NC	A study of user acceptance tests	Leung and Wong	Software Quality Control	1997
NC	A test data generation tool based on inter-relation of fields in the menu structure	Lee and Choi	Journal of KISS: Computing Practices	2003
E	A Test Generation Strategy for Pairwise Testing	Tsuchiya and Kikuno	TOSEM	2002
B	A test sequence selection method for statecharts	Hong *et al.*	Software Testing Verification & Reliability	2000
D	A theory of specification-based testing for object-oriented software	Barbey *et al.*	EDCC	1996

E	A tool for testing synchronous software	Parissis	Achieving Quality in Software	
A	A transition-based strategy for object-oriented software testing	Traore	ACM symposium on Applied computing	2003
A	A UML-Based Approach to System Testing	Briand and Labiche	Technical Report of Carleton University	2002
NC	A use case driven testing process: towards a formal approach based on UML collaboration diagrams	Badri *et al.*	Formal Approaches to Software Testing	2004
E	Action machines - towards a framework for model composition, exploration and conformance testing based on symbolic computation	Grieskamp *et al.*	QSIC	2005
B	Action refinement in conformance testing	Van Der Bijl *et al.*	Lecture Notes in Computer Science	2005
E	Activity based SW process as basis for ISO 9000	Mahabala	CSI Communications	1993
D	Adding natural relationships to simulink models to improve automated model-based testing	Boden and Busser	AIAA/IEEE Digital Avionics Systems Conference	2004
D	ADLscope: an automated specification-based unit testing tool	Chang and Richardson	ASE	1998
NC	An adaptive use case design driven testing	Kim *et al.*	ISCA	
E	An analysis and testing method for Z documents	Ciancarini *et al.*	Annals of Software Engineering	1997
E	An analysis of rule coverage as a criterion in generating minimal test suites for grammar-based software	Hennessy and Power	ASE	2005
E	An Analysis of Test Data Selection Criteria Using the RELAY Model of Fault Detection	Richardson and Thompson	IEEE Transactions on Software Engineering	1993
B	An approach to detecting domain errors using formal specification-based testing	Chen and Liu	Asia-Pacific Software Engineering Conference	2004
NC	An approach to generate integration test cases based on UML collaboration diagrams	zhang, W.; dong, L. & liang, Z.	Acta Electronica Sinica	2004
D	An approach to integration testing based on data flow specifications	Chen *et al.*	Lecture Notes in Computer Science	2005
E	An approach to specification-based testing systems	Zin *et al.*	Software Quality Engineering	1997
E	An approach to verification and validation of a reliable multicasting protocol	Callahan and Montgomery	ISSTA	1996
NC	An experimental evaluation of a higher-ordered-typed-functional specification-based test-generation technique	Sinha and Smidts	Empirical Software Engineering	2006
D	An explorative journey from architectural tests definition down to code tests execution	Bertolino *et al.*	ICSE	2001
E	An extended fault class hierarchy for specification-based testing	Lau and Yu	ACM Transactions on Software Engineering and Methodology	2005
NC	An extended finite state machine based generation method of test suite	Guo and Ping	Journal of Software	2001
E	An improved model-based method to test circuit faults	Cheng *et al.*	Theoretical Computer Science	2005
E	An integrated method for designing user interfaces based on tests	Schilling *et al.*	A-MOST	2005
NC	An object-based data flow testing approach for Web applications	Liu *et al.*	International Journal of Software Engineering and Knowledge Engineering	2001
D	An overview of Lutess: A specification-based tool for testing synchronous software	du Bousquet and Zuanon	ASE	1999
E	An overview of model-based testing	Utting	Technique et Science Informatiques	2006
D	Analyzing software architectures with Argus-I	Vieira *et al.*	ICSE	2000
E	Applicability of non-specification-based approaches to logic testing for software	Kobayashi *et al.*	International Conference on Dependable Systems and Networks	2001
NC	Application of software quality assurance methods in validation and maintenance of reactor analysis computer codes	Reznik	Reactor Physics and Reactor Computations	
D	Applying conventional testing techniques for class testing	Chung *et al.*	IEEE Computer Society's International Computer Software & Applications Conference	1996
NC	Applying extended finite state machines in software testing of interactive systems	Fantinato and Jino	Interactive Systems: Design, Specification, and Verification	2003

E	Applying models in your testing process	Rosaria and Robinson	Information and Software Technology	2000
NC	Applying mutation analysis to SDL specifications	Kovacs *et al.*	SDL 2003: System Design, Proceedings	2003
A	Applying Use-Case Methodology to SRE and System Testing	Meyer and Sandfoss	STAR West Conference	1998
NC	Approach to specification-based testing systems	Mohd Zin *et al.*	SQE	1997
E	Auto-generating test sequences using model checkers: a case study	Heimdahl *et al.*	Formal Approaches to Software Testing	2003
NC	Automated boundary testing from Z and B	Legeard *et al.*	FME	2002
E	Automated consistency and completeness checking of testing models for interactive systems	Paradkar and Klinger	International Computer Software and Applications Conference	2004
A	Automated Generation of Statistical Test Cases from UML State Diagrams	Chevalley and Fosse	COMPSAC	2005
E	Automated generation of test programs from closed specifications of classes and test cases	Leow *et al.*	ICSE	2004
NC	Automated generation of test scenarios based on UML specification	Nagy	Automatizace	2005
E	Automated model-based testing of Chi simulation models with TorX	Van Osch	Lecture Notes in Computer Science	2005
D	Automated Test Case Generation for Programs Specified by Relational Algebra Queries	Tsai *et al.*	IEEE Transactions on Software Engineering	1990
NC	Automated test case generation from IFAD VDM++ specifications	Nadeem and Jaffar-Ur-Rehman	WSEAS Transactions on Computers	2005
B	Automated test oracles for GUIs	Memon *et al.*	ACM SIGSOFT international symposium on Foundations of software engineering	2000
B	Automated testing from object models	Poston	Communications of the ACM	1994
D	Automated Testing of Classes	Buy *et al.*	ISSTA	2000
C	Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains	Beyer *et al.*	ATS	2003
B	Automated validation test generation	Weber *et al.*	DASC	1994
E	Automated verification and test case generation for input validation	Liu and Tan	AST	2006
C	Automated, contract-based user testing of commercial-off-the-shelf components	Briand *et al.*	ICSE	2006
A	Automated-generating test case using UML statechart diagrams	Kansomkeat and Rivepiboon	SAICSIT	2003
NC	Automatic construction of a class state-based testing model using method specifications	Al-Dallal and Sorenson	IASTED: International Conference on Computer Science and Technology	2003
D	Automatic extraction of abstract-object-state machines from unit-test executions	Xie *et al.*	ICSE	2006
E	Automatic generation of test cases from Boolean specifications using the MUMCUT strategy	Yu *et al.*	ASE	2006
E	Automatic Generation of Test Oracles—From Pilot Studies to Application	Feather and Smith	ASE	2001
A	Automatic test case generation for UML activity diagrams	Mingsong *et al.*	AST	2006
E	Automatic test generation for predicates	Paradkar *et al.*	ISSRE	1996
A	Automatic Test Generation: A Use Case Driven Approach	Nebut and Fleurey	IEEE Transaction Software Engineering	2006
NC	Automatic UML-based test data generating tool: AUTEG	Cheongah and Byoungju	Journal of KISS: Computing Practices	2002
D	Automatically testing interacting software components	Gallagher and Offutt	AST	2006
B	Automating formal specification-based testing	Donat	International Joint Conference CAAP/FASE on Theory and Practice of Software Development	1997
A	Automating impact analysis and regression test selection based on UML designs	Briand *et al.*	ICSM	2002
D	Automating software module testing for FAA certification	Santhanam	ACM SIGAda international conference on Ada	2001
E	Automating Specification-Based Software Testing	Poston	IEEE Computer Society Press	1997
NC	Automating test generation for discrete event oriented embedded systems	Cunning and Rozenblit	Journal of Intelligent and Robotic Systems	2005
A	Automation of GUI testing using a model-driven approach	Vieira *et al.*	AST	2006
NC	Axiomatic assessment of logic coverage software testing criteria	Liu and Miao	Ruan Jian Xue Bao/Journal of Software	2004

E	Behavior modeling technique based on EFSM for interoperability testing	Noh *et al.*	ICCSA	2005
NC	Behavior-based integration testing of software systems: a formal scenario approach	Pei *et al.*	International Conference on Systems Integration	
NC	Benefits of using model-based testing tools	Bruno *et al.*	Symposium on Software Quality	1995
D	Black-box testing using flowgraphs: An experimental assessment of effectiveness and automation potential	Edwards	Software Testing Verification and Reliability	2000
B	Boundary Coverage Criteria for Test Generation from Formal Models	Kosmatov *et al.*	ISSRE	2004
E	Building testable software	Zucconi and Reed	ACM SIGSOFT Software Engineering Notes	1996
E	Cause-effect graphing analysis and validation of requirements	Nursimulu and Probert	Conference of the Centre for Advanced Studies on Collaborative research	1995
D	Combining algebraic and model-based test case generation	Dan and Aichernig	Lecture Notes in Computer Science	2005
B	Combining behavior and data modeling in automated test case generation	Schroeder *et al.*	QSIC	2003
E	Comparison of fault classes in specification-based testing	Okun *et al.*	Information and Software Technology	2004
E	Confirming configurations in EFSM testing	Petrenko *et al.*	IEEE Transactions on Software Engineering	2004
B	Constructing multiple unique input/output sequences using metaheuristic optimisation techniques	Guo *et al.*	IEE Proceedings Software	2005
E	Constructing test suites for interaction testing	Cohen *et al.*	ICSE	2003
E	Continuous TTCN-3: testing of embedded control systems	Schieferdecker *et al.*	SEAS	2006
D	Controlling test case explosion in test generation from B formal models	Legeard *et al.*	Software Testing, Verification & Reliability'	2004
B	Coverage metrics for requirements-based testing	Whalen *et al.*	ISSTA	2006
D	Coverage-directed test generation with model checkers: challenges and opportunities	Devaraj *et al.*	COMPSAC	2005
E	Criteria for generating specification-based tests	Offutt *et al.*	ICECCS	1999
NC	DAS-BOOT: design-, architecture- and specification-based approaches to object-oriented testing	Richardson	ACM SIGSOFT Software Engineering	2000
B	Data abstraction and constraint solving for conformance testing	Calame *et al.*	APSEC	2005
E	Data flow testing as model checking	Hong *et al.*	ICSE	2003
NC	Data Generation for Path Testing	Mansour and Salame	Software Quality Control	2004
D	DeepTrans - a model-based approach to functional verification of address translation mechanisms	Adir *et al.*	4th International Workshop on Microprocessor Test and Verification	2003
E	Demonstration of an operational procedure for the model-based testing of CTI systems	Hagerer *et al.*	International Conference on Fundamental Approaches to Software Engineering	2002
B	Dependence analysis in reduction of requirement based test suites	Vaysburg *et al.*	International Symposium on Software Testing and Analysis	2002
E	Deriving operational software specifications from system goals	Letier and van Lamsweerde	ACM SIGSOFT symposium on Foundations of software engineering	2002
NC	Deriving test cases for composite operations in Object-Z specifications	Periyasamy *et al.*		
D	Deriving test plans from architectural descriptions	Bertolino *et al.*	ICSE	2000
A	Deriving tests from UML 2.0 sequence diagrams with neg and assert	Lund and Stølen	AST	2006
D	Design and implementation of Triveni: a process-algebraic API for threads + events	Colby *et al.*	International Conference on Computer Languages	1998
D	Developing a TTCN-3 test harness for legacy software	Okika *et al.*	AST	2006
D	Distributed software testing with specification	Chang et al.	IEEE Computer Society's International Computer Software & Applications Conference	1990
B	Domain specific test case generation using higher ordered typed languages for specification	Sinha and Smidts	University of Maryland at College Park	2005
E	Early estimation of defect density using an in-process Haskell metrics model	Mark Sherriff	A-MOST	2005
E	Economic perspectives in test automation: balancing automated and manual testing with opportunity cost	Ramler and Wolfmaier	AST	2006

E	Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations	Bishop *et al*.	Annual ACM Symposium on Principles of Programming Languages	2006
E	Enhanced testing of domain specific applications by automatic extraction of axioms from functional specifications	Sinha *et al*.	ISSRE	2003
B	Environment behavior models for scenario generation and testing automation	Auguston *et al*.	A-MOST	2005
B	Evaluating several path-based partial dynamic analysis methods for selecting black-box generated test cases	Chan and Yu	QSIC	2004
E	Experience With Teaching Black-Box Testing in a Computer Science/Software Engineering Curriculum	Chen and Poon	IEEE Transactions on Education	2004
E	Experimental Modal Analysis and Computational Model Updating of a Car Body in White	Schedlinski *et al*.	International Conference on Noise and Vibration Engineering	2004
NC	Extended model-based testing toward high code coverage rate	Takahashi and Kakuda	SOFTWARE QUALITYECSQ	2002
E	Extending Simulink Models With Natural Relations To Improve Automated Model-Based Testing	Boden *et al*.	SEW	2005
D	Extending test templates with inheritance	Murray *et al*.	ASWEC	1997
E	Fault classes and error detection capability of specification-based testing	Kuhn	ACM Transactions on Software Engineering and Methodology	1999
E	Fault model-driven test derivation from finite state models	Petrenko	Modeling and verification of parallel processes	2001
NC	Feature interaction detection using a synchronous approach and testing	du Bousquet *et al*.	Computer Networks	2000
E	Formal methods software engineering for the CARA system	Martin	International Journal on Software Tools for Technology Transfer	2004
NC	Formal Specification Based Software Testing: An Automated Approach	Gill and Bhatia	Proceedings of the International Conference on Software Engineering Research and Practise	2003
C	Formal test-case generation for UML statecharts	Gnesi *et al*.	IEEE ICECCS	2004
D	Formally testing fail-safety of electronic purse protocols	Jurjens and Wimmel	ASE	2001
B	From faults via test purposes to test cases: on the fault-based testing of concurrent systems			
D	From MC/DC to RC/DC: Formalization and analysis of control-flow testing criteria	Vilkomir and Bowen	Formal Aspects of Computing	2006
D	From Object-Z specifications to ClassBench test suites	Carrington et al.	Journal of Software Testing Verification and Reliability	2000
E	From U2TP models to executable tests with TTCN-3: an approach to model driven testing	Zander *et al*.	Testing of communicating systems	2005
E	Generating a test oracle from program documentation: work in progress	Peters and Parnas	ACM SIGSOFT international symposium on Software testing and analysis	1994
B	Generating functional test cases in-the-large for time-critical systems from logic-based specifications	Morasca *et al*.	ISSTA	1996
E	Generating optimal distinguishing sequences with a model checker	Mallett *et al*.	A-MOST	2005
E	Generating oracles from your favorite temporal logic specifications	Dillon and Ramakrishna	ACM SIGSOFT Symposium on the Foundations of Software Engineering	1996
D	Generating regression tests via model checking	Lihua *et al*.	COMPSAC	2004
NC	Generating test case based on UML statecharts	Miao-Huai *et al*.	Mini Micro Systems	2005
B	Generating test cases for real-time systems from logic specifications	Mandrioli *et al*.	ACM Transactions on Computer Systems	1995
C	Generating Test Cases from an OO Model with an AI Planning System	Scheetz *et al*.	ISSRE	1999
B	Generating test cases from class vectors	Leung *et al*.	Journal of Systems and Software	2003
A	Generating test cases from UML activity diagram based on Gray-box method	Linzhang *et al*.	APSEC	2004
NC	Generating test data for specification-based tests via quasirandom sequences	Che *et al*.	Lecture Notes in Computer Science	2006
B	Generating test data from SOFL specifications	Offutt and Liu	Journal of Systems and Software	1999
A	Generating Test Data from State-Based Specifications	Offutt *et al*.	Journal of Software Testing, Verification and	2003

			Reliability	
C	Generating Test Sequences from UML Sequence Diagrams and State Diagrams	Sokenou	Informatik Forschung und Entwicklung	2006
B	Generating test suites for software load testing	Avritzer and Weyuker	International symposium on Software testing and analysis	1994
A	Generating tests from UML specifications	Offutt and Abdurazik	UML	1999
E	Generating transition probabilities to support model-based software testing	Walton and Poore	Software - Practice and Experience	2000
B	Generating, selecting and prioritizing test cases from specifications with tool support	Yu *et al.*	QSIC	2003
NC	Generation of reliability test data with UML for real-time embedded software	Jun and Minyan	Journal of Beijing University of Aeronautics and Astronautics	2003
E	Generation of test sequences from formal specifications: GSM 11-11 standard case study	Bernard et al.	Software - Practice and Experience	2004
B	Identification of categories and choices in activity diagrams	Chen *et al.*	QSIC	2005
E	Implementation of model based intelligent next generation test generator using neural networks	Singer	SPIE The International Society for Optical Engineering	1996
D	Improving design dependability by exploiting an open model-based specification	Tomita and Sakamura	IEEE Transactions on Computers	1999
E	Improving functional testing using model-based diagnostics	Nolan and Carey	Proceedings of the Technical Program	
E	Improving functional/diagnostic testing using model-based reasoning	Carey and Dussault	IEEE AUTOTESTCON Proceedings	1998
C	Improving State-Based Coverage Criteria Using Data Flow Information	Briand *et al.*	Technical Report of Carleton University	2004
D	Improving test suites via operational abstraction	Harder *et al.*	ICSE	2003
B	Improving web application testing with user session data	Elbaum *et al.*	ICSE	2003
E	In this Issue	Briand and Basili	Empirical Software Engineering	2005
E	Increasing dependability by means of model-based acceptance test inside RTOS	Zhao *et al.*	Parallel Processing and Applied Mathematics	2005
B	In-parameter-order: a test generation strategy for pairwise testing	Lei and Tai	IEEE International High-Assurance Systems Engineering Symposium	1998
E	Integrating automated test generation into the WYSIWYT spreadsheet testing methodology	Fisher *et al.*	TOSEM	2006
E	Integrating formal specification and software verification and validation	Duke *et al.*	Teaching Formal Methods	2004
C	Integration of ‘components’ to test software components	Bertolino *et al.*	Electronic Notes in Theoretical Computer Science	2003
B	Integration of specification-based and CR-based approaches for GUI testing	Chen *et al.*	AINA	2005
C	Introducing a Reasonably Complete and Coherent Approach for Model-based Testing	Bertolino *et al.*	Electronic Notes in Theoretical Computer Science	2005
D	JUMBL: a tool for model-based statistical testing	Prowell	Annual Hawaii International Conference on System Sciences	2003
D	Korat: automated testing based on Java predicates	Boyapati et al.	ISSTA	2002
B	Lessons learned from automating tests for an operations support system	Fecko and Lott	Software—Practice & Experience	2002
D	Lutess: a specification-driven testing environment for synchronous software	du Bousquet *et al.*	ICSE	1999
E	Mars Polar Lander fault identification using model-based testing	Blackburn *et al.*	IEEE International Conference on Engineering of Complex Computer Systems	2002
E	Mars Polar Lander fault identification using model-based testing	Blackburn *et al.*	Annual NASA Goddard Software Engineering Workshop	2001
B	Mastering test generation from smart card software formal models	Bouquet *et al.*	Lecture Notes in Computer Science	2005
D	MaTeLo - statistical usage testing by annotated sequence diagrams, Markov chains and TTCN-3	Dulz and Zhen	International Conference on Quality Software	2003
E	Micro architecture coverage directed generation of test programs	Ur and Yadin	ACM/IEEE conference on Design automation	1999
NC	Model based development of embedded vehicle software at DaimlerChrysler	Conrad *et al.*	Informatik Forschung und Entwicklung	2005

D	Model based regression test reduction using dependence analysis	Korel *et al.*	ICSM	2002
B	Model based testing in evolutionary software development	Pretschner *et al.*	RSP	2001
NC	Model based testing in incremental system development	Pretschner *et al.*	Journal of Systems and Software	2004
B	Model Based Testing in Practice at Microsoft	Stobie	Electronic Notes in Theoretical Computer Science	2005
E	Model checking	Berg and Raffelt	Model Based Testing of Reactive Systems	
E	Model-Based Approaches for Validating Business Critical Systems	Augusto *et al.*	STEP	2003
C	Model-Based Built-In Tests	Gross *et al.*	Electronic Notes in Theoretical Computer Science	2005
B	Model-based formal specification directed testing of abstract data types	Jia	COMPSAC	1993
D	Model-based functional conformance testing of web services operating on persistent data	Sinha and Pardkar	TAV-WEB	2006
B	Model-based specification and testing applied to the Ground-Based Midcourse Defense (GMD) system: an industry report	Lakey	A-MOST	2005
NC	Model-based system testing of software product families	Reuys *et al.*	Lecture Notes in Computer Science	2005
B	Model-based test case generation for smart cards	Philipps *et al.*	Electronic Notes in Theoretical Computer Science	2003
E	Model-Based Test Driven Development of the Tefkat Model-Transformation Engine	Steel and Lawley	ISSRE'	2004
NC	Model-based test generation and implementation	Hartman and Proeyen	Informatie	2003
E	Model-based testing	Pretschner	ICSE	2005
A	Model-based testing and maintenance	Deng *et al.*	*ICMSE*	*2004*
E	Model-based Testing Considering Cost, Reliability and Software Quality	Htoon and Thein	APSITT	2005
A	Model-based testing for applications derived from software product lines	Olimpiew and Gomaa	A-MOST	2005
E	Model-based testing for enterprise software solutions	Jain	COMPSAC	2005
B	Model-based testing for real : the inhouse card case study	Pretschner *et al.*	International Journal on Software Tools for Technology Transfer (STTT)	2001
A	Model-Based Testing from UML Models	Bernard *et al.*	Informatik Forschung und Entwicklung	2006
B	Model-based testing in practice	Dalal *et al.*	ICSE	1999
B	Model-based testing of a highly programmable system	Dalal *et al.*	International Symposium on Software Reliability Engineering	1998
E	Model-based testing of concurrent programs with predicate sequencing constraints	Wu and Lin	QSIC	2005
A	Model-based testing of object-oriented systems	Rumpe	Formal Methods for Components and Objects	2003
NC	Model-based testing through a GUI	Kervinen *et al.*	FATES	2005
C	Model-based testing with UML applied to a roaming algorithm for Bluetooth devices	Zhen *et al.*	Journal of Zhejiang University Science	2004
E	Model-Based Testing: Challenges Ahead	Heimdahl	COMPSAC	2005
E	Model-based tests of truisms	Menzies *et al.*	ASE	2002
NC	Modeling and testing agent systems based on statecharts	Seo *et al.*	FORTE Workshops	2004
B	Modeling requirements for combinatorial software testing	Lott *et al.*	A-MOST	2005
E	Modellgestutzte Erprobungsmethodik in der Antriebsstrangentwicklung	Albers and Schyr	VDI Berichte	2005
E	Modelling the quality economics of defect-detection techniques	Wagner	WoSQ	2006
B	Models for synchronous software testing	Lakehal *et al.*	International Workshop on Model, Design and Validation	2004
E	Multiplexing of partially ordered events	Campbell *et al.*	Lecture Notes in Computer Science	2005
E	Mutation operators for Object-Z specification	Liu and Miao	ICECCS	2005
E	Mutation Testing Applied to Estelle Specifications	Souza *et al.*	Software Quality Control	1999
B	Mutation-based testing criteria for timeliness	Nilsson *et al.*	COMPSAC	2004
E	Mutually enhancing test generation and specification inference	Tao and Notkin	Formal Approaches to Software Testing	
E	Non-specification-based approaches to logic testing for software	Kobayashi *et al.*	Information and Software Technology	2002

E	On fault classes and error detection capability of specification-based testing	Tsuchiya and Kikuno	ACM Transactions on Software Engineering and Methodology	2002
B	On the Complexity of Generating Optimal Test Sequences	Boyd and Ural	IEEE Transactions on Software Engineering	1991
E	On the economics of requirements-based test case prioritization	Srikanth and Williams	EDSER	2005
NC	On the effectiveness of classification trees for test case construction	Chen and Poon	Information and Software Technology	1998
B	On the effectiveness of mutation analysis as a black box testing technique	Murnane and Reed	ASWEC	2001
E	On the identification of categories and choices for specification-based test case generation	Chen *et al.*	Information and Software Technology	2004
B	On the integration of design and test: a model-based approach for embedded systems	Pfaller *et al.*	AST	2006
E	On the relationships of faults for Boolean specification based testing	Lau and Yu	ASWEC	2001
E	On the State of the Art in Requirements-based Validation and Test of Software	Ryser *et al.*	Techinical Report at University of Zurich	1999
E	On the testing methods used by beginning software testers	Yu *et al.*	Information and Software Technology	2004
E	On the use of the classification-tree method by beginning software testers	Yu *et al.*	ACM symposium on Applied computing	2003
E	One evaluation of model-based testing and its automation	Pretschner *et al.*	ICSE	2005
E	On-line Fault Detection of Sensor Measurements	Koushanfar *et al.*	Proceedings of IEEE Sensors	2003
B	Online testing with model programs	Veanes *et al.*	ESEC/FSE	2005
B	Optimal strategies for testing nondeterministic systems	Lev Nachmanson	ISSTA	2004
D	Parameterized unit tests	Tillmann and Schulte	ESEC/FSE-13	2004
B	Plannable test selection criteria for FSMs extracted from operational specifications	Paradkar	ISSRE	2004
D	Play to test	Blass *et al.*	Formal Approaches to Software Testing	2005
D	Practical approach to specification and conformance testing of distributed network applications	Kuliamin *et al.*	Lecture Notes in Computer Science	2005
NC	Preamble computation in automated test case generation using constraint logic programming	Colin *et al.*	Software Testing, Verification & Reliability	2004
E	Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis	Do *et al.*	Empirical Software Engineering	2006
B	Probe: a formal specification-based testing system	Amayreh and Zin	International conference on Information Systems	1999
E	Product family testing: a survey	Tevanlinna *et al.*	ACM SIGSOFT Software Engineering Notes	2004
B	Projected state machine coverage for software testing	Friedman *et al.*	ISSTA	2002
E	Property-based testing: a new approach to testing for assurance	Fink and Bishop	ACM SIGSOFT Software Engineering Notes	1997
B	ProTest: An Automatic Test Environment for B Specifications	Satpathy *et al.*	Electronic Notes in Theoretical Computer Science	2005
NC	proTEST: an automatic, scenario-driven, requirements-based software test process	Hirt	CONQUEST	2000
E	Random testing of interrupt-driven software	Regehr	EMSOFT	2005
NC	Refinement in statechart testing	Bogdanov and Holcombe	Software Testing Verification & Reliability	2004
D	Regression testing of classes based on TCOZ specification	Liang	ICECCS	2005
B	Requirement-based automated black-box test generation	Tahat *et al.*	COMPSAC	2001
B	Requirements traceability in automated test generation: application to smart card software validation	Bouquet *et al.*	A-MOST	2005
B	Requirements-Based Monitors for Real-Time Systems	Peters and Parnas	IEEE Transactions on Software Engineering	2002
E	Requirements-driven software test: a process-oriented approach	Ramachandran	ACM SIGSOFT Software Engineering Notes	1996
NC	Reusing class-based test cases for testing object-oriented framework interface classes	Dallal and Sorenson	Journal of Software Maintenance and Evolution: Research and Practice	2005
E	Reverse engineering of test cases for selective regression testing	Sneed	European Conference on Software Maintenance and Reengineering	2004
C	Revisiting Strategies for Ordering Class Integration Testing in the Presence of	Briand *et al.*	Technical Report of Carleton University	2002

	Dependency Cycles – An Investigation of Graph-based Class Integration Test Order Strategies			
D	SALT - An Integrated Environment to Automate Generation of Function Tests for APIs	Paradkar	ISSRE	2000
NC	Scenario-based system test with software-product families	Reuys *et al*.	Informatik Forschung und Entwicklung	2005
E	Selecting small yet effective set of test data	Paradkar	IASTED International Multi-Conference on Applied Informatics	2003
E	Separating sequence overlap for automated test sequence generation	Hierons	ASE	2006
NC	Sequencing constraints-based regression testing of concurrent programs after specification changes	Kim *et al*.	Journal of KISS: Software and Applications	2000
NC	Siddhartha - automated test driver-oracle synthesis	Richardson	ACM SIGSOFT Software Engineering Notes	2000
E	Siddhartha: a method for developing domain-specific test driver generators	Reyes and Richardson	ASE	1999
D	Software Architecture Analysis Based on Statechart Semantics	Dias and Vieira	International Workshop on Software Specification and Design	2000
E	Software assurance by bounded exhaustive testing	Sullivan *et al*.	ISSTA	2004
E	Software Fault Tolerance: A Tutorial	Wilfredo	NASA Langley Technical Report Server	2000
E	Software in Metrology	RICHTER	PTB-MITTEILUNGEN	1991
E	Software model checking in practice: an industrial case study	Chandra *et al*.	ICSE	2002
NC	Software requirements and acceptance testing	Hsia *et al*.	Annals of Software Engineering	1997
E	Software requirements validation via task analysis	Zhu *et al*.	Journal of Systems and Software	2002
E	Software test selection patterns and elusive bugs	Howden	COMPSAC	2005
D	Software testing at the architectural level	Richardson and Wolf	ISAW-2 and Viewpoints	1996
E	Software unit test coverage and adequacy	Zhu *et al*.	ACM Computing Surveys	1997
B	Specification based test sequence generation with propositional logic	Wimmel *et al*.	Software Testing Verification and Reliability	2000
NC	Specification testing of synchronous software	Parissis	Technique et Science Informatiques	2002
D	Specification-based class testing with ClassBench	Murray *et al*.	Asia Pacific Software Engineering Conference	1998
A	Specification-based regression test selection with risk analysis	Chen *et al*.	*CASCON*	*2002*
B	Specification-Based Test Generation for Security-Critical Systems Using Mutations	Wimmel and Jürjens	International Conference on Formal Methods and Software Engineering	2002
D	Specification-based test oracles for reactive systems	Richardson *et al*.	ICSE	1992
NC	Specification-based testing for GUI-based applications	Chen and Subramaniam	Software Quality Control	2002
B	Specification-based testing for real-time avionic systems	Biberstein and Fitzgerald	IEE Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems	1999
B	Specification-based testing for real-time reactive systems	Alagar et al.	TOOLS	2000
E	Specification-based testing of concurrent programs	Carver	ACM SIGSOFT Software Engineering Notes	2000
NC	Specification-based testing of concurrent systems	Ulrich and Konig	Formal Description Techniques and Protocol Specification, Testing and Verification	1998
B	Specification-based testing of reactive software: a case study in technology transfer	Jagadeesan *et al*.	Journal of Systems and Software	1998
B	Specification-based testing of reactive software: tools and experiments-experience report	Jagadeesan *et al*.	ICSE	1997
B	Specification-based testing of synchronous software	Parissis and Ouabdesselam	Symposium on the Foundations of Software Engineering	1996
NC	Specification-based testing of user interfaces	Paiva *et al*.	Interactive Systems, Design, Specification, and Verification	2003
B	Specification-based testing using cause-effect graphs	Paradkar *et al*.	Annals of Software Engineering	1997
B	Specification-based testing with linear temporal logic	Tan *et al*.	IRI	2004

D	Specifying and Testing Software Components using ADL	Hayes and Sankar	Techinical Report at Sun Microsystems	1994
D	State-based incremental testing of aspect-oriented programs	Xu and Xu	AOSD	2006
D	State-based testing of integration aspects	Xu and Xu	WTAOP	2006
B	Statechart testing method for aircraft control systems	Bogdanov and Holcombe	Software Testing Verification & Reliability	2001
E	Static driver verifier, a formal verification tool for Windows device drivers	Levin	MEMOCODE	2004
E	Status report: requirements engineering	Hsia *et al.*	IEEE Software	1993
B	Strategies for automated specification-based testing of synchronous software	Parissis and Vassy	ASE	2001
E	Structural coverage analysis method	Gifford	AIAA/IEEE Digital Avionics Systems Conference	1996
D	Structural specification-based testing with ADL	Chang *et al.*	ISSTA	1996
D	Structural specification-based testing: automated support and experimental evaluation	Chang and Richardson	ESEC	1999
NC	Study on software test method based on finite state machine	Lan *et al.*	Journal of North China Electric Power University	2005
E	Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact	Do *et al.*	Empirical Software Engineering	2005
NC	Systematic Model-Based Testing of Embedded Automotive Software	Conrad *et al.*	Electronic Notes in Theoretical Computer Science	2005
B	Systematic model-based testing of embedded control software: the MB^3/T approach	Conrad *et al.*	ICSE	2004
NC	Systematic testing as base for securing quality	Schlatter	Information Management and Consulting	2006
B	Telecommunication software validation using a synchronous approach	du Bousquet *et al.*	ASSET	1998
B	Test Case Generation as an AI Planning Problem	Howe *et al.*	ASE	1997
C	Test cases generation from UML state diagrams	Kim *et al.*	IEE Software	1999
D	Test input generation for java containers using state matching	Visser *et al.*	ISSTA	2006
D	Test input generation with java PathFinder	Visser *et al.*	ISSTA	2004
E	Test prioritization for pairwise interaction coverage	Bryce and Colbourn	A-MOST	2005
A	Test ready UML statechart models	Murthy *et al.*	SCESM	2006
D	Test selection for object-oriented software based on formal specifications	Peraire *et al.*	International Conference on Programming Concepts and Methods	1998
A	Test selection from UML Statecharts	Liuying and Zhichang	TOOLS	1999
NC	Test specification based on trace description	Petrenko	Programming and Computer Software	1993
D	Test template framework: a specification-based testing case study	Stocks and Carrington	ISSTA	1993
D	Test templates: a specification-based testing framework	Stocks and Carrington	ICSE	1993
B	Test-based model generation for legacy systems	Hungar *et al.*	ITC	2003
D	TestEra: Specification-based testing of java programs using SAT	Khurshid and Marinov	ASE	2004
E	Testing against some eventuality properties of synchronous software: A case study	du Bousquet *et al.*	Electronic Notes in Theoretical Computer Science	2004
A	Testing agile requirements models	Botaschanjan *et al.*	Journal of Zhejiang University Science	2004
B	Testing of concurrent programs based on message sequence charts	Chung *et al.*	International Symposium on Software Engineering for Parallel and Distributed Systems	1999
E	Testing polymorphic interactions in UML sequence diagrams	Supavita and Suwannasart	ITCC	2005
B	Testing real-time embedded software using UPPAAL-TRON: an industrial case study	Larsen *et al.*	EMSOFT	2005
E	Testing software modelling tools using data mutation	Shan	AST	2006
B	Testing times: On Model-Driven Test Generation for Non-Deterministic Real-Time Systems	Brinksma, E.	ACSD	2004
B	Testing web applications by modeling with FSMs	Andrews *et al.*	Systems and Modeling	2005

E	Testing with model checker: insuring fault visibility	Okun *et al*.	WSEAS Transactions on Systems	2003
B	Test-suite reduction for model based tests: effects on test quality and implications for testing	Heimdahl and George	ASE	2004
C	TestUml: user-metrics driven web applications testing	Bellettini *et al*.	SAC	2005
E	The (Im)maturity level of software testing	Bertolino	ACM SIGSOFT Software Engineering Notes	2004
E	The advanced mobile application testing environment	Binder and Hanlon	A-MOST	2005
A	The AGEDIS tools for model based testing	Hartman and Nagin	ISSTA	2004
E	The ASTOOT approach to testing object-oriented programs	Doong and Frankl	TOSEM	1994
E	The effect of code coverage on fault detection under different testing profiles	Cai and Lyu	A-MOST	2005
E	The semantics of triveni: A process-algebraic API for threads + events	Colby *et al*.	Electronic Notes in Theoretical Computer Science	1998
E	The specification-based testing of a trusted kernel: MK++	Ford *et al*.	International Conference Conference on Formal Engineering Methods	1997
E	The state problem for test generation in Simulink	Zhan and Clark	GECCO	2006
E	The UniTesK Approach to Designing Test Suites	Kuliamin *et al*.	Programming and Computing Software ,	2003
E	The use of model-based test requirements throughout the product life cycle	Bukata *et al*.	Aerospace and Electronic Systems Magazine	2000
B	Thoroughness of specification-based testing of synchronous programs	Parissis and Vassy	ISSRE	2003
D	TinMan - A test derivation and management tool for specification-based class testing	Murray *et al*.	TOOLS	1999
E	Too darned big to test	Stobie	Queue	2005
E	Tools for model-based security engineering	Jürjens and Fox	ICSE	2006
B	Towards a more efficient way of generating test cases: class graphs	Leung and Wong	Asia-Pacific Conference on Quality Software	2000
C	Towards Automated Support for Deriving Test Data from UML Statecharts	Briand *et al*.	Technical Report of Carleton University	2004
B	Towards integration of use case modelling and usage-based testing	Regnell *et al*.	Journal of Systems and Software	2000
B	Towards model-based generation of self-priming and self-checking conformance tests for interactive systems	Paradkar	Information and Software Technology	2004
E	Towards model-driven testing of a Web application generator	Baresi et al.	Lecture Notes in Computer Science	2005
E	Traceability techniques: Using scenarios to support traceability	Naslavsky *et al*.	TEFSE	2005
C	Traffic-aware stress testing of distributed systems based on UML models	Garousi *et al*.	ICSE	2006
E	Translating test programs using a model-based process	Bearse and Lynch	AUTOTESTCON	1999
B	T-UPPAAL: online model-based testing of real-time systems	Mikucionis *et al*.	ASE	2004
B	T-VEC: a tool for developing critical systems	Blackburn and Busser	COMPASS	1996
B	T-VECTM product summary		Workshop on Industrial Strength Formal Specification Techniques	1998
NC	UML based statistical testing acceleration of distributed safety-critical software	Yan *et al*.	ISPA	2004
C	UML-based integration testing	Hartmann *et al*.	ISSTA	2000
C	UML-Based Integration Testing for Component-Based Software	Wu *et al*.	ICCBSS	2003
A	UML-based statistical test case generation	Riebisch *et al*.	NetObjectDays	2002
E	Uniform descriptions for model based testing	Krishnan	ASWEC	2004
E	Usage model-based automated testing of C++ templates	Sayre	A-MOST	2005
B	Use Case-based Testing of Product Lines	Bertolino and Gnesi	ESEC and FSE-11	2003
E	Use of sequencing constraints for specification-based testing of concurrent programs	Carver and Tai	IEEE Transactions on Software Engineering	1998
B	Use-case driven test for object-oriented system	Choi	Software Engineering and Applications	2001
E	Using a model checker to test safety properties	Ammann *et al.*	IEEE International Conference on Engineering of Complex Computer Systems	2001
B	Using artificial life techniques to generate test cases for combinatorial testing	Shiba *et al*.	International Computer Software and	2004

			Applications Conference	
B	Using formal methods to derive test frames in category-partition testing	Ammann and Offutt	COMPASS	1994
B	Using formal specifications as test oracles for system-critical software	Hagar and Bieman	ACM SIGAda Ada Letters	1996
E	Using information about functions in selecting test cases	Clermont and Parnas	A-MOST	2005
B	Using model checking to generate tests from requirements specifications	Gargantini and Heitmeyer	ESEC/FSE-7	1999
E	Using model-based test program generator for simulation validation	Zhang *et al.*	Lecture Notes in Computer Science	2005
E	Using Simulation to Empirically Investigate Test Coverage Criteria Based on Statechart	Briand *et al.*	ICSE	2004
NC	Using state diagrams to generate unit tests for object-oriented systems	Ipate and Holcombe	Lecture Notes in Computer Science	2005
E	Using Test Oracles Generated from Program Documentation	Peters and Parnas	IEEE Transactions on Software Engineering	1998
E	Using the incremental approach to generate test sets: a case study	Yu *et al.*	QSIC	2003
C	Using UML Collaboration Diagrams for Static Checking and Test Generation	Abdurazik and Offutt	International Conference of UML	2000
C	Using UML for Automatic Test Generation	Crichton *et al.*	ASE	2001
B	Using Z specifications in category partition testing	Amla and Ammann	COMPASS	1992
E	Validation in model-driven engineering: testing model transformations	Fleurey *et al.*	International Workshop on Model, Design and Validation	2004
E	Validation test of distributed program based on event sequencing constraints	Qing *et al.*	Journal of Software	2000
A	Verification of requirements for safety-critical software	Carpenter	SIGAda	*1999*
B	White on black: a white-box-oriented approach for selecting black box-generated test cases	Chen *et al.*	Asia-Pacific Conference on Quality Software	2000

Appendix B – Model-based Testing Approaches Classification

Approach	Publication Year	Category	Testing Level	Software Domain	Technology /Language	Behavior Models	Tools to support	Automated Steps	Complexity of not-automated steps	Intermediate Model
Abdurazik 2000	2000	C	System Testing	OO	UML	Collaboration Diagram	Not-defined	"2/3"	1. Necessary (Initial Behavior Model Modeling)	No
Ali2005	2005	C	Integration Testing	OO	UML and SCOTEM	State Diagram, Collaboration Diagram, and SCOTEM	SCOTEM Constructor, Test Path Generator, Test Executor, and Results Evaluator	"3/4"	0. Necessary (Initial Behavior Model Modeling)	SCOTEM
Ammann1994	1994	B	System Testing	Not defined	Z Notation, TSL	Z Specification	Not Defined	"1/5"	All steps can be defined together by the complexity: Necessary (Initial Behavior Model Modeling)	No
Andrews2005	2005	B	System Testing	Web Application	Finite State Machine	Finite State Machine	One tool is defined	“3/8”	1.1 High (partitioning into clusters) 1.3 Medium (Intermediate Model Modeling or Test Data Definition) 1.4 Medium (Intermediate Model Modeling or Test Data Definition)	No
Barbey1996	1996	B	Unit Testing	OO	CO-OPN/2 – Concurrent Object-Oriented Petri Nets	CO-OPN/2	LOFT	“3/4”	1. Necessary (Initial Behavior Model Modeling)	No
Basanieri 2000	2000	C	Integration Testing	OO	UML	Use Case Diagram, Sequence Diagram, and Class Diagram	Not-defined	“3/7”	1. Necessary (Initial Behavior Model Modeling) 2. High (Analysis of the Behavior Model) 4. Medium (Test Model Modeling) 5. High (Translation or other task)	No
Bellettini2005	2005	C	System Testing	Web Application	UML	Extended ClassDiagram, Statechart Diagrams	TestUML	“4/6”	0. Necessary (Initial Behavior Model Modeling) 1. Low (Only choices) 4. Low (Only choices)	No
Bernard2006	2006	A	System Testing	Not defined	UML 2.0	Class Diagram, Object Diagram, Statechart Diagram, OCL	LEIRIOS Test Generator	“4/5”	1. Necessary (Initial Behavior Model Modeling)	No
Bertolino2003	2003	C	Integration Testing	Component -based Software	UML Components	Use Case Diagram, Sequence Diagram, Classe Diagram	Cow_Suite and CDT	“2/3”	1. Necessary (Initial Behavior Model Modeling)	No

Bertolino2005	2005	C	Integration Testing	OO	UML	Sequence Diagram, State Diagram, Reasonably Complete Model	Cow_Suite	“2/4”	0. Necessary (Initial Behavior Model Modeling) 1. High (Translation or other task) 3. High (Translation or other task)	SEQ’’ Model, and SQRrc Model
Beyer2003	2003	C	Integration Testing	Not defined	UML	Annotated Sequence Diagram and MCUM	MaTeLo	“2/3”	1. Necessary (Initial Behavior Model Modeling)	MCUM (Markov Chain Usage Model)
Botaschanjan 2004	2004	A	System Testing	OO	UML	Object Diagram, Sequence Diagram and OCL	Not-defined	“0/3”	1. Necessary (Initial Behavior Model Modeling) 2,3. Medium (Intermediate Model Modeling)	No
Bousquet 1999a	1999	D	System Testing	Synchronous Reactive Software	LUSTRE	Environment description	Luteness	“3/5”	1. Necessary (Initial Behavior Model Modeling) 4. Low (Only choices)	No
Briand2002	2002	A	System Testing	OO	UML	Use Case Diagram + Activity Diagram, Sequence Diagram, Collaboration Diagram, Class Diagram, Data Dictionary (OCL)	TOTEM	“8/9”	1. Necessary (Initial Behavior Model Modeling)	No
Briand2002a	2002	A	Regression Testing	OO	UML	Class Diagram, Use case Diagram and Sequence Diagrams	RTSTool	“2/3”	3. Low (Only choices)	No
Briand2002b	2002	C	Integration Testing	OO	UML and Graph	Class Diagram and Graph	Not-defined	“2/2”	0. Necessary (Initial Behavior Model Modeling)	Graph
Briand2004a	2004	C	System Testing	OO	UML and TTS	Statechart Diagram + OCL, Class Diagram, Transition Test Sequence	CBCDTool	“3/3”	0. Necessary (Initial Behavior Model Modeling)	No
Briand2004b	2004	C	System Testing	OO	UML and EAFG	State Diagram + OCL, event/action flow graph	Not-defined	“3/4”	2. High (Translation or other task)	Event/action flow graph
Briand2006	2006	C	Unit Testing	COTS	UML	Class Diagram, Sequence Diagram + OCL, CSPE constraints, graph	PrestoSequence	“3/7”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Intermediate Model Modeling or Test Data Definition) 3. Low (Only choices) 4. Low (Only choices)	
Carpenter 1999	1999	A	System Testing	Safety-critical System	UML	Use case Diagram, Sequence Diagram	Not-defined	“2/5”	1. Necessary (Initial Behavior Model Modeling) 3. Medium (Intermediate Model Modeling or Test Data Definition) 5. High (Translation or other task) [Converting test cases in test scripts, definition of value classes for each field]	No

Cavarra2003	2003	A	System Testing	OO	UML and Abstract State Machine	Class Diagram, State Diagram, Object Diagrams, Intermediate Format (ASM)	AGEDIS	"2/3"	1. Necessary (Initial Behavior Model Modeling)	Intermediate Format
Chang1999	1999	D	Unit Testing	Not defined	Assertion Definition Language, C	ADL Specification and Test Data Description (TDD)	ADLscope	"3/5"	1. Necessary (Initial Behavior Model Modeling) 2. Medium (test data and Boolean condition modeling)	No
Chen2002	2002	A	Regression Testing	OO	UML	Activity Diagram	Not-defined	"2/3 (1st option); 0/2 (2nd option)"	Both: 0. Necessary (Initial Behavior Model Modeling) Targeted Testing: 3. Low (Only choices) Safety Testing: 1,2. Low (Only choices)	No
Chen2005	2005	D	Integration Testing	Not defined	SOFL Language	Condition Data Flow Diagram – CDFD (SOFL)	Not Defined	"3/4"	0. Necessary (Initial Behavior Model Modeling) 4. Medium (Test execution and analysis)	No
Chevalley 2001	2001	A	System Testing	Critical Software	UML	State Diagram	Rose RealTime tool	"3/5"	1. Necessary (Initial Behavior Model Modeling – UML State Diagram) 2. Necessary (Initial Behavior Model Modeling – Input Variables)	No
Crichton2001	2001	C	System Testing	OO	UML and Intermediate Format	System Model (Class, Object and State Diagram) and Test Directives (Object and State Diagrams), Intermediate Format	Not-defined	"2/3"	1. Necessary (Initial Behavior Model Modeling)	Intermediate Format
Dalal1999	1999	D	Unit Testing	Not defined	AETGSpec	Requirements described in AETGSpec	AETG	"2/3"	1. Necessary (Initial Behavior Model Modeling)	No
Deng2004	2004	A	System Testing, Regression Testing	OO	UML	Use Case, Class, Stetachart, Sequence, Colaboration, Activity Diagrams	ISSEE	"1/1(1st option); 0/4(2nd option)"	Both: 0. Necessary (Initial Behavior Model Modeling) Regression Testing: 1,2,3,4. Medium (Modeling and Selection of the Test Cases for Regression Testing)	No
Friedman 2002	2002	B	System Testing	Concorrent Software	Murφ Description Language	Finite State Machine	GOTCHA	"4/6"	1. Necessary (Initial Behavior Model Modeling) 3. Low (Only choices)	No
Gargantini 1999	1999	B	System Testing	Not defined	SCR Specifications	SIS Specification	Tool in Java that constructs a suite of test sequences and an other to generate extra test sequences for data boundaries.	"1/3"	1. Necessary (Initial Behavior Model Modeling)	SMV or Spin
Garousi2006	2006	C	System	Distributed	UML and	Class, Sequence,	Not-defined	"2/4"	1. Necessary (Initial Behavior	No

			Testing (Stress Testing)	System	Test Model	Context, Network Deployment (Package Diagram), and Modified Interaction Overview (Interaction Overview Diagram) Diagrams (UML), Test Model (control flow model, network interconnectivity tree, network traffic usage patterns, and inter-SD constraints)			Model Modeling) 2. Medium (Test Model Modeling)	
Gnesi2004	2004	C	System Testing	OO	UML and IOLTS	Statechart Diagram and IOLTS	Not-defined	“2/3”	1. Necessary (Initial Behavior Model Modeling)	IOLTS
Gross2005	2005	C	Integration Testing	OO	UML2.0 and TTCN-3	U2TP and TTCN-3	Not-defined	“2/3”	1. Necessary (Initial Behavior Model Modeling)	TTCN-3
Hartman2004	2004	A	System Testing	Distributed System	UML, IF and XML	* Behavior model: combination of class state and object diagrams * Test Generation Directives: State Diagram * Test Execution Directives: XML file	AGEDIS	“3/6”	1,2,3. Necessary (Initial Behavior Model Modeling)	Intermediate Format
Hartmann 2000	2000	C	Integration Testing, Unit Testing	OO	UML and TSL	Statechart Diagram and Graph	TnT (TDE and TECs)	“3/4”	1. Necessary (Initial Behavior Model Modeling)	Graph
Hong2000	2000	B	System Testing	Not defined	STATEMATE, Extended Finite State Machine	Statechart, Extended Finite State Machine	STATEMATE toolset	“5/6”	1. Necessary (Initial Behavior Model Modeling)	EFSM, flow graph
Kansomkeat 2003	2003	A	System Testing	OO	UML and EFSM	Statechart Diagram, Testing Flow Graph	Not-defined	“2/3”	1. Necessary (Initial Behavior Model Modeling)	Testing Flow Graph
Kim1999	1999	C	Unit Testing	OO	UML and EFSM	State Diagram, EFSM	Not-defined	“3/4”	1. Necessary (Initial Behavior Model Modeling)	Extended Finite State Machine
Legeard2004	2004	D	System Testing	Critical Software	B or Z Language	Formal Specification (B or Z)	BZ-TT	“4/6”	1. Necessary (Initial Behavior Model Modeling) 4. Low (Only choices about the testing coverage criteria to reduce the test cases set)	BZP Format
Linzhang2004	2004	A	System Testing	OO	UML	Activity Diagram	UMLTGF	“2/3”	1. Necessary (Initial Behavior Model Modeling)	No
Liuying1999	1999	A	System Testing	OO	UML, Finite State Machine	State Diagram, FSM	Rose98	“3/7”	1. Necessary (Initial Behavior Model Modeling) 3.1. Low (Only choices) 3.2 Low (Only choices) 3.3 Low (Only choices)	Finite State Machine

Lucio2005	2005	A	System Testing	OO	Fondue	* Concept Model: Class Diagram (Fondue) * Behavior Model: Environment (UML collaboration diagrams), Protocol (UML state diagrams) and Operation (OCL operations) (Fondue).	Not Defined	“3/5”	1. Necessary (Initial Behavior Model Modeling) 2,4. Medium (Intermediate Model Modeling or Test Data Definition)	No
Lund2006	2006	A	System Testing	Not defined	UML 2.0	Sequence Diagram	Not-defined	“0/0”	Not Applied	No
Mandrioli1995	1999	B	System Testing	Real-time System and Critical Software	TRIO Language	Software Specification in TRIO	Prototype Tool is described	“3/6”	1. Necessary (Initial Behavior Model Modeling) 2. Necessary (Initial Behavior Model Modeling) 3. Low (Only choices)	No
Meyer1998	1998	A	System Testing (GUI Testing)	Not defined	UML	Use Case + Operational Profile – Test Engineering Model	TestMaster and WinRunner	“2/5”	1. Necessary (Initial Behavior Model Modeling) 3. High (Translation or other task [extension of models with □hysics information about the software]) 5. High (Translation or other task)	No
Mingsong 2006	2006	A	System Testing	Java Programs	UML	Activity Diagram	AGTCG	“1/4”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Code instrumentation) 3. Low (Only choices)	No
Murthy2006	2006	A	System Testing	Not defined	UML and Context Free Grammar Model	Extended Statechart Diagram, EXTENDED CTGM	Test generation Tool	“3/3”	0. Necessary (Initial Behavior Model Modeling)	No
Nebut2006	2006	A	System Testing	Object-oriented Embedded Software	UML and OCL	Use Case Diagram, Sequence Diagram, Simulation Model	UC-System	“3/5”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Intermediate Model Modeling)	Simulation Model
Offut1999a	1999	B	System Testing, Unit Testing	Critical Software	SOFL, DNF, TSL	DNF, CDFD (SOFL), S-Module (SOFL), I-Module (SOFL)	Not Defined	“2/4”	1. Necessary (Initial Behavior Model Modeling) 2. Necessary (Initial Behavior Model Modeling)	DNF
Offutt1999b	1999	A	System Testing	OO	UML	Statechart Diagram	UMLTest	“3/4”	1. Necessary (Initial Behavior Model Modeling)	No
Offutt2003	2003	A	System Testing	OO	UML and Graph	Statechart Diagram, Specification Graph	SpecTest	“6/8”	0. Necessary (Initial Behavior Model Modeling) 6. High (Translation or other task) 8. Medium (Test Data Definition)	Specification Graph
Olimpiew2005	2005	A	System Testing	Software Product Line	PLUS – UML2.0	Feature model, use case model, static [class] and dynamic [statechart and object	Not-defined	“0/0”	Not Applied	No

						interaction] models, component-based software architecture models				
Paradkar 2004a	2004	B	System Testing	Reactive System	SALT, Extended Finite State Machine	Operational Model (SALT), Extended Finite State Machine (EFSM)	Planner	“2/3”	1. Necessary (Initial Behavior Model Modeling)	No
Parissis1996	19996	B	System Testing	Synchronous Software (Reactive Software)	LUSTRE	Environment Specification and the safety properties	LESAR	“4/5”	1. Necessary (Initial Behavior Model Modeling)	No
Pretschner 2001	2001	B	System Testing	Reactive Systems and Embedded System	AutoFocus, Constraint Logic Programming	System Structure Diagrams, State Transition Diagram, Message Sequence Charts, Data Type Diagram	AutoFocus	"4/5"	1. Necessary (Initial Behavior Model Modeling)	CLP Code
Richardson 1992	1992	D	System Testing	Reactive systems	Not defined, may be changed.	In the case study, RTIL and Z Specification	Not defined	“2/4”	1. Necessary (Initial Behavior Model Modeling) 3. Medium (Intermediate Model Modeling or Test Data Definition)	No
Richardson 1996	1996	D	Integration Testing	Not defined	Chemical Abstract Machine	CHAM	Not Defined	“2/4”	1. Necessary (Initial Behavior Model Modeling) 2. Low (Only choices)	No
Riebisch2002	2002	A	System Testing	OO	UML and Graph	Use Case, Statechart Diagram, Graph Model and Usage Model	UsageTester Tool	“5/6”	1. Necessary (Initial Behavior Model Modeling)	Graph Model and Usage Model
Rumpe2003	2003	A	System Testing	OO	UML	Object Diagram, Sequence Diagram, OCL	Not-defined	“0/3”	1,2,3. High (Translation or other task)	No
Satpathy2005	2005	B	System Testing	Not defined	B Specification, XML, Prolog Representation	State Coverage Graph (B)	ProTest	“7/8”	1. Necessary (Initial Behavior Model Modeling)	XML File and Prolog Representation
Scheetz1999	1999	C	Integration Testing	OO	UML	Class Diagram and State Diagram + OCL	UCPOP 4.0 (Planner)	“2/4”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Intermediate Model Modeling)	Test Plan
Sinha2006a	2006	D	Integration Testing	Web service	WSDL-S, EFSM	WSDL-S, EFSM	Not Defined	“2/3”	1. Necessary (Initial Behavior Model Modeling)	Extended Finite State Machine
Sokenou2006	2006	C	Integration Testing, Unit Testing	OO	UML	Sequence Diagram, Protocol Statechart, OCL	Not-defined	“3/4”	1. Necessary (Initial Behavior Model Modeling)	No
Stobie2005a	2005	B	System Testing	Any Category	ASML, XML	Finite State Machine (FSM) and Abstract State Machine Language (ASML)	TMT and AsmL Test Tool	“3/4”	1. Necessary (Initial Behavior Model Modeling)	No
Stocks1996	1996	D	Unit Testing	Not defined	Z Notation, TSL	Test Template, describing valid inputs and outputs	The framework	“0/0”	Not Applied	No

Tahat2001	2001	B	System Testing and Regression Testing	Distributed System and Embedded System	SDL, EFSM	Requirements expressed in Textual and SDL formats, EFSM	Not Defined	“3/5 and 2/2”	For System Testing: 1. Necessary (Initial Behavior Model Modeling) 4. Low (Only choices)	EFSM
Tan2004	2004	B	System Testing	Not defined	linear temporal logic – LTL	Software Specificaton in LTL	Model Checker: SMV and SPIN	“3/4”	1. Necessary (Initial Behavior Model Modeling)	No
Traore2003	2003	A	System Testing	OO	UML	Statechart, Class and Sequence Diagrams	PrUDE	“4/6”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Extended Model Modeling)	No
Vieira2006	2006	A	System Testing (GUI Testing)	Not defined	UML and TSL	Use Case + Activity Diagram and Class Diagram	TDE/UML and Eclipse (SCR) UML Editor	“2/4”	1. Necessary (Initial Behavior Model Modeling) 2. Medium (Intermediate Model Modeling)	Class Diagram and Notes on Activity Diagram
Wu2003	2003	C	Integration Testing	Component -based Software	UML	UML Collaboration/ Sequence Diagram or UML Statechart Diagram to describe Context-dependence relationships UML Collaboration Diagram or UML Statechart Diagram to describe Content-dependence relationships	Not Defined	“0/0”	Necessary (Initial Behavior Model Modeling)	No
Xu2006a	2006	D	Integration Testing	Aspect-oriented programs	AOP	State Model (AOP)	Not Defined	“4/5”	1. Necessary (Initial Behavior Model Modeling)	No
Zhen2004	2004	C	Integration Testing	OO	UML 2.0	Interaction, Activity and State Diagrams	Not-defined	“0/4”	1,2,3,4. High (Translation or other task)	No

Approach	Non-functional requirements	Inputs	Input Interpretation	Outputs	Output Interpretation	External Software
Abdurazik 2000	No	UML Collaboration Diagram in a format not defined	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Test data	Not defined	No
Ali2005	No	UML Collaboration Diagram in XMI file and test data in text file	Yes. XMI is the starndard for UML Model, and the text file could be easily interpreted	Pass/Fail Results (Test Results)	Easy. The Pass/Fail results are logged in a file. For failed test cases, the Results Evaluator also logs the test path number, message, and object states.	UML Modeling Tool that generates XMI file (Together is quoted on the text)
Ammann1994	No	Software Specification	Z format	Test scripts	TSL Format	No
Andrews2005	No	Test Model: LWP in FSM format, AFSM, all constraints on input selection	Yes	Test Sequences	Evalid Scripts	No
Barbey1996	No	Software Specification	CO-OPN/2 format	Test results	Not defined	No

Basanieri 2000	No	UML diagrams	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Use Case Test Suite (Test cases set)	Not defined	No
Bellettini2005	No	Web applications UML model written in XMI, a Web server log file, and XML configuration files specifying Web server address, information about metrics and coverage use	Yes	Test case specification described using XML, with the format specified by a DTD file.	Easy, because there is a DTD file to support the output interpretation	WebUML
Bernard2006	No	UML Models using the Together Modeling Tool	Yes	Test Results after the tests execution	Yes	Together
Bertolino2003	No	UML Diagrams using UML Components notation	Easy, because this format uses MOF and it is interpreted by the Cow_Suite tool	Test Procedures in JUnit or WCT format	Yes	Not defined
Bertolino2005	No	UML Sequence and State Diagram	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	SEQrc that represents a coherent integration of the initial models	Not defined	No
Beyer2003	No	UML Sequence Diagram in XMI file	Yes	Test Cases in TTCN-3 format	Yes	UML Modeling Tool that generates XMI file
Botaschanjan2004	No	UML Models and OCL in a format not defined by the paper	Not defined	Expected result and/or OCL contract as test oracle	Not defined	Not defined
Bousquet1999a	Security Properties	Environment description and an oracle	LUSTRE Format	Test sequences	LUSTRE Format	No
Briand2002	No	Model behavior using TOTEM	Yes, because it is done in the same tool to generate test cases.	Test cases and Test Oracles in OCL expressions	Easy. This notation (OCL) is included on UML and it's schema is defined by OMG.	No
Briand2002a	No	UML diagrams of two system versions (XMI files produced by UML case tools) along with the original regression test suite	Yes. XMI is the standard to UML definition.	Information about the changes between the models and re-classification of test cases after the changes	Yes, it's a report.	UML Modeling Tool that generates XMI file
Briand2002b	No	UML Class Diagram	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Classes Order for Integration Testing	Not defined, but it must be easy, because described just one ordered list of classes.	Not defined
Briand2004a	No	Inputs: information on (1) statecharts (including state invariants), and operations contracts, (2) the class diagram (operation signatures, attributes, associations), (3) the TTS, (4) equivalent navigation paths (in the form of a user helper file) in the class diagram.	Easy. XMI File and OCL. And there are rules to translate Statechart in DNF easily. There is a metamodel pre-defined.	The constraints for the different transitions in the TTS.	Not defined	UML Modeling Tool that generates XMI file
Briand2004b	No	UML Statechart Diagram in a format not specified	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Paths (Data flow) using EAFGs	Not easy	Not defined

Briand2006	No	Component metadata and Test Specification provided by the component vendor and component user	Not easy. The inputs involve UML, OCL and CSPE constraints	Test Sequences based on the DRPP solution	Easy. Paths (sequences) similar to Traveling Salesman Problem	No
Carpenter1999	No	The approach is abstract, so, the input are UML Models and Test Data Arguments and Types	Easy to define	Test Cases and Test Scripts	Easy to define	Test-harness generation tool to convert test cases into test scripts
Cavarra2003	No	UML specification in XMI file	Yes. It's a standard to UML definition	Test cases is provides in Tree and Tabular Combined Notation (TTCN) - a Standard format in the telecommunication industry	Easy. This output can be easily translated to produce test cases in the language of any API, whether this is C, C++, or Java.	UML Modeling tool that generates XMI file, like rose, objecteering, together, or poseidon
Chang1999	No	ADL Specification and Test Data Description (TDD)	ADL format	Test drivers	C Format	ADL
Chen2002	No	Activity Diagrams for each feature and the test cases	Not defined	Subset of test cases selected from the full set for one software	Yes, just one list of test cases	No
Chen2005	No	SOFL Specification	SOFL Language	Test cases	Not defined	No
Chevalley2001	No	A set of state diagrams S representing super-states of the (sub)system under study.	Yes, using UML State Diagram	Two outputs. 1. outputs produced by simulation of the model with the input values. 2. file represents a Java class that contains the input values translated into Java instructions	Yes. Java class	Rose RealTime tool
Crichton2001	No	UML Specification in XMI file	Yes. It's a standard to UML definition	Test suites in Tree and Tabular Combined Notation (TTCN)	Yes. This output can be translated to produce test cases in the language of any API, whether this is C, C++, or Java.	UML Modeling Tool that generates XMI file
Dalal1999	No	Test data modeling	Not defined	Test Cases	Not defined	No
Deng2004	No	* System Testing: UML Models using ISSEE * Regression Testing: set of test cases provided by SSDM	Not defined	* System Testing: New Test cases * Regression Testing: Test cases related to the changed operation	Not defined	No
Friedman2002	No	behavioral model of the system under test, written in the Murφ Description Language for EFSMs, set of coverage criteria and test constraints written in the syntax given above.	Format described on the paper	Test cases	Not defined	No

Gargantini1999	No	SCR specification	Yes	Test Sequences	XML files	SMV or Spin model checker
Garousi2006	Efficiency and Reliability	UML Diagrams in a not-specified format	Not defined	Stress Test Requirements	Not defined	Not defined
Gnesi2004	No	Statechart Diagram	Yes	test cases written in a formal language	Not easy	AutoFocus and TGV/AGEDIS
Gross2005	No	System UML Models developed manually	Hard. There is not tool to support the input importation	Test cases in TTCN-3 format	Yesy. This notation is used usually to describe test cases	No
Hartman2004	No	a) the behavioral model of the system, b) the test execution directives which describe the testing architecture of the SUT, and c) the test generation directives which describe the strategies to be employed in testing the SUT.	A and B are easy because UML to describe the models. C is easy because use a defined XML schema.	Abstract test suite consisting of test cases which cover the desired test generation directives.	Yes. The test generator is based on the TGV engine, but with additional coverage and observation capabilities derived from the GOTCHA test generator. This format is already defined.	UML modeling tool equipped with the AGEDIS UML profile (e.g. Objecteering UML Modeler)
Hartmann2000	No	The modeling individual or collections of components using UML Statecharts, and establishing a global behavioral model of the composed Statecharts.	Easy using Rational Rose	A Set of conformance tests described in TSL. These test cases ensure the compliance of the design specification with the resulting implementation.	Easy. TSL can be interpreted for a lot of test execution tools	Rational Rose
Hong2000	No	Statechart	STATEMATE format	Test Sequences	Not defined	No
Kansomkeat2003	No	Sequence Diagrama using Rational Rose tool	Yes, but it's necessary license to use this tool	Test cases in the conjunctive form	Yes, because it utilizes a know-format	Rational Rose
Kim1999	No	UML State Diagram	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Test cases	Not defined	Not defined
Legeard2004	No	B or Z Model from Atelier B	Medium	Test Scripts	Not defined	Atelier B
Linzhang2004	No	UML Specification in MDL plain text file (generated by Rational Rose tool)	Yes, because it is a standard notation	Test cases from activity diagrams using UMLTGF tool	Yes, because the tool support the generation and view of the test cases.	Rational Rose
Liuying1999	No	UML State Diagram	UML Specification generated by Rose98	Test sequence	Not defined	No
Lucio2005	No	* Fondue Concept and Behavior Model using XMI file. * Observality hypotheses using COOPN specifications.	* Fondue Models: This XMI can be easily loaded. * Observality hypotheses: Not easy. These hypotheses are based on temporal logic formulas that allow us to express test intentions for COOPN specifications using a language developed by the research group responsible by this approach.	Test cases expressed by the temporal logic formulas which are the main basis of the test/constraint language that is under development by this research group.	Not easy	Astra QuickTest - Execution

Lund2006	No	Diagrams that may contain the operators neg and assert and the tests generated are themselves represented as sequence diagrams.	Not easy. It uses Sequance diagram and STAIRS semantics to describe the software behavior, and this language is not interpretaed easily.	Test cases with themoperators neg and assert	Not easy. The test case are generated using the same STAIRS semantics.	No
Mandrioli1995	Time Requirements	Software Specifications are in text files	Yes. Text files	Test results	Not defined	History checker
Meyer1998	No	Use Case + Operational Profile	Not easy. There are a lot of complex steps to develop the OP	Test script output stream on the format WinRunner Scripts.	Yes, scripts in the TSL format to be executed by WinRunner	No
Mingsong2006	No	Activity Diagram and Java Code Source	Easy	Tests execution results in the Java Program	Easy	No
Murthy2006	No	Extended Statechart Diagram	Easy (need license)	Test Scripts for each statechart diagram	Easy (java code)	Rational Rose
Nebut2006	No	Use Case Diagram and the contract of each use case	Not easy. For each use case must be defined contracts using logic operators to be interpreted during the test cases generation.	Test cases	Easy. Test cases in the form of Java classes, using the JUnit framework as a test harness.	JUnit
Offut1999a	No	Models in SOFL	SOFL Language	Test cases	TSL Format	Not defined
Offutt1999b	Efficiency	Rose specification file (MDL file)	Easy (need license)	Test cases in an ASCII text file	Easy	Rational Rose
Offutt2003	No	Transition conditions directly from SCR tables and from UML specifications	Not easy. The step of translating test specification values into program inputs must be executed manually.	Test scripts	The format is not defined and SPECTEST doesn't support this step.	SCRTool and Rational Rose
Olimpiew2005	No	PLUS Models	Yes. The paper described how to interprete the model used by the approach	Test Case for an Use Case generated randomicaly	Yes. Test cases are described on a table with their specific fields (inputs, outputs,requirements, itens, needs)	Not defined
Paradkar2004a	No	Operational Model (SALT)	Yes	Test cases	Not defined	No
Parissis1996	Security Properties	Environment specification and safety properties	LUSTRE Format	Test data to check the system (including the safety properties)	LUSTRE Format	No
Pretschner2001	Time Requirements	AutoFocus Model	generated by AutoFocus	Tests Veredicts	generated by AutoFocus	Not defined
Richardson1992	Efficiency and Security	System Specification	Not defined	Test Oracles	Not defined	Not defined
Richardson1996	No	Chemical Abstract Machine	Not defined	Test data	Not defined	No
Riebisch2002	No	Use Cases according with a specific template	Described on a table (on the paper), it's easy to be interpreted	Failure data and usage model; metrics for test sufficiency and product quality	The format is not described, but these information are interpreted easily	No
Rumpe2003	Efficiency	Object , Sequence Diagram, and OCL from a not defined format or tool	These diagrams usually use the MOF, and it's easy to interpret this format	Test cases modeled in UML	Easy	The approach suggers the use of tools to accomplish the model translations, but it's not defined one tool

Satpathy2005	No	B Specification	B language	Test verdicts for Java Programs	Not defined	No
Scheetz1999	No	UML Models	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Test cases in not specified format	Not defined	UCPOP 4.0 - Planner
Sinha2006a	No	WSDL-S model	Yes. This is a extension of the international standard to describe web service	ESFM	Yes	No
Sokenou2006	No	UML Sequence and State Diagram	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Test cases in a format not specified	Not defined	Not defined
Stobie2005a	No	Specification in ASML	Yes	Test cases in XML	Yes	No
Stocks1996	No	Input Space	Yes. Z notation	Output Space	Yes. Z notation	Not defined
Tahat2001	No	Software Requirements	expressed in Textual and SDL formats	Test cases	Not defined	Not defined
Tan2004	Security and Efficiency (Time)	Software Specification	LTL Format	Test cases	LTS Format	Checkers: SMV and SPIN
Traore2003	No	UML Specification in XMI file	Yes	Test cases definitions and tests results in PRUDE Toolkit format	Not defined. There is a tool to support the model creation	UML Modeling tool that generates XMI file
Vieira2006	No	Activity Diagram	Easy - Eclise Diagram	Test Scripts in TSL format	Easy - used for several replay capture tools	Eclipse - adapted to the approach
Wu2003	No	UML Diagrams describing context dependence relationships and content dependence relationships	Not defined, but UML use MOF to define its models. This standard may be interpreted easily	Not defined	Not defined	Not defined
Xu2006a	No	State modeling (class and aspects)	Not defined	Test results	Not defined	No
Zhen2004	Efficiency	Design Model System in UML 2.0 format	Easy. It's not defined a specific format, but UML 2.0 uses MOF to defined the models and could be easily interpreted	Test cases	Not defined	No