Identifying approaches to generate test cases in model-based testing: a systematic mapping

Matheus Monteiro Mariano[1], Érica Ferreira de Souza[2], André Takeshi Endo[2], and Nandamudi L. Vijaykumar[1]

Lab. of Comp. and Applied Math., National Institute for Space Research – INPE[1]
São José dos Campos/SP - Brazil
matheus.mariano@inpe.br,vijay.nl@inpe.br
Department of Computer, Federal University of Technology Paraná – UTFPR[2]
Cornélio Procópio/PR - Brazil
ericasouza@utfpr.edu.br,andreendo@utfpr.edu.br

Abstract. Context: Model-Based Testing (MBT) has attracted a lot of attention from software testing researchers, since it has been efficient in employing formal models to represent the reactive systems behavior in order to guide test case generation. **Objective:** This paper investigates approaches to automatically generate test cases in MBT, such as methods, criteria, formal models, and evaluation strategies of the generated test cases. **Method:** A systematic mapping was conducted in order to summarize the main approaches used in the MBT context. **Results:** From the mapping, we identified 97 studies addressing approaches to generate test cases in MBT. **Conclusion:** Several pointers to some approaches have been identified which will help practitioners and researchers to identify future research from existing research efforts in MBT. Among the indicators that most caught the attention is that there is a great effort spent in developing new test methods (32 methods), test criteria (29 criteria), formal models (24 models) and evaluations conducted in the selected studies (145 types).

1 Introduction

In order to assure software quality, several efforts in academia and industry have been dedicated to more refined test strategies [1]. Moreover, several products and services depend on developing systems considered critical and complex. A software system is considered critical when a fault (even simple ones) can lead to serious damage or loss both financially and socially. For example, if a software embedded in a satellite fails, the ground stations may not receive proper information to be used in several applications, causing financial losses.

In such scenarios, activities related with Verification, Validation and Testing (VV&T) have been employed throughout the software development process. These activities analyze the compliance of the developed software with the specification [2]. Among such activities, software testing can be defined as an execution process of a program with intention of finding errors.

During the testing activity, a model can be created to guide the test process in order to verify whether the behavior of implemented software is in conformance with its specification. Therefore, it is necessary to observe whether the software implementation is behaving in accordance with the specified model. According to El-Far and Whittaker [3], software testing requires the use of a model to guide efforts on generation and selection of test cases. In this context, *Model-Based Testing* (MBT) has attracted a lot of attention since there are several advantages in using models to represent the system's behavior and to guide the generation of test cases [4]. MBT can bring some benefits, such as high rate of failure detection, reduction of cost and time, traceability, and ease of updating software requirements [4].

One of the main features of MBT is the automated generation of test cases from a formal test model. In this respect, several formal techniques have been adopted to specify the system behavior as a test model, such as *Statecharts* [5], Finite State Machines (FSMs) [6], *Specification and Description Language* (SDL) [7] and *Labelled Transition Systems* (LTSs) [8]. Each modeling technique has specific methods and criteria to traverse a model and automatically generate test cases. However, depending on the nature of system, the number of test sequences generated can be potentially infinite or even very large, exceeding the capacity of exercising the test cases on the implementation.

The main objective of this paper is to investigate elements involved in automatic test case generation in MBT, focusing on studies that addressed, mainly, algorithms for test case generation, such as methods, criteria, formal models for generating test cases and evaluation strategies of the generated test cases and shows the main approaches used for each of these elements in the MBT context. To achieve this, we conducted a systematic mapping to summarize the main approaches used in such a context. A systematic mapping provides a broad overview of an area of research, determining whether there is research evidence on a particular topic [9]. We believe that the results of this mapping will help to identify areas for future research and also provide a map that appropriately allows placing new research activities.

The remainder of this paper is structured as follows. Section 2 presents the main underlying concepts of this work. Section 3 formally introduces the methods and procedures used to conduct the systematic mapping. Section 4 shows how the systematic mapping was conducted. Results are presented in Section 5. Section 6 reports a general discussion to highlight some points of our research. Lastly, conclusions, remarks and future directions are described in Section 7.

2 Background

Software Testing can be defined as as an execution process of a program with intention of finding errors [2]. Several activities that help in finding errors before the systems are finalized can avoid this problem, such as VV&T activities. VV&T dedicate(s) efforts to make a software specification cohesive with implementation and an approach to make this is to check if the system will behave as expected from a finite set of test cases [10].

Test activity can be applied in different levels of software. The three main levels are [2]: *(i)* unit testing, that focuses in identifying faults in small and independent components of software; *(ii)* integration testing, responsible in testing the integration of small units of software; and *(iii)* system testing, that verifies if the behavior of software is in conformance with requirements, analyzing the internal flow of a program. Each level analyzes a specific behavior of the program and for each level a specific technique can be applied. Two main testing techniques are: Structural testing which has the objective of analyzing the software based on the program code; and Functional testing which generates tests from a formal model without the code, analyzing if the behavior of software is in accordance with specification. Functional testing might employ an abstraction of the software, based on its specification, that represents the behavior of system. This approach is well-known as Model-Based Testing (MBT).

MBT is an approach that represents the behavior of system as a formal model [11], using an abstraction. Software modeling is important to conduct test activities because it enables to identify the behavior and its expected result. In MBT, model can be designed using a formal model technique without ambiguity [4]. Some of the most well-known techniques to represent test models for a system under test are Finite State Machine (FSM) [6], Statecharts [5], Specification and Description Language (SDL) [7], and Labelled Transition Systems (LTS) [8].

With a model, it is possible to employ test criteria and methods to generate test cases. The selection of criteria and methods is important because depending on the model size or the input range set it becomes impracticable to analyze all the possible combinations. A test criterion defines which elements of a model must be covered to generate a test sequence (test case). The main test criteria mentioned in literature are [12]: all nodes, all edges, all transition-pairs and fault model coverage. A method is an approach used to find a set of test cases implemented by an algorithm. In general, a method aims to achieve a test criterion by covering a model. Some known methods are: Distinguished Sequence (DS) [13], W-method [14], Transition Tour (TT) [15], Unique Input/Output (UIO) [16], Breadth First Search (BFS) [17] and Depth First Search (DFS) [17]. These methods differ with respect to cost of generated test suites and adopted criteria complexity.

2.1 Related Work

Before accomplishing the mapping, we adopted concepts of a tertiary study to look for secondary studies that already investigated MBT. Tertiary studies are reviews that focus only on secondary studies; i.e., it is a review about other secondary studies [9]. In this study, we used the following search string:

("MBT" OR "model based test" OR "model based testing" OR "model based tests" OR "MDT" OR "model driven test" OR "model driven testing" OR "model driven tests") AND ("Systematic Literature Review" OR "Systematic Review" OR "Systematic Mapping" OR "Mapping Study" OR "Mapping Studies" OR "Systematic Literature Mapping").

The search string was applied in the Scopus electronic database: 86 documents were returned and we found ten complete studies that are reviews on MBT. Some related studies are briefly described as follows.

Barmi et al. [18] conducted a systematic mapping with respect to the alignment of specification and testing of functional and non-functional requirements to identify gaps in the literature. The map summarizes studies and discusses them within sub categories with MBT and traceability. The main results showed that the papers that have linked the specification with test requirements focused on model-centric approaches, code-centric approaches, traceability, formal approaches and test cases, with focus on problems of MBT and traceability.

Siavashi and Truscan [19] conducted a systematic review in order to understand how an environment model can be used in MBT and challenges. Environment modeling is an activity that specifies a part of the real world, in which the system is integrated. From the characteristics and advantages, the authors conclude that using environment models can be helpful in robustness testing, safety testing and regression testing. However, methodological aspects of creating environment models are only discussed in a limited number of studies.

Gurbuz and Tekinerdogan [20] presented a systematic mapping to describe the advances in MBT for software safety. The results show that the safety area of MBT is broad and applicable to several domains with focus on reducing costs and increasing test coverage. Nevertheless, the solutions proposed are focused on specific domains, making the solution less generic and less reusable. More recently, Khan et al. [21] showed a systematic review in order to analyze the quality of empirical studies with MBT. The results identified that the community of MBT needs to provide more details in reporting guidelines, and few papers were applied in industry artifacts.

Each presented study conducted some kind of secondary study on MBT, but with different purposes. Other related studies have focused on mobile applications [22], search based techniques [23], aspect-oriented software [24], MBT tools [25, 26] and MBT approaches [27]. In the same way as related works, we also conducted a secondary study on MBT, however, our focus was to systematically map the methods, criteria, formal models for generating test cases and the evaluation strategies of the generated test cases.

3 Research Protocol

The research method was defined based on the guidelines given in Kitchenham and Charters [9]. The method involves three main phases: *(i)* **Planning:** refers to the pre-review activities, and establishes a protocol, defining the research questions, inclusion and exclusion criteria, sources of studies, search string, and mapping procedures; *(ii)* **Conducting:** focuses on searching and selecting the studies in order to extract and synthesize data from included ones; and *(iii)* **Reporting:** is the final phase and it writes up the results and circulates them to potentially interested parties. Following, the main parts of the mapping protocol used in this work are presented.

Research Questions. This mapping study aims at answering the following Research Questions (RQs):

- RQ1. When and where have the studies been published?
- RQ2. What are the methods used in the generation of test cases?
- RQ3. What are the test criteria used in the generation of test cases?
- RQ4. What formal modeling technique is presented for generating test cases?
- RQ5. What evaluation strategy is carried out in the selected study?

Inclusion and Exclusion Criteria. The selection criteria are organized in one Inclusion Criterion (IC) and seven Exclusion Criteria (EC). The inclusion criterion is: (IC1) The study must present an empirical approach for test case generation in MBT. The exclusion criteria are: (EC1) The study is published as a short paper or extended abstract; (EC2) The study is not written in English; (EC3) The study is an older version (less updated) of another study already considered; (EC4) The study is not a primary study, such as editorials, summaries of keynotes, workshops, and tutorials; (EC5) The study does not present an empirical approach to generate test cases; (EC6) The full paper is not available; and (EC7) Studies published before the year 2000. The EC7 was defined based on Offurt and Abdurazik's [28] study published in 1999. This study is referred to as one of the first MBT approaches. Thus, in this mapping we consider only studies published after that date.

Keywords and Search String. We defined the search string in this way to focus only in papers that use methods for test case generation. So, the search string considered two areas, MBT and test case generation (see Table 1), and it was applied in three metadata fields (namely, title, abstract and keywords). The search string was modified to be adapted to particularities of each source.

Table 1: Keywords for the search string

Areas	**Keywords**
MBT	"MBT", "model based test", "model based testing", "model based tests", "MDT", "model driven test", "model driven testing", "model driven tests"
Test Case	"test criteria", "test criterion", "testing criteria", "testing criterion", "generation method", "generation algorithm", "test cases generation"
Search String: ("MBT" OR "model based test" OR "model based testing" OR "model based tests" OR "MDT" OR "model driven test" OR "model driven testing" OR "model driven tests") AND ("test criteria" OR "test criterion" OR "testing criteria" OR "testing criterion" OR "generation method" OR "generation algorithm" OR "test cases generation")	

Source. We chose to work with Scopus[1] database, since this source is considered the largest abstract and citation database of peer-reviewed literature, with more than 60 million records. In addition, Scopus attaches papers of other international publishers, including Cambridge University Press, Institute of Electrical and Electronics Engineers (IEEE), Nature Publishing Group, Springer, Wiley-Blackwell, and Elsevier.

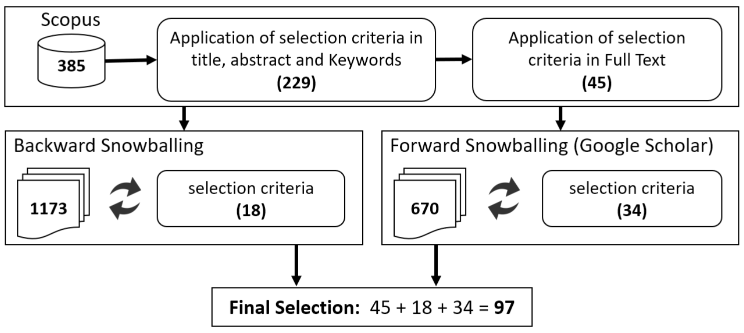
[1] *https://www.scopus.com/home.uri*

The studies returned from sources in the searching phase were cataloged and stored appropriately. This catalog helped us in the classification and analysis procedures. With the studies remained after applying IC and EC, the snowballing process was conducted. First, referring to papers cited in these remaining studies (backward snowballing) and second, referring to those studies that cited these remaining studies (forward snowballing).

Assessment. Before conducting the mapping study, we tested the mapping protocol. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to our investigation. First, the review process was conducted by the first author of this paper, and, only then, the other three authors carried out the review validation. The authors analyzed the studies using three different samples (approximately 30% each).

4 Conducting the mapping study

The main steps performed in this mapping study are shown in Figure 1.



Scopus
385
Application of selection criteria in title, abstract and Keywords
(229)
Application of selection criteria in Full Text
(45)
Backward Snowballing
1173
selection criteria
(18)
Forward Snowballing (Google Scholar)
670
selection criteria
(34)
Final Selection: 45 + 18 + 34 = 97

Fig. 1: Selection process

We considered the studies published until March 2018. As a result, 385 publications were returned by Scopus. The selection process was divided into two stages. In the 1^{st} stage, inclusion and exclusion criteria were applied considering the title, abstract and keywords, so 156 publications (approximately 41%) were eliminated. In the 2^{nd} stage, the exclusion criteria were applied considering the entire text, resulting in 45 studies (approximately 12% of total articles). Over these 45 studies considered relevant, we performed backward and forward snowballing. The backward snowballing resulted in 1173 studies. From these 1173 studies, the selection criteria were applied considering the title, abstract and keywords. Next, the selection criteria were applied considering the full text. This stage was carried in an iterative form. Three iterations were conducted and 18 new studies were selected. The forward snowballing returned 670 studies. Four iterations were conducted, resulting in 34 new studies. As a result, we got a final set of 97 studies[2].

[2] The references of the 97 selected studies can be accessed at: *https://goo.gl/i9QTiz.*

5 Data Extraction and Synthesis

After selecting the primary studies, we analyzed each one in order to answer the RQs. The studies classification was made from the RQs answers. The same answers were grouped and consequently a category was defined. Next, we present the main findings from data extraction and synthesis. The complete mapping of individual studies and its categories can be accessed at: *https://goo.gl/i9QTiz*.

In data extraction and synthesis, notice that for some RQs the sum of all classifications might be greater than the total number of papers in the systematic mapping. This occurs because a given paper can answer a RQ with more than one target (e.g. models, methods) of our investigation. In addition, in some RQs, elements that were mentioned only once were grouped into "Others" category.

RQ1. When and where have the studies been published?
Figure 2 shows the distribution of the studies by years. The results suggest that research on MBT has been growing, with a significant increase from 2014 to 2017. In fact, the period between 2011-March 2018 represents 72.2% of all publications of this mapping. That indicates the increased interest in MBT recently. In addition, in relation to the publication vehicle, Conferences are the sources with most publications representing 54.6% and Journals 41.2%. For 4.1% of the papers we do not identified the source of publication.

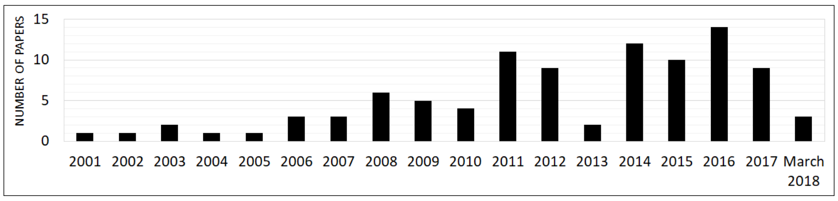


Fig. 2: Distribution of studies by year

RQ2. What are the methods used to generate test cases?
This RQ led us to identify the methods that have been used in MBT. Figure 3 shows the main methods identified. A total of 32 types were identified. Most of the papers propose a new method (24.1%), always to improve an existing method in terms of efficiency and coverage. For example, Hessel and Pettersson [29] proposed a new global algorithm that uses less memory and time to generate a test suite. In Dorofeeva and Koufareva [30], a modification in the classic W-method was proposed. Despite based on a classic method, it is considered as a new method, according to authors. The same occurs in Souza et al. [1] by proposing a new method called H-Switch Cover based on a classic algorithm n-switch set cover.

Optimal-path algorithms have also been used in the generation of test cases (13.5%), as shown in Figure 3. These algorithms are very useful to generate test cases with maximum coverage of a system model, while optimizing the number of test cases. Different types of algorithms were selected in this group. For example, Belli and Hollmann [31] employed the k-transition coverage to generate a test

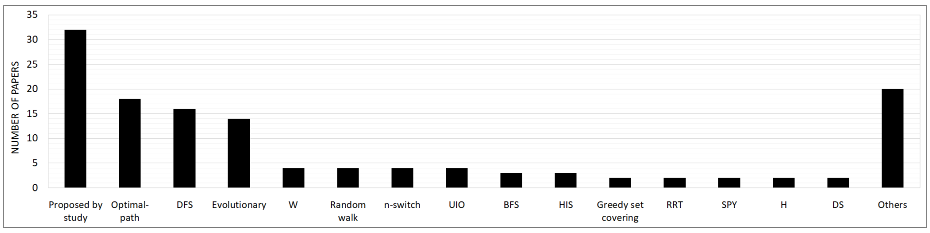


Fig. 3: Methods identified in the selected studies

case for each transition sequence of length k. Then, it is minimized using the Floyd-Warshall algorithm to generate the shortest path. In Julliand et al. [32], a method was proposed to generate test cases from a predicate abstraction of a system specified as an event-based system. This method uses the Chinese Postman Problem (CPP) to cover the transitions of the given model.

We also identified that the well-known classic algorithms continue to be used in MBT. Depth-First Search (DFS) algorithm, for instance, was the most mentioned (12%). This algorithm was used to generate a complete and fast coverage of the model, since it is well-known and very simple to implement. Many studies used the DFS concurrently with another criterion or method, such as Takagi et al. [33] that uses DFS to satisfy the n-switch coverage criterion or applied in a tool to automate all the MBT process like [34].

RQ3. What are the criteria used in the generation of test cases? RQ3 focuses on verifying the criteria used in the identified studies. Figure 4 shows the main criteria found in the selected studies. We identified 29 types of criteria, and the three most mentioned criteria are related to graph coverage: *All-edges* or equivalent (e.g., all-transitions in an FSM) with 36.5%, followed by *all-states* (13.9%), and *all-paths* (12.4%).

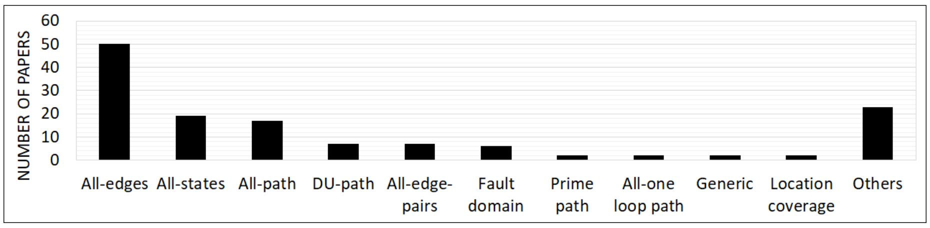


Fig. 4: Criteria identified in the selected studies

Many methods or algorithms identified in RQ2 use a single criterion to generate test cases. For instance, Belli et al. [35] propose a method that use an optimal-path finding algorithm to generate test cases covering *all transitions*. Devroey et al. [36] proposed an approach to generate a suite of abstract test cases that satisfies *all-states* criterion. Other studies adopt a method to cover two or more criteria. For instance, Hessel and Pettersson [29] apply both *all-transitions* and *all-states* to generate test cases using the global algorithm. Belli et al. [37] introduce an event-oriented approach to MBT of Web Service Compositions. In order to do this, their algorithm uses three different criteria to generate test cases: 2-length sequences (*all edges*), 3-length sequences (*all edge-pairs*), and 4-length sequences.

RQ4. What formal modeling technique is presented for generating test cases?
A large number of formal models for generating test cases were observed. However, several Unified Modeling Language (UML) models used in the studies drew our attention. So, we analyzed this RQ in two steps. First, we analyzed only UML models, in order to identify which UML diagram was the most used. Then, all other formal modeling techniques were analyzed.

Figure 5 shows the distribution of UML models. State diagram are the most used. Some studies convert a UML diagram to another formal model to generate test cases. For example, in Anbunathan and Basu [38], a state diagram is converted into Extended FSM (EFSM), and then to Control Flow Graph (CFG), to generate test cases applying some criteria like transition coverage, path coverage, or dataflow coverage. In other studies, the authors use state diagrams to generate test cases directly. For instance, Li and Offutt [39] analyze the ability of test oracles in revealing failures. To do so, test suites are generated from UML state diagrams applying ten elicited test oracle strategies.

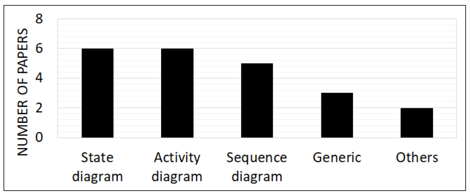


Fig. 5: UML models

Figure 6 presents the main identified formal models. In general, similarly to UML diagrams, state-based models are widely employed in the literature, with FSM being the most used. In Majeed and Ryu [40], for instance, a novel testing approach is proposed to combine OS-level replay mechanism with MBT and generate test cases applying the n-switch set cover in an FSM. Other studies proposed an extended version of FSM for a specific problem, as in Yao et al. [41]. In addition, despite state diagram representation in general can be considered as a graph representation, other formal models based on graphs (like event-driven models) are also popular in the literature.

RQ5. What evaluation strategy is carried out in the selected study?
Finally, RQ5 verifies what are the most used evaluation strategies in the literature. This evaluation strategy is related to the test cases generated and models used. We separated the evaluations into two categories (test case and model) since we observed that these were the most used in the selected studies.

Evaluations related to test cases were more commonly used in the selected studies. Some evaluations are: number of test cases (18.2%), fault detection

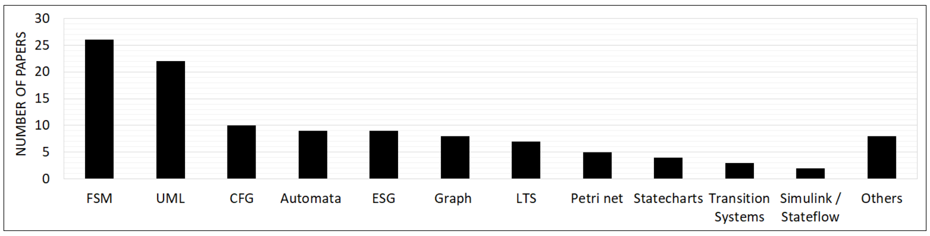


Fig. 6: Formal models used

(11.5%) and mutants analysis (7.4%). In Belli et al. [42], the authors propose a framework that uses pushdown automata (PDA) related to context-free languages to generate a small test set. The framework evaluation was based on the analysis of the number of test cases generated, as well as their average length, number of events executed, number of faults and the performance of fault detection. Ali et al. [43] introduce a technique to combine UML diagrams and Statecharts as an intermediate model (SCOTEM) to generate test paths, also defining coverage criteria to be used in the model. The validation of this technique analyzes fault detection capability, generating mutant programs.

With respect to model evaluations, they were based mainly on number of edges (7.4%), number of states (6.8%) and fault coverage (4.1%) in a model. For example, in Satpathy et al. [44], a method to cover parallel and hierarchical stateflow model was proposed, without flattening the components of the model, and generate test cases. The approach is evaluated considering the coverage of states and transitions. Cartaxo et al. [45] proposed to use a similarity function to generate less redundant test suite. The approach is compared with a random selection and uses the transition coverage to evaluate it.

Finally, we identified evaluations for both test cases and models, such as time (24.3%) and memory used (2.7%). The time is the most mentioned evaluation in the selected studies. Some studies use the time as a performance metric of computation time. For example, Dang and Nahhal [46] proposed a framework to use conformance testing in continuous and hybrid systems and a method based on Rapidly-exploring Random Tree, called RRT, to generate test cases. The time is used to analyze the efficiency of method in linear systems, comparing both the methods. In Wang et al. [47] time is used in an empirical evaluation of a tool, Tansuo, that uses a proposed approach to generate navigations graphs for dynamic web applications using combinatorial testing. The paper also analyzes how much time is needed to generate a navigation graph and to generate test cases using the t-way coverage criterion.

6 Discussion

In this work, directions to several approaches associated to MBT have been identified. Some relevant points are discussed as follows.

MBT is a research topic that shows itself in a constant growth, as can be seen in RQ1. This can also be confirmed by the number of secondary studies that have been conducted. We have identified ten secondary studies with different purposes and each with a considerable number of primary studies returned, which we believe are in the strong interest in this research area.

Another result that makes us infer the constant growth of interest in this area of research is the number of contributions on new methods proposed for the generation of test cases. As can be observed in RQ2, there is a great effort spent in developing new test methods; out of the 97 selected studies, 32 proposed a new method for generating test cases from formal models. We also identified 29 test criteria in RQ3. The formal models used were also quite varied with more than 24. And, with respect to RQ5 we identified 145 types of evaluations conducted in the selected studies considering the test cases generated and models used.

One of the main features of MBT is the automated generation of test cases usually based on a formal representation of the software specification. We noted by the systematic mapping that the industry has an inclination in employing UML models (22 UML models), due to the approximation with user and focus on systems that need the user control. On the other hand, in the case of academy, models like FSM (26) and Statecharts (4) are more used. In particular, FSMs have been heavily adopted to generate test cases for different kinds of applications, such as Web applications and embedded systems, specially reactive systems. FSM is commonly used for testing due to its rigor and simplicity. Great effort has been spent on the development of methods and criteria for FSM that select or generate effective test sets capable of revealing all faults from a given domain, as can be seen in RQ4. In general, these models (FSM and Statecharts) apply graph algorithms to generate test cases, either to obtain the best path or high coverage.

With respect to the evaluation strategy carried out in the selected studies, we have identified that the researchers usually divide the evaluations into two main categories: test cases and formal models. Evaluations related to test cases are more commonly used (39 different type of evaluations). Different analyses in the test cases are conducted to understand the behavior of the test methods. A tester can identify the trade-offs between test case characteristics, such as test suite length, and fault detection ratio, as shown in RQ5, and consequently choosing the best method to fit for a given project.

6.1 Threats to validity

The study selection and data extraction stages were initially performed by the first author. In order to reduce some subjectivity, the other authors performed these same stages over a sample of studies. The samples were compared in order to detect possible bias.

Our review was limited by the search terms used in the Scopus database. Although Scopus is considered the largest abstract and citation database, we tried to overcome possible limitations by using backward and forward snowballing. We also considered papers from a control group to calibrate the string. The categorization of papers was based on research questions (method, criterion, formal model and evaluation strategies). Even so, we are possibly leaving some valuable studies out of our analysis, since we considered papers indexed just by the Scopus database, and those obtained from snowballing. However, the studies discussed in this mapping provide an overview of empirical research on outcomes of existing research on MBT.

7 Conclusions

In this paper, we have reported the results of a systematic mapping on approaches to generate test cases in MBT. It is important to emphasize that our focus was on studies that addressed, mainly, algorithms for test case generation. We identified 97 studies addressing methods, algorithms, formal models and evaluation strategies on MBT. The major contribution of this study was to summarize and highlight the main aspects associated to MBT, but focused on the methods, formal models and evaluation strategies most used in the existing literature. These results could be of interest to several researchers involved with MBT. We believe that our summarization can help to direct researchers in their future research providing a pointer to appropriately position new activities in this research topic.

Future directions include conducting research on the behavior of the main methods and criteria returned in this mapping, considering FSMs with different configurations to represent different applications. In addition, as evaluation methods, the main evaluation strategies identified in this systematic mapping will be considered.

Acknowledgment

M.M. Mariano is financially supported by CAPES/Brazil. A.T. Endo and E.F. Souza are partially financially supported by CNPq/Brazil (grant number 420363/2018-1 and 432247/2018-1).

References

1. Souza, É.F., de Santiago Júnior, V.A., Vijaykumar, N.L.: H-switch cover: a new test criterion to generate test case from finite state machines. Software Quality Journal **25**(2) (2017) 373–405
2. Myers, G.J., Sandler, C., et al.: The art of software testing (2004)
3. El-Far, I.K., Whittaker, J.A.: Model-based software testing. Encyclopedia of Software Engineering (2001) 825–837
4. Utting, M., Legeard, B.: Practical model-based testing: a tools approach. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (2010)
5. Harel, D.: Statecharts: A visual formalism for complex systems. Science of computer programming **8**(3) (1987) 231–274
6. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE **84**(8) (1996) 1090–1123
7. Ellsberger, J., Hogrefe, D., Sarma, A.: SDL: formal object-oriented language for communicating systems. Prentice Hall (1997)
8. Tretmans, J.: Model based testing with labelled transition systems. In: Formal methods and testing. (2008) 1–38

9. Kitchenham, B.A., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University, UK (2007)
10. IEEE: THE INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS (IEEE): standard for software verification and validation. Standard 1012 (2004)
11. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J., Lott, C.M., Patton, G.C., Horowitz, B.M.: Model-based testing in practice. In: International conference on Software engineering. (1999) 285–294
12. Ammann, P., Offutt, J.: Introduction to software testing. Cambridge University Press (2008)
13. Gonenc, G.: A method for the design of fault detection experiments. IEEE transactions on Computers **100**(6) (1970) 551–558
14. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE transactions on software engineering (3) (1978) 178–187
15. Naito, S.: Fault detection for sequential machines by transition-tours. Proc. FTCS (Fault Tolerant Comput. Syst.), 1981 (1981)
16. Sidhu, D.P., Leung, T.K.: Formal methods for protocol testing: A detailed study. IEEE transactions on software engineering **15**(4) (1989) 413–426
17. Bondy, J., Murty, U.: Graduate Texts in Mathematics: Graph Theory. Springer, USA (2008)
18. Barmi, Z.A., Ebrahimi, A.H., Feldt, R.: Alignment of requirements specification and testing: A systematic mapping study. In: Software Testing, Verification and Validation Workshops (ICSTW). (2011) 476–485
19. Siavashi, F., Truscan, D.: Environment modeling in model-based testing: Concepts, prospects and research challenges: A systematic literature review. In: International Conference on Evaluation and Assessment in Software Engineering. (2015) 1–6
20. Gurbuz, H.G., Tekinerdogan, B.: Model-based testing for software safety: a systematic mapping study. Software Quality Journal (2017) 1–46
21. Khan, M.U., Iftikhar, S., Iqbal, M.Z., Sherin, S.: Empirical studies omit reporting necessary details: A systematic literature review of reporting quality in model based testing. Computer Standards & Interfaces (2017)
22. Ascate, M.S., Oliveira, K., Mendes, I., Fontão, A., Villanes, I., Dias-Neto, A.: Model-based testing for mobile applications: A systematic mapping study. In: Ibero-America Conference on Software Engineering (CIbSE). (2017)
23. Saeed, A., Ab Hamid, S.H., Mustafa, M.B.: The experimental applications of search-based techniques for model-based testing: Taxonomy and systematic literature review. Applied Soft Computing **49** (2016) 1094–1117
24. Hooda, S., Dalai, S., Solanki, K.: A systematic review of model-based testing in aspect-oriented software systems. In: Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on. (2016) 2944–2949
25. Shafique, M., Labiche, Y.: A systematic review of state-based test tools. International Journal on Software Tools for Technology Transfer **17**(1) (2015) 59–76
26. Bernardino, M., Rodrigues, E.M., Zorzo, A.F., Marchezan, L.: Systematic mapping study on mbt: tools and models. IET Software **11**(4) (2017) 141–155
27. Dias, N.A.C., Subramanyan, R., Vieira, M., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. In: International Conference on Automated Software Engineering (ASE). (2007) 31–36
28. Offutt, J., Abdurazik, A.: Generating tests from uml specifications. In: International Conference on the Unified Modeling Language (UML'99). 416–429
29. Hessel, A., Pettersson, P.: A global algorithm for model-based test suite generation. Electronic Notes in Theoretical Computer Science **190**(2) (2007) 47–59

30. Dorofeeva, M., Koufareva, I.: Novel modification of the w-method. Bulletin of the Novosibirsk Computing Center. Series: Computer Science (18) (2002) 69–80
31. Belli, F., Hollmann, A.: Test generation and minimization with basic statecharts. In: ACM symposium on Applied computing. (2008) 718–723
32. Julliand, J., Kouchnarenko, O., Masson, P.A., Voiron, G.: Test generation from event system abstractions to cover their states and transitions. Programming and Computer Software **44**(1) (2018) 1–14
33. Takagi, T., Oyaizu, N., Furukawa, Z.: Concurrent n-switch coverage criterion for generating test cases from place/transition nets. In: Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on. (2010) 782–787
34. Cartaxo, E.G., Andrade, W.L., Neto, F.G.O., Machado, P.D.: Lts-bt: a tool to generate and select functional test cases for embedded systems. In: ACM symposium on Applied computing. (2008) 1540–1544
35. Belli, F., Hollmann, A., Kleinselbeck, M.: A graph-model-based testing method compared with the classification tree method for test case generation. In: Secure Software Integration and Reliability Improvement. (2009) 193–200
36. Devroey, X., Perrouin, G., Schobbens, P.Y.: Abstract test case generation for behavioural testing of software product lines. In: International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2. (2014) 86–93
37. Belli, F., Endo, A.T., Linschulte, M., Simao, A.: A holistic approach to model-based testing of web service compositions. Software: Practice and Experience **44**(2) (2014) 201–234
38. Anbunathan, R., Basu, A.: Dataflow test case generation from uml class diagrams. In: IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). (2013) 1–9
39. Li, N., Offutt, J.: Test oracle strategies for model-based testing. IEEE Transactions on Software Engineering **43**(4) (2017) 372–395
40. Majeed, S., Ryu, M.: Model-based replay testing for event-driven software. In: 31st Annual ACM Symposium on Applied Computing. (2016) 1527–1533
41. Yao, J., Wang, Z., Yin, X., Shiyz, X., Wu, J.: Formal modeling and systematic black-box testing of sdn data plane. In: Network Protocols (ICNP), 2014 IEEE 22nd International Conference on. (2014) 179–190
42. Belli, F., Beyazit, M., Takagi, T., Furukawa, Z.: Model-based mutation testing using pushdown automata. TRANSACTIONS on Information and Systems **95**(9) (2012) 2211–2218
43. Ali, S., Briand, L.C., Rehman, M.J.u., Asghar, H., Iqbal, M.Z.Z., Nadeem, A.: A state-based approach to integration testing based on uml models. Information and Software Technology **49**(11-12) (2007) 1087–1106
44. Satpathy, M., Yeolekar, A., Peranandam, P., Ramesh, S.: Efficient coverage of parallel and hierarchical stateflow models for test case generation. Software Testing, Verification and Reliability **22**(7) (2012) 457–479
45. Cartaxo, E.G., Machado, P.D., Neto, F.G.O.: On the use of a similarity function for test case selection in the context of model-based testing. Software Testing, Verification and Reliability **21**(2) (2011) 75–100
46. Dang, T., Nahhal, T.: Coverage-guided test generation for continuous and hybrid systems. Formal Methods in System Design **34**(2) (2009) 183–213
47. Wang, W., Sampath, S., Lei, Y., Kacker, R., Kuhn, R., Lawrence, J.: Using combinatorial testing to build navigation graphs for dynamic web applications. Software Testing, Verification and Reliability **26**(4) (2016) 318–346