2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)

Some Thoughts on Model-Based Test Optimization*

Pan Liu[1,2], Yudong Li[1]
[1]Shanghai Business School, Shanghai, China
[2]Engineering Research Center for Software Testing and Evaluation of Fujian Province, Fujian, China
panl008@163.com

Zhaojun(Steven) Li[1,3,#]
[1]Shanghai Business School, Shanghai, China
[3]Department of Industrial Engineering and Engineering Management, Western New England University, USA
stevenli777@hotmail.com

***Abstract*—As a test method, model-based testing has been proven to have the ability to find inconsistencies between a software system and its design model and objectives. Because the design of the model is independent of the development of program codes, the generation of test cases from the model is synchronized with the development of software codes, saving the software test cost. This paper reviews the development of model-based test methods from the optimization perspective and presents five issues in the test optimization process. In addition, the paper discusses four types of optimization strategies for reducing the redundancy of test sequences and an optimization method by using the regular expression model. Finally, this paper gives two potential research directions, including model optimization and software bug location in the program, for model-based testing in the future. This research contributes to the applications of model-based testing in industry.**

Keywords—test optimization; model-based testing; redundancy reduction; model optimization

I. INTRODUCTION

Computer software has become an information infrastructure that penetrates into all areas of the national economy, national defense infrastructure, and personal life. However, due to the imperfection of software testing technologies, the quality of the software is usually not satisfactory, leading to the "software trustworthy" problem [1], such as the failure of China UnionPay trading system in 2006 and the failure of the domain name resolution of the storm video website in 2009. The American institute of quality assurance has conducted a follow-up study on software testing which reveals three key points. First of all, the earlier the errors in the software are discovered, the lower the cost of software development. Secondly, the cost of modifying the software defects after encoding is 10 times that before the coding, and the cost of modifying software defects after product delivery is 10 times that before delivery. Thirdly, the higher the quality of the software, the lower the maintenance cost after the software is released.

Software testing is an important means of discovering software errors and ensuring software quality. However, the software test cost in traditional software test methods has exceeded 50% of the total software development cost [2]. To reduce software test cost, the research on new methods for software testing has become an important research direction and an emergent research topic in the field of software engineering. How to improve software testing techniques, enhance software testing efficiency, and reduce software testing costs has become an challenging research subject.

Model-based testing [3-6] can generate test cases from the behavioral model of the software system. Then testers can observe whether the actual operation of the software system conforms to its expected behavior model. So this test method is also called conformance testing. To build the model of the expected behavior of the software under testing, people generally used some description tools such as the visual modeling language or the formal specification language to describe the operating states and processing of the software. Compared to test cases generated from program codes, test cases generated from the model are more systematic and representative, and can be easily used to detect errors in complex systems. Besides, testers can also use models to quickly locate software errors in program modules. Last but not least, model-based testing realizes simultaneously between the program code development and test case generation. So it changes the paradigm of "now programming, later testing" in software engineering, allowing testing to be completed in parallel throughout the software lifecycle.

Due to the advantages of model-based testing, this test method has evolved tremendously over the past few decades, forming a list of test theories, test methods, and test support tools. The paper studies the method of model-based testing from the optimization perspective and focuses on model-based test optimization strategies from two perspectives. On the one hand, we discuss how to reduce the redundancy in test cases obtained by the model-based test method. On the other hand, we study how to enhance the modeling capabilities of the model so that the improved model can represent more types of software behaviors. Improving the software modeling method and building a testability model can greatly simplify the process of software development, test case generation, and conformance testing. Therefore, it is very important to study the methods of model-based test optimization generation for enhancing the model-based test theory and popularizing its application in industry.

The rest of the paper is structured as follows. Section 2 reviews existing theories of the model-based test method. Section 3 discusses some optimization issues in model-based testing. Section 4 studies some strategies of model optimization and the test generation method based on the optimization model. Section 5 introduces the ERE-based test method.

*This work is supported by National Natural Science Foundation of China (NSFC) under grant (No. 61502299) and the project of business scholars in Shanghai Business School.
#Corresponding author: Zhaojun(Steven) Li (stevenli777@hotmail.com)

978-1-7281-3925-8/19/$31.00 ©2019 IEEE
DOI 10.1109/QRS-C.2019.00058

Section 6 summarizes the paper and discusses future research directions.

II. MODEL-BASED TESTING

The idea of model-based testing comes from the 1956 Moore's description of the automaton. In the following decades, new theoretical studies on model-based testing were carried out, forming two forms of Moore automata [7] and Mealy automata [8].

Definition 1 (Moore automaton). A Moore automaton (Mo for short) can be defined as Mo =< S, I, s_0 ,T, f > , where

- S: Set of finite states;
- I: Set of input conditions;
- s_0: A initial state of Mo, and $s_0 \in S$;
- T: Set of termination (accepted) states of Mo, and $T \subseteq S$;
- f: State transition function $f : S \times I \rightarrow S$.

Definition 2 (Mealy automaton). A Mealy automaton (Me for short) can be defined as Me＝<I, O, S, δ, λ>, where

- I: Set of finite non-empty input symbols,
- O: Set of finite non-empty output symbols,
- S: Set of finite states,
- δ: S×I→S is a state transition function, and
- λ: S×I→O is an output function.

From definitions 1 and 2, both Moore and Mealy automata can describe the state transition process in the system. But there is a fundamental difference between the two types of automata. A Moore automaton only contains a state transition function $f : S \times I \rightarrow S$ and doesn't produce an output on states. Different from Moore automata, there are some outputs on states in a Mealy automaton. As a result, the state transition process is accompanied by a series of outputs on states in the Mealy automaton.

In the past, researches referred to both the Moore and Mealy automata as finite state machines (FSMs). Model-based testing generates test cases and test execution based primarily on the FSM model. Due to the difference between the Moore automaton and the Mealy automaton, we are able to use different types of automata to model different systems. When there is no obvious output on states of the system, the Moore automaton can be used to model the system, otherwise, the Mealy machine is more suitable for modeling the system. For example, when modeling the web login system, the system often does not contain output, so it is suitable to use the Moore machine to model. For telecommunication systems, the Mealy machine is more suitable for modeling because some outputs are found on states of the system.

Definition 3(transition). A transition of Mo/Me is defined as a triplet $t = <s_i , s_j , x_{ij}>$, where $s_i, s_j \in S$, $x_{ij} \in I$ and $f(s_i , x_{ij}) = s_j$ or $\delta(s_i , x_{ij}) = s_j$.

Definition 4 (test sequence) A test sequence of *Mo or Me* consists of zero or more continuous transitions, which can be formally defined as

$ts = \varepsilon \mid t \mid ts.t \mid t.ts,$

where ε denotes an empty transition [9], t is a transition, the symbol "." denotes the concatenation of between two transitions.

Generally, an FSM can be taken as a graph on the computer. So, some graph traversal algorithms, such as the depth traversal algorithm and the breadth traversal algorithm, can be used to traverse the FSM to generate a set of test sequences. Finally, the tester can observe whether the software execution paths are consistent with these transition sequences generated from the model and whether the accompanying outputs of the system under testing are also consistent with the expected output sequences generated from the model.

III. MODEL-BASED TEST OPTIMIZATION PROBLEMS

In 1978, Chow [10] proposed the W method that is a classical model-based test method and applied this method to the test of a telecommunications switching system. This method can detect whether the software is executed according to the expected model. In the following decades, some model-based test methods were proposed with the goal of test optimization. In general, there are some key scientific issues related to model-based test optimization.

What kind of model can facilitate the generation of an optimized set of test cases?

As system functions and complexity increase, modeling for software behavior becomes very difficult. Because modeling focuses on describing user requirements rather than focusing on test case generation, it is easy to generate a large number of redundant test cases from the model, which increase the cost of software testing. Some researchers [11-13] considered the use of formal methods for modeling of the software system, which enables the model to have a capability for testing and verification automatically. In addition, the formal model is more suitable for the automatic execution of test cases so that improve the controllability of the test process.

How to decompose and abstract the test model to avoid state explosion and obtain an optimized set of test cases?

In a parallel system or component combination system, the number of system states is very large and the relationship among states is extremely complex. So it is hard or even impossible to establish the global state space of the system. The state space explosion problem [14] makes model-based testing methods difficult to apply in industry. The "divide and conquer" strategy [15-17] is an effective way to solve the state space explosion problem. Pan Liu et al. [18, 19] proposed the model decomposition method to transform a complex model into several simple models. Then test cases can be obtained from those simple models, which makes the testing process easier.

What technology is used to generate the desired set of test cases in a targeted manner?

Generally, software testing is conducted for some certain test intents, and test cases generated from the model are to

satisfy given test coverage criteria. However, test coverage criteria are not exactly equivalent to test intents. According to the 2-8 law of software error distribution, the part of the software with errors often contains other types of errors. So test cases designed for the part containing software errors are likely to find more errors. To find those parts containing software errors, the test scenario needs to be distinguished first. So the scenario specification-based test method [20, 21] was proposed to generate test cases to test this software scenario. This test method not only can effectively reduce the space of the test source but also test cases generated from the scenario specification can meet the specific test intent. However, it is difficult to locate all software errors in program codes because some of the errors are generated from the communication process of different scenarios.

How to optimize the test case generation process to eliminate redundant test cases?

Redundancy of test cases is a reason that restricts the application of model-based testing in the industry. Generally, most model-based test generation methods need to convert a model into a test tree [22, 23], then test sequences are obtained from the root to leaf nodes of the test tree. When the model is complex, the depth and width of the test tree will increase greatly. Different leaf nodes may have many common ancestors, resulting in a large amount of redundancy among test sequences. Redundancy of test cases not only increases the burden of test case instantiation [24] but also leads to the increase of test costs. Optimized test coverage criteria can produce a set of non-redundant or less redundant test cases. Thus effective redundancy reduction strategies need to be studied to reduce the redundancy of test cases that meet a specific test coverage criterion, that is to say, both retained test cases and original test cases have the same software's error detection capability. The optimization of the test set is the key technology to solve the problem of test redundancy [25].

How to effectively implement the instantiation of test cases and automatically execute test cases?

The instantiation of test sequences is one of the important aspects of model-based testing. A test sequence corresponds to a test execution trace. Designing efficient test cases is key for revealing software errors and automatic test execution. However, the existing methods of test sequence instantiation require manual participation, which makes model-based testing difficult to be automated [2]. Therefore, improving the process of instantiation of test sequences is a challenge in model-based testing. This is also the key to the model-based test method being widely accepted by the industry.

IV. OPTIMIZATION METHODS

To optimize model-based test generation, some new modeling theories, such as constraint logic programming, transition systems, and Petri nets, were proposed [26-28] since the 1980s. In the late 1980s, many model-based test methods were proposed to implement consistency testing for communication systems. Sidhu and Leung [29] studied several model-based test methods on an actual protocol system and evaluated the test efficiency of these test methods. With the rise of object-oriented software development methods, researchers used FSM to build the behavior model of object-oriented software and realize the model-based test method. For example, Kung et al. [30] modeled and tested object-oriented programs using FSM. In the early 21 century, with the popularity of the Internet, those methods for testing Web applications and Web service portfolios were gradually being proposed. For example, researchers studied how to produce a set of test cases from the UML model [31-33]. In 2005, Andrews and Offutt et al. [34] presented the hierarchical method for modeling Web applications. Their method divides a web application into different levels and generates test cases from the low-level FSM. Then, many researchers studied the extended method of the FSM model and build the EFSM model for generating test cases [35]. In addition, the way of model checking was proposed to obtain counterexamples and then test cases can be generated from counterexamples [36]. After 2010, hybrid test methods combining different models and technologies were proposed for software testing. For example, a combination of state-based and scene-based test methods [37, 38] was proposed to generate test paths. Some model-based test support tools or tool prototypes have also been developed [3, 34, 39], and the evaluation of model-based test methods [40, 41] was given to promote model-based testing into the industry. We summarize four types of model-based test optimization strategies.

The first type of test optimization strategy focused on the study of reducing the total length of test sequences generated from the model. In view of the problem that the total length of test sequences is too long for the W method, Fujiwara et al. [42] proposed the Wp method and theoretically proved that the Wp method has the same error detection capability as the W method. Then, Zhang Yong et al. [43] proposed the R-Wp method, which further reduces the total length of test sequences generated from the Wp method. They have also proved that the R-Wp method has the same error detection ability as the Wp method. Liu Pan et al. improved the core algorithm in the above three methods [11]. The new algorithm reduces the generation time of the test sequence set and optimizes the composition of the test sequence set.

To further save test costs, the second type of test optimization strategy that was studied to reduce the total length and the total number of test sequences simultaneously. Sabnani and Dahbura [44] proposed the Unique Input/Output (UIO) method based on the characteristics of the UIO sequence in the model. This method not only reduces the total length of the test sequence but also reduces the number of test sequences. However, Vuong et al. [45] found that the UIO method does not guarantee to reveal all errors in the application. So, they proposed the UIOv method. Although the UIOv method can find more software errors than the UIO method, it does not reduce the total length and the total number of test sequences compared to the W method and the Wp method. Alfred [46] et al. proposed a method of traversing

the model to generate a UIO sequence by using the solution of the traveler problem, reducing the number of test sequences to a unique one. The DS method [29] based on the distinguishing sequence [12] was also proposed. This method can effectively reduce the total number and the total length of the test sequences of both the W method and the Wp method. The disadvantage of the DS method is that there must be a distinguishing sequence in the model. To test web applications, Liu Pan et al. [47] proposed a new test sequence optimization generation method. The method generates test cases according to the characteristics of the browser. Since the browser has the functions forward, back, and go history, Liu Pan et al. combined test sequences by using these functions of the browser. Compared with the previous methods, the proposed method considers the characteristics of the application environment.

The third type of test optimization strategy is concentrated in the study of the redundancy reduction techniques for the test suite. In the past, two types of redundancy reduction techniques were highlighted. One is a redundancy reduction technology in the test generation process. Liu Pan et al. [11] proposed a redundancy reduction method based on the method of string matching, which makes the newly generated test sequence to these existing sequences so as to achieve the purpose of deleting redundant sequences. A feature of the method is to ensure that the reduced test sequence set still meets a selected test coverage criterion. The second redundant reduction technique is proposed for reducing the redundancy in a test suite [48-50]. Some algorithms, including heuristic methods, simulated annealing algorithm, and ant colony algorithm, were designed to realize the reduction of test suites. Duan and Chen [51] concatenated the overlapping portions of different test sequences, reducing redundancy in the test sequences.

The fourth type of optimization strategy is mainly for the research of test source optimization. Wang et al. [33] used the UML sequence diagram to obtain a test scenario. Test cases can be generated from the test scenario constrained the test source. Zhang and Xu et al. [52] reduced the test requirements according to the relationship between test requirements and generated test cases from the reduced test requirements. Li et al. [53] divided a behavioral phase cluster according to the transition conditions of the output signal of the circuit and then used the behavioral phase clustering of FSM to generate the diagnosis of the gate-level fixed fault of the test sequence test circuit. The essence of the method is to classify the test source. Liu Pan et al. [41] proposed a test generation method based on regular expression decomposition, which not only reduces the space complexity of the model but also can generate more effective test cases from the model for consistency testing.

We summarize the characteristics of the four optimization strategies in table I.

TABLE I. A COMPARISON TABLE OF OPTIMIZATION STRATEGIES

Optimization Type	Intent	Method
Strategy 1	Reduce the total length of test sequences.	W, WP, R-WP
Strategy 2	Reduce the total length and total number of test sequences.	UIO, UIOv, DS
Strategy 3	Redundancy reduction of the test suite.	String matching algorithms, heuristic algorithms, simulated annealing algorithm, and ant colony algorithm
Strategy 4	Test resource optimization	UML, test scenario modeling, behavioral phase cluster, model decomposition

V. MODEL EXTENSION

Model-based testing [12, 54, 55] is an effective automatic testing method. Usually, the finite state machine (Finite State Machine, FSM) is used to establish the behavior model of the software. Although the FSM model can automatically generate test cases, it is not suitable for building the testability model [56] of software. The testability model should have powerful modeling capability, making it easier to generate test cases and locate software bugs in programs. Due to the limitations of the modeling capability, FSM cannot model for all network software [57], such as concurrent systems. The concurrent system is composed of multi-programming, multi-processing, multi-tasking and distributed systems [58, 59]. Modeling concurrent systems not only need to consider the complex relationships in the system, such as selection, connectivity, concurrency, synchronization, and alternation [60] but also researches the controllability of the system during testing [61].

In the past, researchers often used a Petri net to model the concurrent system. For example, Lai [62] and Wu et al. [63] used Petri nets to model the concurrent system, and then test cases were generated from the Petri net model of software to perform the reachability test [64]. However, compared with FSM, it is easier to generate the state space explosion problem when traversing a Petri net of the complex system [65]. In addition, regular expressions are also applied to the modeling of concurrent systems, forming regular expressions such as UNIX patterns [66], path expressions [67], behavioral expressions [68], and extended regular expressions [69]. However, most of these studies are to validate the model, not for test generation, so it is difficult to construct a testability model for software behavior.

To simplify the modeling process, the regular expression model was adopted to build the model of software behaviors. In the past, modeling research on regular expressions mainly focused on the transformation between FSM and regular expressions [66]. Usually, the input conditions in FSM are used to construct regular expressions. To construct a regular expression for software behavior, the FSM model needs to be

constructed first and then it can be converted to the corresponding regular expression model. Qian [70] discussed the conversion rules between FSM and regular expressions and designed a related conversion algorithm. Liu Pan et al. [19] have designed an algorithm to obtain the regular expression model of software behavior and then generated some high-quality test cases by the way of model decomposition. Li and Moon [67] proposed a correlation algorithm for constructing regular path expressions for XML format data, which improves the search efficiency of XML files. Garg and Ragunath [68] discussed the relationship between Petri nets and regular expressions and extended the interaction of traditional regular expressions. Sen and Roşu [69] extended a non-operation in traditional regular expressions and optimized the extended regular expressions. Milner [71] uses regular expressions to model the behavior of communication systems and establish a set of communication system calculus rules. The most influential regular expression modeling study is the unified modeling theory for programming based on the Turing Award winner Hoare [72].

A new type of model, the extended regular expression (ERE) model [9, 73], was built to model software behaviors with complex operations. Compared with FSM, ERE not only has the ability to describe complex software behavior but also the established model is more concise and has excellent reasoning and verification capabilities. Thus, the ERE model is more suitable for test case generation and modeling complex systems, and the ERE-based test method was also proposed to performing conformance testing [59]. Liu Pan et al. [59, 74] extended the study of regular expressions and established a modeling theory based on extended regular expressions. They then applied the ERE model to the modeling of the program and the program bug location [73]. In the experiment, they succeeded in discovering errors inserted in the program through the derivation of formal methods. Recently, they used the extended regular expression model to build a hierarchical model of the program and obtained a suite of program slices through model decomposition [75]. A tool [9] has also developed by them to support the software modeling and test generation based on the extended regular expression.

We construct table II to compare different modeling methods in the process of model extension.

TABLE II. A COMPARISON TABLE OF MODEL EXTENSION

Model	Advantage
FSM	Theory of automatic
Petri net	Modeling for the concurrent system
Regular expression	String operations
Extended regular expression	Describe complex software behavior and model reasoning.

VI. CONCLUSION

Over the past 40 years, model-based testing has formed a complete theoretical system and a series of related tools were developed to support model-based software testing. In the advancement process, test optimization plays an important role to improve the effectiveness of this test method. Research topics have also evolved from early test case redundancy reduction methods to recent studies of model testability, from FSM modeling to Petri net, regular expressions, and extended regular expression modeling method. This paper reviews the development of model-based testing, with test optimization as the main focus, and discusses related research challenges. In the future, we conjecture that model-based testing will encounter the following continuing challenges.

1) Continuous optimization of the model. Users' behavior is very rich and diverse, hence new models need to be developed to build the model tackling both software behaviors and users' behaviors.

2) Error location problem. Traditional model-based testing methods use models to describe the abstract behavior of the system, regardless of the system's codes. Therefore, when there is an error in the system, it is not possible to directly locate the specific location in program codes. Therefore, it is necessary to study how to combine the model-based test method with the method of the software error location so as to realize a holistic process of modeling, testing, and the software error location.

REFERENCES

[1] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," *HASP@ ISCA,* vol. 11, 2013.

[2] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, 2007, pp. 85-103.

[3] W. Li, F. L. Gall, and N. Spaseski, "A Survey on Model-Based Testing Tools for Test Case Generation," in *International Conference on Tools and Methods for Program Analysis*, 2017, pp. 77-89.

[4] M. Blackburn, R. Busser, and A. Nauman, "Why model-based test automation is different and what you should know to get started," in *International conference on practical software quality and testing*, 2004, pp. 212-232.

[5] J. Gläscher, R. Adolphs, and D. Tranel, "Model-based lesion mapping of cognitive control using the Wisconsin Card Sorting Test," *Nature communications,* vol. 10, p. 20, 2019.

[6] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated Model-Based Testing of Mobile Apps," *Software, IEEE,* vol. 32, pp. 53-59, 2015.

[7] H. Löding and J. Peleska, "Timed moore automata: test data generation and model checking," in *2010 Third International Conference on Software Testing, Verification and Validation*, 2010, pp. 449-458.

[8] T. Godin, I. Klimann, and M. Picantin, "On torsion-free semigroups generated by invertible reversible Mealy automata," in *International Conference on Language and Automata Theory and Applications*, 2015, pp. 328-339.

[9] P. Liu and Z. Xu, "MTTool: A Tool for Software Modeling and Test Generation," *IEEE Access,* vol. 6, pp. 56222-56237, 2018.

[10] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE transactions on software engineering,* pp. 178-187, 1978.

[11] H. Miao, P. Liu, J. Mei, and H. Zeng, "A new approach to automated redundancy reduction for test sequences," in *Dependable Computing,*

2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on, 2009, pp. 93-98.

[12] P. Liu, H.-K. Miao, H.-W. Zeng, and Y. Liu, "FSM-based testing: Theory, method and evaluation," *Jisuanji Xuebao(Chinese Journal of Computers),* vol. 34, pp. 965-984, 2011.

[13] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, and W. E. Wong, "Model-based mutation testing–approach and case studies," *Science of Computer Programming,* 2016.

[14] P. Liu, H. Miao, H. Zeng, and J. Mei, "DFSM-Based Minimum Test Cost Transition Coverage Criterion," *Journal of Software,* vol. 22, 2011.

[15] R. Alur, A. Radhakrishna, and A. Udupa, "Scaling enumerative program synthesis via divide and conquer," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2017, pp. 319-336.

[16] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Transactions on Mathematical Software (TOMS),* vol. 42, p. 13, 2016.

[17] M. Uzam, Z. Li, G. Gelen, and R. S. Zakariyya, "A divide-and-conquer-method for the synthesis of liveness enforcing supervisors for flexible manufacturing systems," *Journal of Intelligent Manufacturing,* vol. 27, pp. 1111-1129, 2016.

[18] P. Liu, H. Miao, H. Zeng, and L. Cai, "An Approach to Test Generation for Web Applications," *International Journal of u-and e-Service, Science and Technology,* vol. 6, pp. 61-76, 2013.

[19] P. Liu and H. Miao, "A new approach to generating high quality test cases," in *Test Symposium (ATS), 2010 19th IEEE Asian*, 2010, pp. 71-76.

[20] J. Bach, S. Otten, and E. Sax, "Model based scenario specification for development and test of automated driving functions," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 1149-1155.

[21] Y. Choi and T. Byun, "Constraint-based test generation for automotive operating systems," *Software & Systems Modeling,* vol. 16, pp. 7-24, 2017.

[22] T. Kitamura, A. Yamada, G. Hatayama, C. Artho, E.-H. Choi, N. T. B. Do*, et al.*, "Combinatorial testing for tree-structured test models with constraints," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 141-150.

[23] S. Herfert, J. Patra, and M. Pradel, "Automatically reducing tree-structured test inputs," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 861-871.

[24] S. Benz, "AspectT: aspect-oriented test case instantiation," in *Proceedings of the 7th international conference on Aspect-oriented software development*, 2008, pp. 1-12.

[25] D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, "On-demand test suite reduction," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 738-748.

[26] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. Yap, "The CLP (R) language and system," *ACM Transactions on Programming Languages and Systems (TOPLAS),* vol. 14, pp. 339-395, 1992.

[27] R. D. Nicola, "Models and operators for nondeterministic processes," in *Mathematical Foundations of Computer Science*, 1984, pp. 433-442.

[28] N. Ran, H. Su, and S. Wang, "An improved approach to test diagnosability of bounded Petri nets," *IEEE/CAA Journal of Automatica Sinica,* vol. 4, pp. 297-303, 2017.

[29] D. Sidhu and T. K. Leung, "Formal methods for protocol testing: a detailed study," *Software Engineering IEEE Transactions on,* vol. 15, pp. 413-426, 1989.

[30] D. Kung, J. Gao, H. Pei, and Y. Toyoshima, "A test strategy for object-oriented programs," in *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, 1995, pp. 239-244.

[31] Y. G. Kim, H. S. Hong, D.-H. Bae, and S. D. Cha, "Test cases generation from UML state diagrams," *IEE Proceedings-Software,* vol. 146, pp. 187-192, 1999.

[32] A. Abdurazik and J. Offutt, "Using UML collaboration diagrams for static checking and test generation," in *International conference on the unified modeling language*, 2000, pp. 383-395.

[33] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, "Generating test cases from UML activity diagram based on gray-box method," in *11th Asia-Pacific software engineering conference*, 2004, pp. 284-291.

[34] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing Web applications by modeling with FSMs," *Software & Systems Modeling,* vol. 4, pp. 326-345, 2005.

[35] Y. Zhang, L. Q. Qian, and Y. F. Wang, "Automatic Testing Data Generation in the Testing Based on EFSM," *Chinese Journal of Computers,* pp. 1295-1303, 2003.

[36] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar, "Generating tests from counterexamples," in *Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 326-335.

[37] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp*, et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software,* vol. 86, pp. 1978-2001, 2013.

[38] W. Grieskamp, "Multi-paradigmatic model-based testing," in *Formal Approaches to Software Testing and Runtime Verification*, ed: Springer, 2006, pp. 1-19.

[39] A. Belinfante, L. Frantzen, and C. Schallhart, "14 Tools for Test Case Generation," *Cryobiology,* vol. 24, pp. 578-579, 2004.

[40] A. D. Neto, R. Subramanyan, M. Vieira, G. H. Travassos, and F. Shull, "Improving evidence about software technologies: A look at model-based testing," *IEEE software,* vol. 25, pp. 10-13, 2008.

[41] P. Ammann and J. Offutt, *Introduction to software testing*: Cambridge University Press, 2016.

[42] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on software engineering,* vol. 17, pp. 591-603, 1991.

[43] Y. ZHANG, L.-Q. QIAN, and Y.-F. WANG, "TEST SEQUENCES SELECTION BASED ON DETERMINISTIC FINITE-STATE MACHINES [J]," *Journal of Computer Research and Development,* vol. 9, 2002.

[44] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN systems,* vol. 15, pp. 285-297, 1988.

[45] S. T. Vuong, "The UIOv-method for protocol test sequence generation," in *Proc. 2nd IFIP Int. Workshop on Protocol Test Systems (IWPTS'89)*, 1989, pp. 161-175.

[46] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE transactions on communications,* vol. 39, pp. 1604-1615, 1991.

[47] B. Song and H. Miao, "Modeling web applications and generating tests: A combination and interactions-guided approach," in *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, 2009, pp. 174-181.

[48] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 2, pp. 270-285, 1993.

[49] H. Zhong, L. Zhang, and H. Mei, "An experimental comparison of four test suite reduction techniques," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 636-640.

[50] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on software Engineering,* vol. 33, 2007.

[51] L. Duan and J. Chen, "Reducing test sequence length using invertible sequences," in *International Conference on Formal Engineering Methods*, 2007, pp. 171-190.

[52] Z. Xiaofang, X. Baowen, N. Changhai, and S. Liang, "An Approach for Optimizing Test Suite Based on Testing Requirement Reduction [J]," *Journal of Software,* vol. 18, pp. 821-831, 2007.

[53] H. Li, Y. Min, and Z. Li, "Clustering of behavioral phases in FSMs and its applications to VLSI test," *Science in China Series F: Information Sciences,* vol. 45, pp. 462-478, 2002.

[54] M. Markthaler, S. Kriebel, K. S. Salman, T. Greifenberg, S. Hillemacher, B. Rumpe*, et al.*, "Improving model-based testing in automotive software engineering," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2018, pp. 172-180.

[55] E. Villani, R. P. Pontes, G. K. Coracini, and A. M. Ambrósio, "Integrating model checking and model based testing for industrial software development," *Computers in Industry,* vol. 104, pp. 88-102, 2019.

[56] P. Liu, "Testability Metrics for Software Behavioral Models," *International Journal of Performability Engineering,* vol. 13, 2017.

[57] J. I. Perna and C. George, "Model checking RAISE applicative specifications," in *Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2007)*, 2007, pp. 257-268.

[58] E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen*, et al.*, "VCC: A practical system for verifying concurrent C," in *International Conference on Theorem Proving in Higher Order Logics*, 2009, pp. 23-42.

[59] P. Liu and H. Miao, "Theory of Test Modeling Based on Regular Expressions," in *Structured Object-Oriented Formal Language and Method*, ed: Springer, 2014, pp. 17-31.

[60] H. Zhao and J. SUN, "An algebraic model of Internetware software architecture," *Scientia Sinica Informationis,* vol. 43, pp. 161-177, 2013.

[61] W. WANG, C.-l. DU, and H.-l. ZHANG, "Reachability testing algorithm for parallel program in heterogeneous network environment," *Journal on Communications,* p. 26, 2006.

[62] R. Lai, "A survey of communication protocol testing," *Journal of Systems and Software,* vol. 62, pp. 21-46, 2002.

[63] N. Q. Wu and M. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *Automation Science and Engineering, IEEE Transactions on,* vol. 9, pp. 446-454, 2012.

[64] M. Notomi and T. Murata, "Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis," *IEEE Transactions on software engineering,* vol. 20, pp. 325-336, 1994.

[65] F. Chu and X. L. Xie, "Deadlock analysis of Petri nets using siphons and mathematical programming," *IEEE Transactions on Robotics & Automation,* vol. 13, pp. 793-804, 1997.

[66] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*: Pearson Education India, 2008.

[67] Q. Li and B. Moon, "Indexing and querying XML data for regular path expressions," in *VLDB*, 2001, pp. 361-370.

[68] V. K. Garg and M. Ragunath, "Concurrent regular expressions and their relationship to Petri nets," *Theoretical Computer Science,* vol. 96, pp. 285-304, 1992.

[69] K. Sen and G. Roşu, "Generating optimal monitors for extended regular expressions," *Electronic Notes in Theoretical Computer Science,* vol. 89, pp. 226-245, 2003.

[70] Z. Qian, "Model-based approaches to generating test cases for web applications," Ph. D. thesis, Shanghai University, 2008.

[71] R. Milner, *Communicating and mobile systems: the pi calculus*: Cambridge university press, 1999.

[72] C. A. R. Hoare, "Unified theories of programming," in *Mathematical methods in program development*, ed: Springer, 1997, pp. 313-367.

[73] P. Liu, J. Ai, and Z. J. Xu, "A study for extended regular expression-based testing," in *Computer and Information Science (ICIS), 2017 IEEE/ACIS 16th International Conference on*, 2017, pp. 821-826.

[74] M. Zeng, P. Liu, and H. Miao, "The Design and Implementation of a Modeling Tool for Regular Expressions," in *Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on*, 2014, pp. 726-731.

[75] P. Liu, Z. Xu, and J. Ai, "An Approach to Automatic Test Case Generation for Unit Testing," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2018, pp. 545-552.