2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops

Successive Refinement of Models for Model-Based Testing to Increase System Test Effectiveness

Ceren Şahin Gebizli
Vestel Electronics,
Manisa, Turkey
ceren.sahin@vestel.com.tr

Hasan Sözer and Ali Özer Ercan
Ozyegin University,
Istanbul, Turkey
{hasan.sozer, ali.ercan}@ozyegin.edu.tr

***Abstract*—Model-based testing is used for automatically generating test cases based on models of the system under test. The effectiveness of tests depends on the contents of these models. Therefore, we introduce a novel three-step model refinement approach. We represent test models in the form of Markov chains. First, we update state transition probabilities in these models based on usage profile. Second, we perform an update based on fault likelihood that is estimated with static code analysis. Our third update is based on error likelihood that is estimated with dynamic analysis. We generate and execute test cases after each refinement. We applied our approach for model-based testing of a Smart TV system and new faults were revealed after each refinement.**

***Keywords*-model-based testing, model refinement, statistical usage testing, risk-based testing, industrial case study, software test automation**

I. Introduction

The size and complexity of software systems are continuously increasing. This trend makes it infeasible to test systems exhaustively. This is in particular an issue in the consumer electronics domain due to limited resources and high reliability expectations of end users. Systems in this domain become increasingly software-intensive. For instance, Smart TV systems are mainly controlled by software, comprising tens of millions of lines of code. As such, software faults become a primary threat for reliability and they have to be revealed effectively.

Model-based testing (MBT) [1], [2] is used for automatically generating test cases based on models that represent the desired behavior of the system under test. This automation decreases the test preparation time and helps to achieve increased and measured coverage of possible execution scenarios. It is usually not feasible to achieve full-path coverage on large-scale models that represent real systems. For example, models that are used for MBT of Smart TVs comprise thousands of states and transitions. Hereby, contents of the model determine the effectiveness of the generated test cases in revealing faults.

In this paper, we introduce a new model refinement approach. In our approach, we represent test models in the form of Markov chains. These models comprise a set of states and a set of state transitions that are annotated with probability values. These values steer the test case generation process, which aims at covering the most probable paths. We refine these models in three steps. First, we update transition probabilities based on a collected usage profile. Second, we update the resulting models based on fault likelihood at each state, which is estimated based on static code analysis. Third, we perform updates based on error likelihood at each state, which is estimated with dynamic analysis. We generate and execute test cases after each refinement step. We applied our approach in the context of an industrial case study for MBT of a Smart TV system. We observed promising results, in which new faults were revealed after each refinement.

The remainder of this paper is organized as follows. In the following section, we summarize related work. In Section III, we introduce our approach. In Section IV, we discuss the application of our approach in the context of an industrial case study and present the results. Finally, in Section V, we conclude the paper with a discussion of future work.

II. Related Work

MBT techniques are extensively studied in the literature [3]. There exist tools applied in practice [4] and different formalisms for model specification like finite state machines [5] [6] and Markov chains [7]. There also exist case studies [4], [8] for evaluating the effectiveness of MBT. In this paper, we evaluate the effectiveness of MBT in consumer electronics domain. In particular, we evaluate the effectiveness of a successively refined model.

Recent studies on MBT [9], [10] take the timing behavior and concurrency into account. Petri nets are used for specifying test models [10] and test oracles are automatically generated to enforce timing requirements [9]. The generated test cases reflect the timed and concurrent nature of inputs for the system. In our approach, we did not take the timing behavior and concurrency into account. However, we make the failure behavior explicit in test models.

There exist recent studies [11], [12], [13] on refinement of test models for MBT. These studies aim at extending existing models by adding missing states and transitions. Missing model elements are discovered by analyzing run-time states and events collected from the executing system. In our approach, we assume that the model is complete; however, we

978-1-5090-3674-5/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSTW.2016.10

IEEE computer society

update transition probabilities to focus the generated test cases on different parts of the model. These updates are performed not only based on dynamic analysis, but also usage profile and static code analysis.

We have previously proposed another approach for model refinement [14] based on risk of error calculated with dynamic analysis. This approach is also inspired from the principles of risk-based testing [15]. However, we complement dynamic analysis with usage profile and static analysis for estimating risks.

III. Successive Model Refinement Approach

The overall approach is depicted in Figure 1. We assume that a model of the system exists in the form of a Markov chain. First, we generate test cases based on this model. We use the MaTeLo tool[1] for this purpose. The test case generation process focuses on covering the most probable paths on the model according to transition probabilities. Next, the generated test cases are executed on the system. We collect a memory usage profile during test case execution. After test case execution, we update the model. This cycle of model update, test case generation and test execution is repeated three times. Each time, the model is updated based on a different source of information. In the first update cycle (5.1), usage profile is used for assigning a probability value to each transition. As a result, the generated test cases after this update will focus on execution paths that are mostly visited by the users of the system. In the second update (5.2), we exploit static analysis alerts generated for the source code. We use Klocwork[2] as the static analysis tool and we use alerts generated by this tool to estimate the relative risk of faults [16] being present for different software modules. These modules are associated with different states in the model based on the represented feature and utilization of modules for that feature. As such, we manually map them to states and update the model based on estimated risks. In the third update (5.3), we exploit the memory usage profile that is collected during the previous test cycles. This profile reveals test execution paths that lead to memory leaks, and it is used for estimating the relative risk of errors. We perform a further update on the model based on this estimation. In the following, we explain how the model updates are performed in each iteration.

Initially, we assume that all the probability values for outgoing transitions of a state are equal to each other, adding up to 1. Hence, if there are n transitions going out from a state, the probability value for each of these transitions is $1/n$. In the first update cycle, we utilize a previously collected usage profile (See Section IV). Since, the test model represents the usage behavior, we can calculate the number of visits made to each state of this model. Then, we assign probability values to each outgoing transition based on the ratio of visits made to its target state. Assuming that there are n transitions going out from a state to n different states, we calculate the number

[1]http://www.all4tec.net

[2]http://www.klocwork.com/

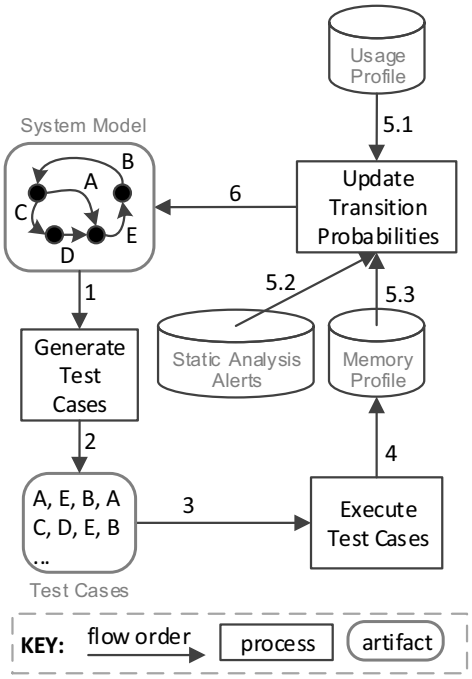

Fig. 1: The overall approach.

of visits made to each of these states i, as v_i. Then, the probability value assigned for each transition targeting at state i is $v_i / \sum_{i=0}^{n-1} v_i$.

In the second update cycle, we utilize alerts generated by a static code analysis tool working on the system source code. We map source code modules to different system states in the model. Then, for each state, we calculate the ratio of alerts that are associated with this state. We consider this ratio as a relative risk of fault, which can lead to an error if the corresponding alerts are not false positives and if the identified potential faults are triggered during the visit to the state. We update the test model according to this calculated risk as follows. We introduce a new state to the model, namely an *error state*, *E*. Then we introduce transitions from each of the existing system states to *E*. If there were previously n outgoing transitions from a state s, the number of outgoing transitions becomes $n+1$. We assign the probability value, p to this new transition according to the risk for s, calculated as the ratio of alerts associated with s to the total number of alerts. For each of the other outgoing transitions from s, the probability value is multiplied by $(1-p)$. Figure 2 illustrates such an update with a simple example. On the left hand side of the figure, we see a part of the model before the update. Hereby, we see a source state s and 3 target states t_0, t_1 and t_2, with transition probabilities 0.2, 0.3 and 0.5, respectively. On the right hand side of the figure, we see the updated model, where a new state E is introduced. The probability value for

the transition targeting at E is calculated as 0.2. Hence, all the other transitions are multiplied by 0.8. This update guarantees that the probability values for all the outgoing transitions add up to 1.

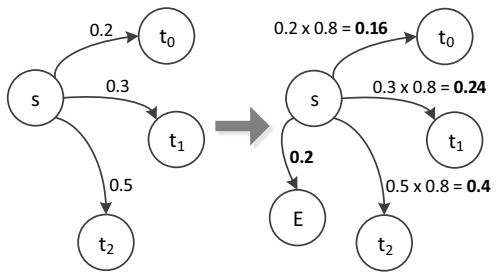

Fig. 2: The updated transition probabilities for outgoing transitions of state s after introducing a risk of error with probability 0.2.

In the third update cycle, we utilize a memory profile to calculate another risk [14]. This profile is collected during the previous test cycles. We use a previously developed tool [14], which measures memory utilization before and after the usage of each feature in the system. The difference between the two measurements reveals memory leakage. Features are mapped to the states of the model and a memory error probability is calculated for each state, which is proportional to the amount of memory leak caused by each feature. Then, the model is updated just like the previous update cycle. An additional *error state* is introduced. Transitions from all the states to this state are annotated with the calculated error probabilities. Finally, all the other transitions are updated to keep probability values for outgoing transitions sum up to 1.

In the following, we introduce a case study and illustrate each step of the approach. We also share and discuss the results obtained.

IV. Industrial Case Study

In this section, we introduce a case study from the consumer electronics domain. In particular, we will illustrate the application of our approach for MBT of Smart TV systems developed by Vestel[3], which is one of the largest TV manufacturers in Europe. Smart TV systems are highly cost sensitive and they are subject to short development time periods. The market is highly competitive and end users are less tolerant to failures [17]. Hence, approximately %35 of product development is dedicated for testing. Unlike safety-critical systems, not all the failures are critical in this domain; however, *user perceived failure severity* is important, which is defined as the level of irritation experienced by the user caused by a product failure [17]. Therefore, the testing process must be optimized to detect faults that lead to user-perceived failures.

Figure 3 depicts the model that is used in our case study. This model was previously developed by the software test group in the company. The model is defined in the form of a Markov chain with the MaTeLo tool. It constitutes a hierarchical structure, in which states can further comprise sub-models. In Figure 3, we see the top level model, where states mainly represent different features of the system. The list of main features include Electronic Program Guide (EPG), Media Browser (MBR) Video, MB Audio, MB Picture, Portal, Youtube, HBBTV, PVR, Source Switch (HDMI-SCART), Teletext and Channel List.

We used two of the test case generation algorithms that are provided by the MaTeLo tool. We employed the so called Minimum Arc Coverage algorithm [18] with default parameters before model refinements (*Iteration 0*). This algorithm terminates when all the transitions in the model are visited at least once. After each model refinement step, we employed the so called User Oriented algorithm [18]. This algorithm has two parameters: i) the maximum number of test steps within a test case, and ii) the maximum number of test cases. We set the maximum number of test steps to 5000. We set the maximum number of test cases to 1. As a result, this algorithm starts from the start state and traverses the model by choosing a transition at each state non-deterministically based on transition probabilities. The algorithm terminates when the finish state is reached.

In total 847 test steps were generated from the initial model before refinements. Execution of these test steps took 4 hours. 7 faults were detected during test execution. 5 of these faults were previously found during manual test activities. Therefore, 2 new faults were identified with MBT. In the following subsections, we describe the application of the 3 iterations of model updates.

A. Model Refinement based on Usage Profile

The first model update is performed based on usage profile. To collect this profile, 30 products were sent to 30 field testers. As soon as they connect their TVs to the Internet, log files are created and sent to record which modules are visited by users. After 30 days, we collected the log files and analyzed them. Then we calculated the probability values for each module as listed in the first part of Table I. After updating the transition probabilities, 809 test steps were generated. Execution of these test steps took 4 hours. 9 faults were detected during test execution. 7 of them were previously found. Therefore, 2 new faults were identified after the first iteration.

B. Model Refinement based on Static Analysis

In the second iteration, we utilized static code analysis alerts. The second part of Table I lists the number of alerts reported for each module and the calculated probability values accordingly. We performed a model update as depicted in Figure 4. 136 test steps were generated based on this model. Execution of these test steps took 1.5 hours. 3 faults were detected during test execution. 2 of them were previously found. Therefore, one new fault was identified after the second iteration.

[3]http://www.vestel.com.tr

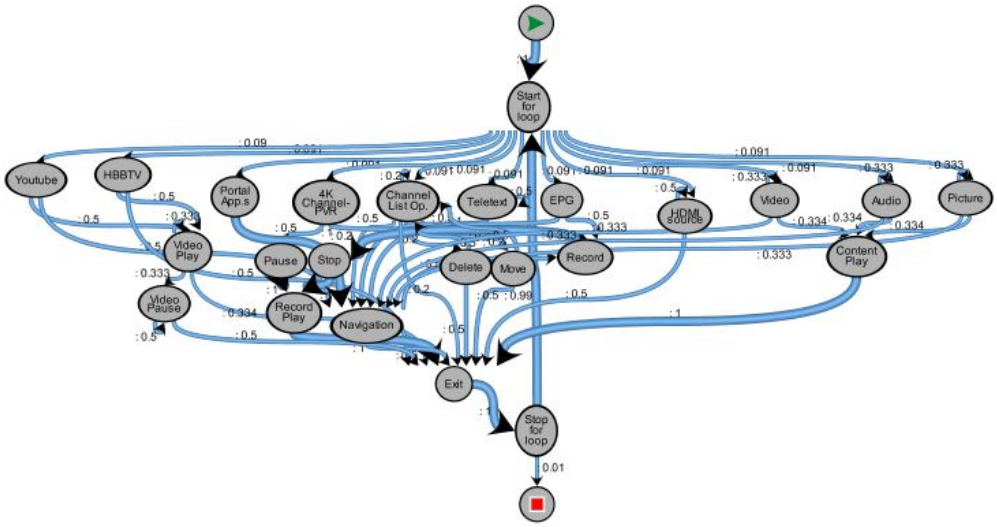

Fig. 3: An initial test model used for MBT with default probabilities.

Software Module	Iteration 1		Iteration 2		Iteration 3	
	# of Visits	Calculated Prob.	# of Warnings	Calculated Prob.	Memory leak (MB)	Calculated Prob.
Portal	1900	0.146	18	0.322	40.855	0.218
Youtube	2250	0.173	18	0.322	89.380	0.477
HBBTV	500	0.038	6	0.108	8.846	0.047
MBR Video	1750	0.134	2	0.036	22.375	0.119
MBR Audio	400	0.03	1	0.017	4.167	0.022
MBR Picture	100	0.007	1	0.017	3.980	0.021
PVR	1000	0.076	3	0.054	9.351	0.05
Channel List	1750	0.134	3	0.054	2.516	0.013
EPG	2000	0.153	2	0.036	3.094	0.017
Teletext	1250	0.096	1	0.017	1.675	0.009
HDMI-SCART	100	0.007	1	0.017	1.002	0.005

TABLE I: Collected data and the corresponding probability values used for updating the test model in successive iterations.

C. Model Refinement based on Dynamic Analysis

In the third iteration, we utilized memory profiles that are collected in the previous iterations. The third part of Table I lists the amount of memory leak associated with each module and the calculated probability values accordingly. The updated model can be seen in Figure 5. 117 test steps were generated based on this model. Execution of these test steps took 1.5 hours. 3 faults were detected during test execution. 1 of them were previously found. Therefore, 2 new faults were identified after the third iteration.

D. Results and Discussion

The overall results are summarized in Table II. We were able to detect 9 faults with the original model, out of which only 2 were actually detected with MBT for the first time. In successive iterations, we were able to detect 5 more new faults. These faults mainly lead to stability issues in the platform such as crashes and lack of response for remote controller commands.

In this study, we introduce two new *error states* to the test model throughout the refinement iterations. In principle, different types of *error states* can be added each of which represents a different type of error. The probability (risk) of causing each of these types of errors can be calculated for system states with static and/or dynamic analysis.

Another alternative approach would be to combine all the analysis results and perform an update once, instead of performing updates in several iterations. The advantage of this approach would be the decreased completion time of tests. However, as a drawback, one can be able to detect less number of faults. This is because, test steps generated in each iteration mainly focuses on different paths on the model although there can be occasional overlaps. Systematic avoidance of these overlaps can help to reduce overall test duration.

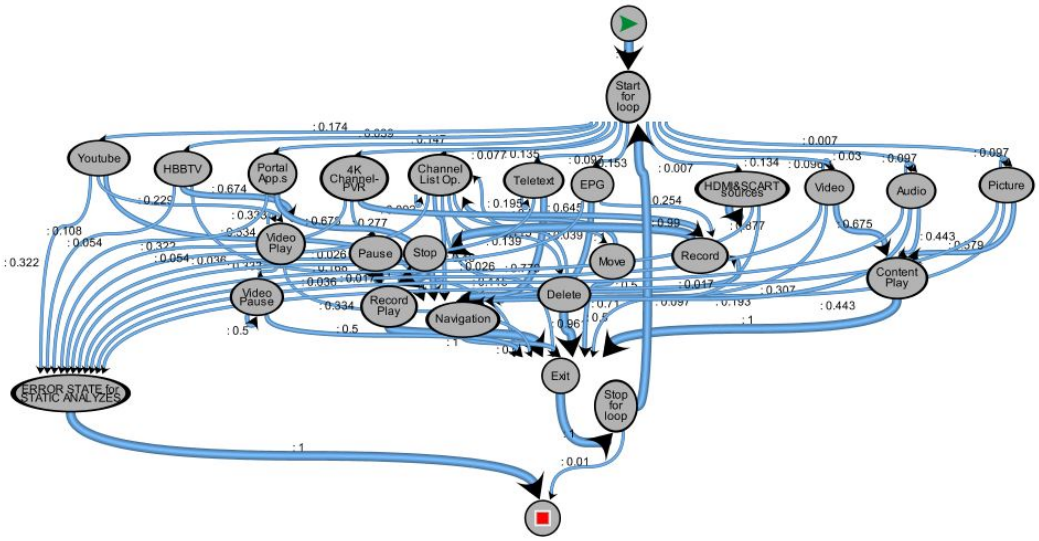

Fig. 4: The test model updated based on static analysis after the second iteration.

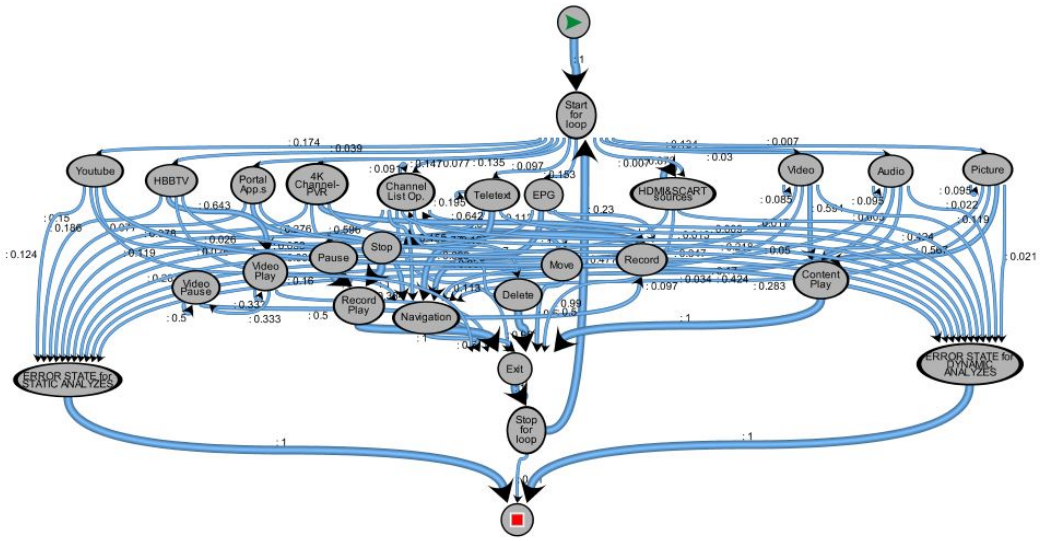

Fig. 5: The test model updated based on memory profile after the third iteration.

The total test execution time, including all the refinement steps took 11 hours. This duration can be reduced by skipping *Iteration 0*, which does not contribute much for detecting user-perceived failures. We can observe that the durations for the second and third iterations are reduced dramatically (more than 50%). The total time spent for successive test executions is 3 hours. Yet, 3 new faults were detected. Hereby, we should note that the number of test steps is not directly proportional to test duration. The execution of some of the test steps can not be fully automated and as such these steps take more time than others to execute. In these steps, a test engineer has to manually control a set of properties regarding audio and video. Therefore, elimination of such steps can further decrease the overall test duration.

V. Conclusions and Future Work

In this work, we introduced an iterative model refinement approach that utilizes usage profiles, static analysis and dynamic analysis. The overall goal is to cover different parts of a test model to effectively detect faults that are more likely to be exposed to the users. We applied our approach in the context of an industrial case study for MBT of a Smart TV system. We were able to detect new faults in each iteration, which shows that successive refinements of test models from different perspectives is a viable approach for increasing MBT effectiveness. In our study, probability values were manually adjusted. In our future work, we plan to automate the whole process and perform more case studies.

Iteration #	# of Test Steps	Test Execution Time (hr)	# of Faults Detected	# of New Faults Detected
0	847	4	7	2
1	809	4	9	2
2	136	1.5	3	1
3	117	1.5	3	2

TABLE II: Test results after each iteration

ACKNOWLEDGMENT

This work is supported by the joint grant of Vestel Electronics and the Turkish Ministry of Science, Industry and Technology (909.STZ.2015). The contents of this article reflect the ideas and positions of the authors and do not necessarily reflect the ideas or positions of Vestel Electronics and the Turkish Ministry of Science, Industry and Technology. We would like to thank software developers and software test engineers at Vestel Electronics for sharing their code base with us and supporting our case studies.

REFERENCES

[1] L. Apfelbaum and J. Doyle, "Model-based testing," in *Software Quality Week Conference*, 1997, pp. 296–300.

[2] J. Boberg, "Early fault detection with model-based testing," in *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, 2008, pp. 9–20.

[3] A. C. D. Neto, R.Subramanyan, M.Vieira, and G. H. Travassos, "A survey on model-based testing approaches: A systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, 2007, pp. 31–36.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proceedings of the International Conference on Software Engineering*, 1999, pp. 285–294.

[5] H. Robinson, "Finite state model-based testing on a shoestring," in *Proceedings of the Software Testing and Analysis and Review West Conference*, 1999.

[6] A. Chander, D. Dhurjati, S. Koushik, and Y. Dachuan, "Optimal test input sequence generation for finite state models and pushdown systems," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 140–149.

[7] J. Whittaker and M. Thomason, "A markov chain model for statistical software testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[8] J. Keranen and T. Raty, "Model-based testing of embedded systems in hardware in the loop environment," *IET Software*, vol. 6, no. 4, pp. 364–376, 2011.

[9] F. Bohr, "Model based statistical testing and durations," in *Proceedings of the 17th IEEE International Conference on Engineering of Computer Based Systems*, 2010, pp. 344–351.

[10] F. Bohr, "Model based statistical testing of embedded systems," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 18–25.

[11] X. Yuan and A. Memon, "Generating event sequence-based test cases using GUI runtime state feedback," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 81–95, 2010.

[12] B. Nguyen and A. Memon, "An observe-model-exercise* paradigm to test event-driven systems with undetermined input spaces," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 216–234, 2014.

[13] C. Gebizli and H. Sozer, "Improving models for model-based testing based on exploratory testing," in *Proceedings of the 8th IEEE International Computer Software and Applications Conference Workshops*, 2014, pp. 656–661.

[14] C. S. Gebizli, D. Metin, and H. Sözer, "Combining model-based and risk-based testing for effective test case generation," in *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation*, 2015, pp. 1–4.

[15] M. Felderer and I. Schieferdecker, "A taxonomy of risk-based testing," *International Journal of Software Tools and Technology Transfer*, vol. 16, no. 5, pp. 559–568, 2014.

[16] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[17] I. de Visser, "Analyzing user perceived failure severity in consumer electronics products incorporating the user perspective into the development process," Ph.D. dissertation, Eindhoven University of Technology, The Netherlands, 2008.

[18] C. Joye, "Matelo test case generation algorithms: Explanation on available algorithms for test case generation," 2014, http://www.all4tec.net/MaTeLo-How-To/understanding-of-test-cases-generation-algorithms.html.