Models in graphical user interface testing : study design

Models Used in Graphical User Interface Testing: Study Design

Grafiksel Kullanıcı Arayüzü Testinde Kullanılan Modeller: Çalışma Tasarımı

Alper Silistre
International Computer Institute, Ege University.
Izmir, Turkey.
alpersilistre@gmail.com

Onur Kilincceker
University of Paderborn, Paderborn, Germany.
Mugla Sitki Kocman University
Mugla, Turkey.
okilinc@mail.upb.de

Fevzi Belli
University of Paderborn, Paderborn, Germany.
Izmir Institute of Technology
Izmir, Turkey.
belli@upb.de

Moharram Challenger
University of Antwerp and Flanders Make,
Belgium.
moharram.challenger@uantwerpen.be

Geylani Kardas
International Computer Institute, Ege University.
Izmir, Turkey.
geylani.kardas@ege.edu.tr

***Abstract*—Model-based GUI testing is an important concept in Software GUI testing. Manual testing is a time-consuming labor and heavily error-prone. It has several well-accepted models that Software Testing community has been working and contributing to them for many years. This paper summarizes, reviews different models used in model-based GUI testing, presents a case study with a proposed approach for how to convert several well-accepted models to ESG (Event Sequence Graphs) to generate test cases and execute them with an aim to consolidate past and future works in a single model.**

Keywords—GUI Testing, Model-Based Testing, Finite State Machine, Event Sequence Graph, Event Flow Graph, Regular Expression

***Özet*—Model tabanlı grafiksel kullanıcı arayüzü testi (GUI), yazılım GUI testinde önemli bir kavramdır. Elle yapılan test zaman alıcı bir iştir ve büyük ölçüde hataya açıktır. Yazılım testi alanında çalışan insanların uzun yıllardır üzerinde çalıştığı ve katkıda bulunduğu belli modeller vardır. Bu bildiride, model tabanlı yazılım testinde kullanılan bu modelleri incelemenmiş ve belirlenmiş bazı modellerin ana model olarak kabul ettiğimiz Olay Sıra Çizgisi (ESG) modeline dönüştürülmesi, bunlardan test dizileri üretip model tabanlı test senaryoları oluşturulması ve çalıştırılması üzerine bir vaka çalışması sunulmuştur. Bu çalışma yapılırken, geçmişte ve gelecekte yapılacak olan çalışmaların birleştirilmesi amacı da güdülmektedir.**

Anahtar Kelimeler—Grafiksel Kullanıcı Arayüzü Testi, Model Tabanlı Test, Sonlu Durum Makinesi, Olay Sıra Çizgisi, Olay Akış Çizgisi, Düzenli ifade

I. INTRODUCTION

GUI (Graphical User Interface) is an essential part of all computer programs whether it is a web page, mobile, or desktop application. We interact with all kinds of GUIs to navigate or make programs to do their job. It's essentially an interface for us to communicate and interact with software programs. A user can execute an action by clicking a button or typing a text to an input field to interact with the application. GUI Testing is a process of validating GUI widget behavior and state based on preconditions that are decided by GUI testers. In the current software ecosystem, validating the business logic behind GUIs are often considered more important. The importance of GUI testing is often neglected by application developers due to a large number of possible test cases that need to be tested even if the application has a small number of widgets. Same action might put the program in an error state depending on the state of the program. This is hard to test manually and it leads applications to go into production with bugs in it. Thus, testing and validating the GUI of an application properly can reveal errors and defects, which is as important as testing underlying business logic. Additionally, the usability of an application is important in the modern software world, especially for consumer programs such as mobile applications in smartphones.

Model-based testing is a popular method of black-box testing of software. Creating the model of the system in a higher abstraction layer leads us to formalize test cases based on this model. In literature, there are different models such as FSM [2], EFG [6], ESG [5][12], RE [27][28]. Model-based testing allows us to generate test cases based on the model(abstraction) of the SUT (System under test) and then execute these tests on the model based on a defined oracle. There are automated tools and processes studied and evolved around this topic. Using model-based methods instead of code-based allows us to generate and execute test sequences more efficiently than to execute these tests with code.

In this study, we aimed to propose an approach for converting other models to the ESG model to use it in the complete test generation process and applying model-based test execution using this unified model. The ESG model has several advantages over other models that we want to take advantage from such as simplicity, generality, and scalability. Test sequences are generated automatically and execute them on the ESG model to unify the model-based test generation and execution processes. The main reason to convert existing models to the ESG is that different models need different processes and implementations to apply end-to-end model-based testing. With our study, we want to consolidate these efforts into a single unique model that is efficient to generate and execute test sequences.

The work in this paper is a design study. We review the literature in the related work section and present a proposed

approach that we plan to implement. Based on our experience and deductions from the literature review, we decide to use the ESG model for test generation and execution steps. We plan to explain its advantages compared to other models in detail. In the discussion section, we provide what we expect as the results of the study. In the future, we have plans to extend this study with larger models to find potential improvements to make the study more robust.

The remaining of the paper is organized as follows: Section 2 gives the related work within the scope of the proposed approach that is presented in section 3. Section 4 discusses expected results and implications with possible threats to the validity of the current work. Finally, Section 5 concludes the paper.

II. Related Work

This section introduces related works with respect to already existing models in GUI testing.

Memon et al. [14] focus on coverage criteria of GUIs and define the GUI component term to structure GUI into a hierarchy in order to identify important test sequences to be tested. They represent the GUI component by using the Event-Flow graph which identifies the interaction between GUI widgets in a GUI component.

Memon [15] describes why traditional software techniques and tools to test software applications are not the best fit for GUI testing because GUIs are different from application codes in terms of abstraction levels. He describes the process of GUI testing and how GUI testers should approach the process of GUI testing. Even though the given examples in the article reflect its time of writing which is 2002, pitfalls and process are still applicable today.

Belli [5] proposes a new approach as he called the 'holistic' approach. In this approach, testing GUI not only with correct test cases but also with incorrect test cases to show that the application should work as expected and cover cases even when the input and events are illegal. With this, we have a complete system coverage in terms of application behavior.

Shehady and Siewiorek [2], present a new formal model called VFSM (Variable Finite State Machine) for the GUI with a smaller number of states than an FSM while keeping the system design equivalent. VFSM can be converted to an equivalent FSM anytime in order to create test cases. Since the total number of states is less, modeling a system with VFSM is easier and in less time than FSM.

White and Almezen [4] utilize a concept they called responsibility, an activity that involves one or more GUI objects which results in an observable effect on the system. For this defined responsibility, they created a term called CIS (Complete Interaction Sequences), which is a combination of all actions and GUI objects that can invoke the defined responsibility.

Memon et al. [6] present a new technique, an AI-based planning algorithm for automated test generation from the EFG model. Based on the defined operators, the initial and final steps are created to apply the planning algorithm on the EFG model. The algorithm creates test sequences between the initial and final states, taking into account GUI events and interactions.

Memon [7] presents a new method for model-based testing by using event-space exploration strategies. He combines all models for model-based testing into one scalable model, called the event-flow model. He automates the procedure to reduce the cost and effort of the model creation steps.

Xie and Memon [8] define a new concept by using their previous works on EIG and EFG called Minimal Effective Event Context (MEEC) which are the shortest event sequences needed to show the error on the GUI model. Because you can naturally traverse through a combination of events to detect the fault (since the response to an event may defer based on the current state of a system) in a GUI system which would make the test sequence unnecessarily long. Instead, MEEC shows the shortest path to detect the fault.

Huang et al. [9] develop a method to repair GUI test cases that are useless for the GUI testing because of the possibilities like premature termination of the test. They use a genetic algorithm to correct these problematic test series and increase coverage.

Belli et al. [10] present a case study on the reliability of GUIs and the selection of a GUI's reliability model in human-machine systems to gain experimental insights about them. They state that choosing an appropriate modeling technique for the GUI test affects the quality of the evaluation process and therefore the software.

Banerjee et al. [11] survey about GUI testing studies and matched the related papers with a systematic mapping technique. They identify selection criteria for studies from the pool of 230 articles written between 1991 and 2011 about the GUI testing. They categorize the studies and provide an overview of current approaches and areas which require further study and research. They provide examples from traditional and modern techniques for model-based GUI testing.

Belli et al. [12] review existing work on model-based GUI testing in detail by considering modeling and test case generation techniques. They examine the optimization of these techniques while giving real-world examples of these models and their usages.

Belli et al. [13] perform a study that reduces the number and cost of test cases by recommending the layered-centric test method and the associated test creation system in case the system under test grows too large. Using this methodology, they demonstrate that many faulty states can be found even with a small number of test cases.

Kilincceker et al. [24] introduce regular expression for modeling and testing GUI. They also generate random test sequences from regular expression and evaluate their random test generation algorithm on a case study.

Mercan et al. [25] present finite state maching for modeling and testing GUI of mobile application. They also propose a methodology for testing presence and absence of faults with respect to the finite state machine model.

Kilincceker and Belli [26] propose a novel coverage criteria for GUI testing by means of an analysis based on the regular expression. After analysis of regular expressions, they obtain contextual tables from which they present coverage criteria. These coverage criteria are used in [28] for test generation and testing including quality evaluation based on mutation testing.

Kilincceker and Belli [30] present a unified modeling method for both hardware design and software GUI testing. They also utilize holistic testing approach combined with mutation testing. They evaluate their modeling and testing method in two cases studies taken from hardware design and software GUI domain.

The current study uses ESG as a unified model based on comparison with other models in terms of their effectiveness.

III. Proposed Approach

The proposed approach provides a way to create an ESG presentation in any of the open-standard file formats such as JSON or XML to represent the model of the system. The ESG model should be able to be converted from a Finite State Machine (FSM), Hierarchical Finite State Machine (HFSM), Regular Expression (RE), and Event Flow Graph (EFG). These are all existing models in the literature that allow us to model a GUI system. For example, this can be a sign-up form on a website or a screen that accepts user interactions in a mobile application.

A. Used Notations

Used notions are defined formally in this section. These notions are Finite State Machine (FSM), Hierarchical Finite State Machine (HFSM), Regular Expression (RE), and Event Flow Graph (EFG).

For each formal notation, we will demonstrate an example GUI system including corresponding models. The example system is a simplified version of the ISELTA [23] website's Special module:

Fig. 1: ISELTA [23] website's Special module

I. Finite State Machine (FSM)

Definition 1: Following 5-tuple defines an FSM [22] <Q, Σ, δ, q0, F> with

- Q: a finite set of states
- Σ: a finite set of input symbols (alphabet)
- δ: a state transition function
- q0: an initial (starting) state belongs to Q
- F: a finite set of final states belongs to Q

Example 1: Following 5-tuple defines the FSM of ISELTA [23] website's Special module (see Fig. 2).

- Q: = {q0, q1, q2, q3, q4, q5, q6, q7, q8}
- Σ: = {t, a, p, d, s}
- δ: = {δ(q0, t)=q2, δ(q2, a)=q7, δ(q2, p)=q3, δ(q2, d)=q8, δ(q7, p)=q4, δ(q3, a)=q4, δ(q3, d)=q5, δ(q8, p)=q5, δ(q4, d)=q6, δ(q5, a)=q6, δ(q6, s)=q1}
- q0: = {q0}
- F: ={q1}

Where "t", "d", "p", "a", and "s" represent set title, set departure, set price, set arrival add button events respectively.

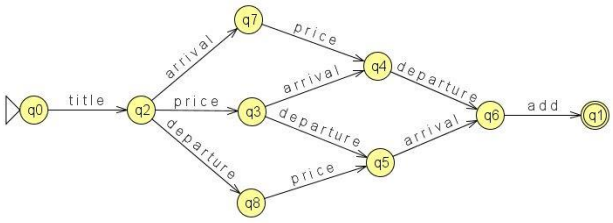

Fig. 2: ISELTA Special Module FSM

II. Hierarchical Finite State Machine (HFSM)

Definition 2: Following 6-tuple defines an HFSM [6] <Q, Σ, δ, q0, F> with

- Q: a finite set of states
- Σ: a finite set of input symbols (alphabet)
- δ: a state transition function
- q0: an initial (starting) state belongs to Q
- F: a finite set of final states belongs to Q
- L: a finite set of layers

Example 2: Following 6-tuple defines the HFSM of ISELTA [23] website's Special module.

- Q: = {q0, q1, q2, q3, q4, q5, q6, q7, q8}
- Σ: = {t, a, p, d, s}
- δ: = {δ(q0, t)=q2, δ(q2, a)=q7, δ(q2, p)=q3, δ(q2, d)=q8, δ(q7, p)=q4, δ(q3, a)=q4, δ(q3, d)=q5, δ(q8, p)=q5, δ(q4, d)=q6, δ(q5, a)=q6, δ(q6, s)=q1}
- q0: = {q0}
- F: = {q1}
- L: = {Ø}

III. Event Sequence Graph (ESG)

Definition 3: Following 4-tuple defines an ESG [12] <E, A, S, F> with

- E: a finite set of nodes representing events
- A: A ⊆ N×N a finite set of directed arcs representing follows relation between events
- S: non-empty set of events representing start event
- F: non-empty set of events representing final event

Example 3: Following 4-tuple defines an ESG [12] of ISELTA [23] website's Special module (see Fig. 3).

- E: = {[, set title, set arrival, set price, set departure, click add,]}
- A: = {([, set title), (set title, set arrival), (set title, set price), (set title, set departure), (set arrival, set price), (set arrival, set departure), (set arrival, click add), (set price, set arrival), (set price, set departure), (set price,

click add), (set departure, set arrival), (set departure, set price), (set departure, click add), (click add,])}

- S: = {[}
- F: = {]}

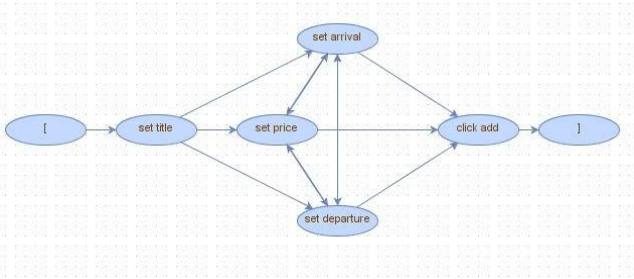

Fig. 3: ISELTA Special Module ESG

IV. Event Flow Graph (EFG)

Definition 4: Following 4-tuple defines an EFG [14] <V, E, B, I> with

- V: a set of vertices that represent all events
- E: a set of directed edges between vertices
- B: a set of vertices that are available at the start of a modeled GUI
- I: a set of restricted-events (events that are not possible to occur) for the GUI component

Example 4: Following 4-tuple defines an EFG [14] of ISELTA [23] website's Special module (see Fig. 4).

- V: = {set title, set arrival, set price, set departure, click add}
- E: = {([set title, set arrival), (set title, set price), (set title, set departure), (set arrival, set price), (set arrival, set departure), (set arrival, click add), (set price, set arrival), (set price, set departure), (set price, click add), (set departure, set arrival), (set departure, set price), (set departure, click add)}
- B: = {set title, set arrival, set price, set departure}
- I: = {Ø}

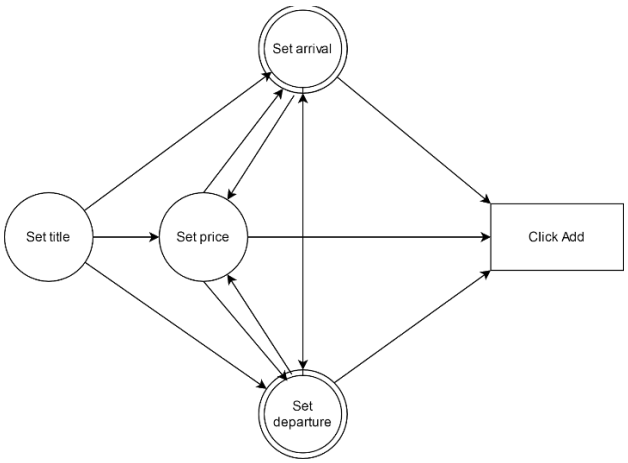

Fig. 4: ISELTA Special Module EFG

V. Regular Expression (RE)

Definition 5: Regular Expression (RE) [22]: A RE by means of rules is defined by the sequence of symbols x, y, z, ... Symbols can occur zero or more times related to the following rules which define the RE.

- Concatenation: represented by '.' or ''(blank). For example, 'ab' refers to 'a' is followed by 'b'
- Selection: represented by '+'. For example, x + y refers to 'x (exclusive) or y'.
- Iteration: represented by '*'. For example, 'x*' refers to 'x is iterated a desired time'.

Example 5: Following RE defines ISELTA [23] website's Special module.

R: = (tdpas+tapds+(tpda+tpad)s)

B. Approach

The proposed approach is divided into three steps (see Fig. 5), which are test preparation, test generation, and test execution. We aim to automate these three steps to derive a complete model-based test automation tool.

In test preparation, the FSM, HFSM, RE, and EFG models are converted into the ESG model due to simplicity, generality, and scalability advantages. Users are also able to directly import ESG into the system. In test generation, a test suite containing a valid set of test sequences is generated from the ESG model utilizing a graph traversal algorithm. Also, the mutants of ESG models are obtained from original ESG on which to apply insertion, replace, and omission mutation operators. Finally, the test suite is executed on all mutant models to calculate the mutation score for measuring the quality of the test suite.

During our study whose design is described above, we will briefly compare these models and present their advantages and disadvantages. After creating test sequences from the ESG model, we will demonstrate how to apply model-based test execution into the model to test the system. We will use ISELTA [23] website's forms inside the case study of our approach to show the results.

IV. Discussion

A. Expected results and implications

We aim to provide an end to end model-based test generation and execution approach of the example system (ISELTA website form) that we present in this paper. A unified model (ESG) that is generated from other models or provided directly will be the main input of this approach. Algorithms to make conversions possible will be presented in the study. With this, a unified test generation approach will be studied by using the unified ESG model. The quality of the test suite that is generated from the test generation approach will be evaluated by utilization of mutation testing. We expect these efforts will make model-based testing more available to a broader mass of people who are working in the GUI testing field of Software Engineering. Depending on the quality of the approach, the existing models of the systems can use this approach by converting their model representations to the ESG and hence the applying the end to end test generation and execution approach would be the best outcome of our study. Another possible expected result would be to show the software testing community about the ease of usage of model-based testing and its evaluation.

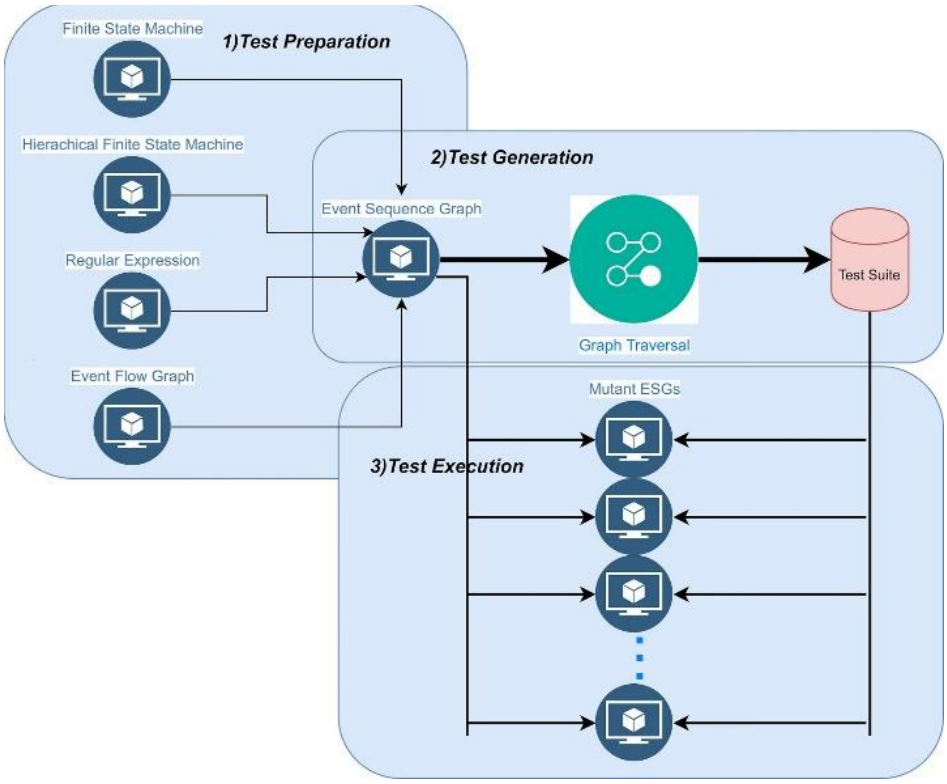

Fig. 5: The overview of the proposed approach

B. *Threats to validity*

I. *Conclusion Validity*

The sample size of our case study is a potential threat to generalize the methodology. We need large size of case studies for each model (FSM, HFSM, EFG, ESG, RE) to validate the approach and find any potential problem that we might miss due to the small size of our sample. We plan to extend our work with bigger test cases to cope with this potential problem. This will let us find issues when the sample size similar in size to real-life systems.

II. *Internal Validity*

The nature of model-based testing is a threat to internal validity because the entire approach runs on models rather than the actual GUI program. It's not possible to fully cover and test a system with model-based testing as if running tests in the software code with a white box testing approach. A model is just a representation and description of the actual software behavior. Depending on the complexity of the software under test, creating a correct model to represent the system properly might be hard. Hence, the correctness of the initial model of the system is important and it is a threat to our approach. If a model represents the system in the wrong way, all conversions and test generation/execution approach will not cover the system as it should be. We will create our models for an example system to make sure the proposed approach can be applied in appropriate models that fully represent the system.

III. *External Validity*

Applying the approach outside of context is a threat to external validity. Current work aims to detect functional and operational faults rather than other types of faults such as visual attributes and their semantics as mostly used in GUI of games. This is related to what model-based testing is created for. Since models functionally represent systems, wanting to test a system's visual elements on the screen might not be suitable for the approach. Normally, a white box code-based testing approach might be more suitable for these kinds of validations. However, when sequential and behavioral models used in a testing method rather than Petri-Nets modelling, this proposed approach is practicable.

IV. *Construct Validity*

Conversion of the models to the ESG model may result in a threat to construct validity due to their different expressive powers. After conversion, we increase or decrease their expressiveness to the unified model's expressive power that might result in missing functionalities. This needed to be tested with bigger sample size and with different cases to understand in what level this expressive power might be lost during the conversion. As we mentioned before, we plan to extend our work with bigger sample sizes to cope with this issue to prevent the loss of expressive power beyond a reasonable point.

V. CONCLUSION

We aim to analyze well-known models in GUI testing in the current work whose design is given in this paper. The initial analysis shows that the utilization of different models requires distinct abilities and results in different syntax and semantics. These differences affect abilities of the models with respect to representing systems and further processes such as test generation and test execution. Based on our

experiences, ESG stands out as the most suitable of these models in terms of both test generation and test execution. This is the main reason for choosing ESG in the proposed approach.

In the proposed approach, these models will be converted automatically to unified ESG in the test preparation step and then test sequences will be generated from ESG in the test generation step. Finally, the generated test sequences will be executed on mutants of the ESG model to evaluate quality of test sequences. With this, we will have the ability to use its advantages for our model-based test generation and implementation process, which potentially will have capabilities to be upgraded as needed in the future work. Studying test generation and execution on different implementations hold us to improve our processes with future studies because the model-based testing is segmented with different efforts on these different models. A system can be modelled directly with ESG or can be converted from other models to broaden our reach in the model-based testing area. One of our goals for the future is that existing systems that are modelled with present models (FSM, HFSM, EFG, etc.) can benefit from our study and test their system with our model-based test generation and execution process.

REFERENCES

[1] Banerjee, Ishan, Bao Nguyen, Vahid Garousi, and Atif Memon. "Graphical user interface (GUI) testing: Systematic mapping and repository." Information and Software Technology 55, no. 10 (2013): 1679-1694.

[2] R.K. Shehady, D.P. Siewiorek, A method to automate user interface testing using variable finite state machines, in: Proceedings of the 27th Annual International Symposium on Fault-Tolerant Computing (FTCS 1997), IEEE Computer Society, Washington, DC, 24–27 June, 1997, pp. 80–88.

[3] Chow, T. S. (1978). Testing software design modeled by finite-state machines. IEEE transactions on software engineering, (3), 178-187.

[4] L. White, H. Almezen, Generating test cases for GUI responsibilities using complete interaction sequences, in: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE 2000), IEEE Computer Society, Washington, DC, 2000, pp. 110–121.

[5] Belli, F., Finite state testing and analysis of graphical user interfaces. Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, (2001).

[6] A.M. Memon, M.E. Pollack, M.L. Soffa, Hierarchical GUI test case generation using automated planning, IEEE Trans. Software Eng. 27 (2) (2001) 144–155.

[7] Memon, Atif M. "An event-flow model of GUI-based applications for testing." Software testing, verification and reliability 17.3 (2007): 137-157.

[8] Q. Xie, A.M. Memon, Using a pilot study to derive a GUI model for automated testing, ACM Trans. Software Eng. Methodol. 18 (2) (2008), 1–35.

[9] S. Huang, M.B. Cohen, A.M. Memon, Repairing GUI test suites using a genetic algorithm, in: Proceedings of the 2010 3rd International Conference on Software Testing, Verification and Validation (ICST 2010), IEEE Computer Society, Washington, DC, 6–10 April, 2010, pp. 245–254.

[10] Belli, F., Beyazit, M., Güler, N., Event-Oriented, Model-Based GUI Testing and Reliability Assessment—Approach and Case Study. Advances in Computers, 85, 277-326, (2012).

[11] Banerjee, Ishan, Bao Nguyen, Vahid Garousi, and Atif Memon. "Graphical user interface (GUI) testing: Systematic mapping and repository." Information and Software Technology 55, no. 10 (2013): 1679-1694.

[12] Belli, Fevzi, Mutlu Beyazıt, Christof J. Budnik, and Tugkan Tuglular. "Advances in model-based testing of graphical user interfaces." In Advances in Computers, vol. 107, pp. 219-280. Elsevier, 2017.

[13] Belli, Fevzi, Nevin Güler, and Michael Linschulte. “Layer-centric testing.” FERS-Mitteilungen: Vol. 30, No. 1 (2012)

[14] Memon, Atif M., Mary Lou Soffa, and Martha E. Pollack. "Coverage criteria for GUI testing." Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering. 2001.

[15] Memon, Atif M. "GUI testing: Pitfalls and process." Computer 8 (2002): 87-88.

[16] Chow, Tsun S. "Testing software design modeled by finite-state machines." IEEE transactions on software engineering 3 (1978): 178-187.

[17] Lee, David, and Mihalis Yannakakis. "Principles and methods of testing finite state machines-a survey." Proceedings of the IEEE 84.8 (1996): 1090-1123.

[18] Fujiwara, S., Bochmann, G. V., Khendek, F., Amalou, M., & Ghedamsi, A. (1991). Test selection based on finite state models. IEEE Transactions on software engineering, (6), 591-603.

[19] Utting, Mark, Alexander Pretschner, and Bruno Legeard. "A taxonomy of model-based testing approaches." Software testing, verification and reliability 22.5 (2012): 297-312.

[20] Belli, Fevzi. "Finite state testing and analysis of graphical user interfaces." Proceedings 12th international symposium on software reliability engineering. IEEE, 2001.

[21] Belli, Fevzi, Mutlu Beyazit, and Atif Memon. "Testing is an event-centric activity." 2012 IEEE Sixth International Conference on Software Security and Reliability Companion. IEEE, 2012.

[22] Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. Automata theory, languages, and computation." International Edition 24.2.2 (2006).

[23] ISELTA website, Available online: http://iselta.ivknet.de

[24] Kilincceker, O., Silistre, A., Challenger, M., & Belli, F. (2019, July). Random test generation from regular expressions for graphical user interface (GUI) testing. In 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 170-176). IEEE.

[25] Mercan, G., Akgündüz, E., Kılınççeker, O., Challenger, M., & Belli, F. (2018). Android uygulaması testi için ideal test ön çalışması. CEUR Workshop Proceedings.

[26] Kilincceker, O., & Belli, F. (2017). Grafiksel kullanıcı arayüzleri için düzenli ifade bazlı test kapsama kriterleri. CEUR Workshop Proceedings.

[27] Kilinccceker, O., Turk, E., Challenger, M., & Belli, F. (2018, July). Regular expression based test sequence generation for HDL program validation. In 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 585-592). IEEE.

[28] Kilincceker, O., Turk, E., Challenger, M., & Belli, F. (2018, April). Applying the Ideal Testing Framework to HDL Programs. In ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems (pp. 1-6). VDE.

[29] Kilincceker, O., & Belli, F. (2019, November). Towards Uniform Modeling and Holistic Testing of Hardware and Software. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)* (pp. 1-6). IEEE.

[30] Kilincceker, O., & Belli, F. (2019, November). Towards Uniform Modeling and Holistic Testing of Hardware and Software. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)* (pp. 1-6). IEEE.