Testing the Usability and Accessibility of Smart TV Applications Using an Automated Model-based Approach

Miroslav Bures, Miroslav Macik, Bestoun S. Ahmed, Vaclav Rechtberger, and Pavel Slavik

***Abstract*—As the popularity of Smart Televisions (TVs) and interactive Smart TV applications (apps) has recently grown, the usability of these apps has become an important quality characteristic. Previous studies examined Smart TV apps from a usability perspective. However, these methods are mainly manual, and the potential of automated model-based testing methods for usability testing purposes has not yet been fully explored. In this paper, we propose an approach to test the usability of Smart TV apps based on the automated generation of a Smart TV user interaction model from an existing app by a specialized automated crawler. By means of this model, defined user tasks in the Smart TV app can be evaluated automatically in terms of their feasibility and estimated user effort, which reflects the usability of the analyzed app. This analysis can be applied in the context of regular users and users with various specific needs. The findings from this model-based automated analysis approach can be used to optimize the user interface of a Smart TV app to increase its usability, accessibility, and quality.**

***Index Terms*—Usability Testing, Model-based Testing, User Interface Quality, Smart TV application.**

I. Introduction

Currently, Smart TVs are coming to dominate the television market, and the number of connected TVs is growing exponentially. This growth is accompanied by an increase in consumers and the use of Smart TV apps that drive these devices. Smart TV apps fully interact with the user via a visualized UI and a remote device. Due to the increasing demand for Smart TV apps, especially with the rise of the Internet of Things (IoT), developing new usability testing methods for these apps is essential. The classic User Interface (UI) evaluation approaches for usability testing are based on mainly manually performed testing with respect to the UI of the System Under Test (SUT) [1]. The potential of automated generation of a UI model from an existing Smart TV app combined with model-checking principles has not been fully explored.

To this end, the motivation of this study is threefold. First, the combination of UI model generation from an existing Smart TV app with model-checking principles to detect possible UI design suboptimality is not sufficiently covered in the literature. Second, concerns related to the usability of Smart TV apps were raised by Ingrosso et al. [2] and Alam et al. [3] in 2015 and 2017, respectively. In fact, Alam et al. [3] discussed a number of potential usability issues of Smart TV apps. Considering the growth of the Smart TV market and the increase in app users, the focus on the usability testing of these apps must be intensified to prevent the usability problems reported by previous studies [2], [3]. A systematic and efficient usability testing method for Smart TV apps should be provided. Third, the current UI testing studies focus on various devices and types of apps. The Smart TV app domain remains relatively underexplored by relevant studies.

Based on the motivations mentioned above, the objective of this paper is to propose and verify an automated model-based method to detect possible design flaws or suboptimalities in the UI of a Smart TV app. We propose a method based on the analysis of the UI model of a Smart TV app that is acquired automatically by a specialized crawler. Defined user tasks in the Smart TV app are mapped to this model and then evaluated by a set of rules to verify feasibility and effectiveness of these tasks in which the user interacts with the app's UI. The context of the user interacting with an app is reflected in these rules. This context is expressed by a set of configuration constants, i.e., user capability to perform individual actions in the UI, device factor, environmental factor, and a default user effort of the individual actions in the UI. In this context, we can model users with various specific needs. The verification rules assess the feasibility of the task in the app for the user in a particular context and estimate the length to detect potential suboptimalities in the UI design or to detect repetitive steps in the UI needed to achieve the task. The findings of this analysis can help UI designers and app developers to optimize their UI in consideration of both the specific features of the Smart TV app and the particular needs of a user. This method can also aid the evaluation of user feedback on the quality of the app's UI in an independent objective manner. The contributions of this paper can be summarized as follows:

- We present an approach that potentially synergizes usability testing and functional testing based on the underlying model-based testing principles.
- We propose an innovative method that enables analysis

This study is conducted as a part of the project TACR TH02010296 “Quality Assurance for Internet of Things Technology”. This work has been supported by the OP VVV funded project CZ.02.1.01/0.0./0.0./16_019/0000765 âĂđResearch Center for InformaticsâĂİJ. *(Corresponding author: B. Ahmed)*

M. Bures and V. Rechtberger are with the Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University, Karlovo nam. 13, Prague, Czech Republic, email: buresm3@fel.cvut.cz

M. Macik and P. Slavik are with the Department of Computer Graphics and Interaction, FEE, Czech Technical University in Prague, Karlovo nam. 13, Prague, Czech Republic, email: macikmir@fel.cvut.cz, slavik@fel.cvut.cz

B. Ahmed is with the Department of Mathematics and Computer Science, Karlstad University, Sweden and the Department of Computer Science, Czech Technical University, Karlovo nam. 13, Prague, Czech Republic, email: bestoun@kau.se

of the feasibility and ease of user tasks in a UI and assessment of the optimality based on a UI model that is generated automatically by a special crawler. Thus, an up-to-date and accurate design model of the UI from the design phase of the project is not needed.
- We propose a novel application of model-based UI analysis in the Smart TV domain, which has not been sufficiently explored.
- We report the parametrization of the user interaction model for Smart TV apps that is calibrated during several sets of experiments performed with real users.

II. Related Work

Smart TV represent prospective stream of consumer electronic development. Compared to traditional TV, besides the possibility to personalize their user environment [4], users appreciate variety of applications that can be installed in the smart TV set, spanning from various games, media and infotainment applications to various services, including employment of smart TV sets in various home IoT solutions. Especially this field is a subject of recent research and development, for instance controlling of smart home appliances [5], [6], smart light management system [7] or the whole smart home solution [8], [9] using a smart TV set, various personal healthcare application employing smart TV, for instance [10] or personal sleep management employing a video analysis using a smart TV application [11]. As another example, smart home security system using cameras and smart TV set can be given [12]. Integration of smart TV sets into various smart home systems and services as well as increasing popularity of smart TV among users also increase requirements on usability of their applications.

Regarding usability testing of smart TV applications, previous work related to manual usability testing and assessments can be found. To give few examples, Shin et al. [13] examined the users' attitude and perception of Smart TV devices from a usability perspective.

Ingrosso et al. [2] examined the usability of Smart TV apps using a case study of a T-commerce application.

A number of potential usability issues of Smart TV apps were discussed in a more recent analysis by Alam et al. [3]. These recent studies can also be seen as motivation to develop specific usability testing methods to improve the general usability of Smart TV apps.

Regarding the automation of usability tests, several previous projects can be identified. For example, automated testing of usability and accessibility of web pages has been proposed by Okada et al. [14]. Here, the proposed system collects logs from users' interaction with the SUT. The usability and accessibility were evaluated by comparing the logs with hypothetical ideal scenarios. Also, more formal approaches to usability testing have been examined in the literature to enable a more systematic approach to the design of the test automation system. Gimblett and Thimbleby [15] proposed a testing approach using a theorem discovery method to find and check usability heuristics automatically. Here, sequences of equivalent or very similar user inputs and their effect on the SUT were analyzed [15].

Cassino and Tucci [16] proposed an approach to evaluate the interactive visual environments, which is based on SR-Action Grammars [17]. This approach aids developers to create applications in which the UI respect defined usability rules. The practical implementation of the method resulted in the automatic usability verification tool [16]. The formal specification is created from the SUT and used for subsequent usability checks and as particular usability rules, set of Nielsen heuristics [18] were employed in the proposed tool.

However, during our analysis of the state of the art, we have found only a few studies related to the automated usability testing of Smart TV apps based on a model created by an automated scan of the app's UI. Previous effort regarding the modeling of the smart TV app has been done by Cui et al. [19]. Instead of a user interaction model with the app's UI as we propose, Cui et al. employed the hierarchical state transition matrix (HSTM), which is based on a state machine and hierarchical structure of the app.

Several crawlers creating a model for the UI have been presented in the literature for mobile and web apps. For instance, the projects by Mesbah et al. [20] for web applications, Memon et al. [21] for thick-client app UIs, Amalfitano et al. [22], [23], and Wang et al. [24] for mobile apps. Also, the universal frameworks allowing connection to a particular app's UI by a modular interface, as proposed by Nguyen et al. [25].

The concepts presented in this paper can also be conceptually compared to the model-checking approach. However, the applications of model-checking techniques usually focus on the detection of potential functional defects on various levels of the SUT in its classical form [26], or when model-checking is combined with dynamic testing [27]. As modeling structures, different formal notations and employed currently. These notations include finite state machines and their various extensions and modifications for the modeling of discrete systems [28], Petri nets or marked graphs for the modeling of concurrent processes, or hybrid automata or real-time temporal logics to model real-time systems [28].

Using the model-checking approach for UI usability testing is relatively under-explored in the literature. Harrison et al. [29] focused on this domain recently, using temporal logic as an underlying model of the SUT.

III. Overview of Our Proposed Approach

The proposed method is applicable mainly to Smart TV apps during the development and testing process. However, the method can be applied to apps in alpha and beta testing or even production run, when the users report UI suboptimalities during their interaction with the app. Different types of suboptimalities exist, such as (1) user discomfort, (2) confusing organization of the individual elements of the UI, (3) too long or confusing sequence of steps to be taken to achieve frequently performed tasks, and (4) suboptimality of the app's UI for users with specific needs of particular category, or any other UI design flaws.

These suboptimalities are detected by metrics based on the proposed user interaction model (defined in Section IV-A) and the execution time of user scenarios.

The following steps summarize the conceptual process of the proposed approach:

- The UI of the app is scanned by a special crawler (described in detail in Section V) that creates an extensive user interaction model of the Smart TV app (described in Section IV-A).
- The user (the UI designer or the developer) defines a set of test scenarios. The scenarios capture the most frequent user tasks to be performed in the app and/or the user tasks that are reported as problematic from a usability/accessibility viewpoint by users or usability testers of the app.
- Defined test scenarios are captured in the user interaction model using the specialized Model-based Testing (MBT) platform (details follow in Section VII-C).
- The context in which the defined test scenarios are assessed is defined using a set of configuration constants (discussed further in Section IV-C).
- A set of verifications is performed for each of the scenarios and defined context. These verifications include feasibility assessment of the scenario in the app's UI, user effort needed to execute the scenario and repetition of IU elements. The exact description of these verifications is presented in Section VI.
- During the removal of the UI design problems identified in the previous step, the UI designer edits the SUT user interaction model in the MBT platform (more possible transitions or shortcuts in the SUT UI can be added, for instance). After these corrections, scenarios that were evaluated as problematic during the previous step can be reanalyzed until satisfactory results are achieved.
- Finally, the adjustments in the user interaction model can be transformed into a set of change requests for the UI development team.

The used MBT system[1] is an experimental platform for process and path-based testing developed and issued by the Software Testing IntelLigent Lab (STILL), Dept. of Computer Science, FEE, Czech Technical University in Prague. The application supports creation of user models via a graphical UI and employs a set of algorithms to validate the created models and generate test cases from these models.

IV. User Interaction Model

Our proposed approach is based on the user interaction model (explained in Section IV-A) and its parametrization that reflects the context. The suggested values for the Smart TV domain are discussed in Section IV-C.

A. Model Definition

A user's interaction with the Smart TV app's UI is abstracted as the user interaction model. Here, we use a directed multigraph to describe the model as $G = (N, E, n_s, N_e, s, t)$, such that $N \neq \emptyset$ is a finite set of nodes, E is a set of edges, $s : E \rightarrow N$ assigns each edge to its source node and $t : E \rightarrow N$ assigns each edge to its target node. The node $n_s \in N$ is the initial/start node of the graph G, and $N_e = \{n_e \mid n_e \in N \text{ has no outgoing edge }\}$ defines nonempty set of end nodes of graph G. A node in the directed graph models a screen or a screen element of the UI. A screen element is a standalone clickable part of the screen layout or a nested container on the screen.

[1]http://still.felk.cvut.cz/oxygen/

A graph edge in the model represents a transition between nodes via the interactive (control) element. Each transition $e \in E$ can be triggered by an **input action** $a(e)$. An input action is a physical action of the user on the remote control device that leads to transition e in the app. Consider the remote control device as an example for a Smart TV app. Here, the input actions are events sent from the device to the Smart TV app when a user presses UP, DOWN, OK, or another button. An edge e can have identical source and target node, making a simple loop; this case models the situation where an input action $a(e)$ does not trigger a transition between nodes on the app's UI but changes an internal state of the app.

The Tested User Scenario t is an ordered sequence of nodes $N_t \subseteq N$ and edges $E_t \subseteq E$ which have to be visited during the execution of the user scenario. The $n_1 \in N_t$ is a starting node of t and $n_n \in N_t$ is a terminal node of t. A set T is a set of all Tested User Scenarios. The nodes and edges in t can repeat.

User scenario path $p(t)$ of the tested user scenario t is a path in G that contains the nodes N_t and the edges E_t of t and can also contain other nodes or edges of G. $p(t)$ starts with n_1 and ends with n_n. The order of N_t and E_t, as defined in t, is maintained in $p(t)$. Furthermore, $|p(t)|$ denotes the number of edges of $p(t)$, and $nodes(p(t))$ denotes the unique number of nodes of $p(t)$. Note that t itself is not necessarily a path of G. Additionally, as the nodes and edges in t can repeat, $p(t)$ is not necessarily the shortest path from n_s to a node from N_e.

C is the context in which the user accesses the app's UI. The user effort required to perform a transition $e \in E$ is

$$\mathcal{E}(e, C) = \delta(a(e)) \times \frac{1}{UC(a(e), C)} \times \frac{1}{\mathcal{E}_{dev}(C)} \times \frac{1}{\mathcal{E}_{env}(C)},$$

and the total user effort of user scenario path $p(t)$ is

$$\mathcal{E}(p(t), C) = \sum_{i=1}^{|p(t)|} \mathcal{E}(e_i, C), e_i \in p(t),$$

where $UC(a(e), C)$ is the user capability to perform an action $a(e)$ in context C (0 - user is unable to perform the action, 1 - user is able to perform the action with standard effort). $\mathcal{E}_{dev}(C)$ is the device factor, and $\mathcal{E}_{env}(C)$ is the environmental factor. $\delta(a(e))$ is the default effort of the particular action measured in milliseconds, including the time of cognitive effort to operate and the time to interact with the UI. The other constants, UC, $\mathcal{E}_{dev}$, and $\mathcal{E}_{env}$, are unitless.

The suggested values of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$, and δ are discussed further in Sections IV-C (initial values of the constants) and VII-D (refined values of the constants after the experiments). The total user effort is further used in the assessment of defined tested scenarios T in the UI modeled by G (details are provided in Section VI).

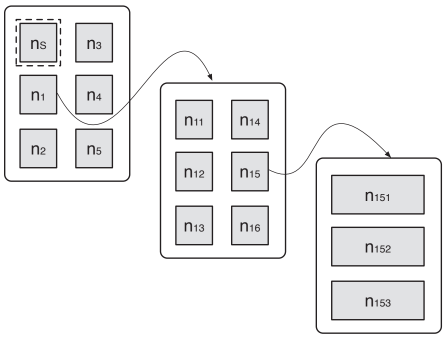


Fig. 1: An abstracted example of the Smart TV app's UI

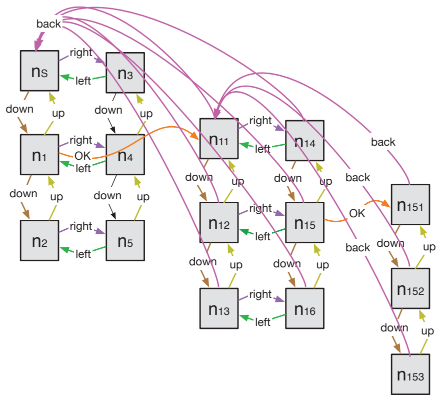


Fig. 2: Sample user interaction model created for the example

B. Model Illustration

In this section, we demonstrate the user interaction model concepts using an abstracted example. Figure 1 shows three screens of a sample Smart TV app that contain various screen elements (N). Element n_s is an initial screen element of the app. Using the remote control device, the user triggers possible transitions in the UI (E), and his focus changes to another screen element during this process.

All possible paths that can be taken in this example are depicted in Figure 2; this is also the model that will be produced by the specialized crawler used in the proposed approach (a detailed description follows in Section V). The outcome of this crawling process is a directed graph generated to model the elements of the app.

C. Parametrization of the Model

User effort depends on mainly the contextual circumstances, which we model by the context C. As an example, we take a system (i.e., a Smart TV app) that is controlled by a person challenged with a serious dexterity issue (i.e., quadriplegic). This person controls the app with a special controller that allows six actions (left, right, up, down, back, OK). Performing the individual actions with the controller requires different effort from the user and allows different efficiency. The basic actions – left, right, back, OK – are easy to perform. By contrast, significantly more effort is required to perform the remaining two actions (i.e., up and down). Hence, in different contexts, the settings of UC, $\mathcal{E}_{dev}$ and $\mathcal{E}_{env}$ would logically be different. Thus, we need to perform an initial setting of these constants, including δ. Additionally, we need to calibrate these constants in the experiments. In this paper, we used six main actions by which the user can interact with the app's UI. Those actions are represented by the buttons UP, DOWN, LEFT, RIGHT, OK and BACK on the remote control device.

TABLE I: An initial model parametrization

action	$\delta(a(e))$	$UC(a(e), C)$	$\mathcal{E}_{dev}(C)$	$\mathcal{E}_{env}(C)$
LEFT	800	1.0	1.0	1.0
RIGHT	800	1.0		
UP	800	1.0		
DOWN	800	1.0		
OK	2500	1.0		
BACK	1500	1.0		

As a baseline, we consider the context C_s, which models a user without any special needs or disabilities. We also consider a standard Smart TV set with no environmental factors that might make the interaction with the Smart TV set more difficult. Table I shows the first setting of δ, UC, $\mathcal{E}_{dev}$ and $\mathcal{E}_{env}$, or C_s, based on our previous empirical investigations. As δ aggregates the time of the user's cognitive preparation to perform an action in the app and the time needed to interact with the UI by the respective remote control button, the value of δ is higher for actions such as OK and BACK. After the pressing OK or BACK button, the user moves to a new UI screen, which must be analyzed before taking the next action to complete a task. Hence, the time needed for cognitive preparation is longer.

V. Automated Model Creation from the Smart TV App

The user interaction model G introduced in Section IV-A is created by a specialized crawler that we developed for this purpose. The crawler starts at a defined screen n_s of the Smart TV and explores its screens. During this process, only the code of the app screen is analyzed, and no knowledge of the internal structure of the app's code is obtained. On each screen, the crawler analyzes the available nested containers by examining each clickable element. During this analysis, each clickable screen or individual nested container is assigned a separate node in the G, being dynamically constructed during the crawling. The exploration process stops when no more clickable element is available to be explored or when a defined termination criterion has been met. The termination criteria are defined by a number of nodes $|N|$ in the created mode G. The termination criteria are used for dynamically generated UIs of apps with an online content (which essentially create an infinite space to explore). When the exploration process terminates, N contains all the examined nodes.

When the crawler arrives at a screen or screen nested container, it examines the user actions available on this screen. This is done by simulating the user's remote control by pressing UP, DOWN, LEFT, RIGHT, OK and BACK buttons. Identified possible actions leading to a transition to next screens or screen nested containers are then added as edges to G.

When the crawler finishes the exploration of the SUT UI, E contains all the possible transitions available from the screens and screen nested containers contained in N. The set N_e contains all screens (or screen nested containers) for which no outgoing action is available (in the case of a well-designed Smart TV app, N_e should be empty).

Regarding the time requirements to create the user interaction model G via the crawler, the initial configuration of the crawler for a new Smart TV app takes up to 30 minutes for a completely new user. The crawling process itself depends on the size of the explored space; however, the run time of the crawler did not exceed 60 minutes for the testing app used in this study.

VI. Automated UI Analysis of the Smart TV App

In our approach, we use the user interaction model G that is generated by our automated crawler. As mentioned in Section IV-A, we designed the crawler to scan the Smart TV app and create the model without knowing the internal structure of the app code. The details of the crawler implementation and the full source code are available for download[2]. A step-by-step running example of the crawler can be found in [30]. The following points detail the concepts of our approach:

1) The UI of the Smart TV app is scanned by our crawler, which creates model G as the output.
2) A set of tested user scenarios T is defined by a tester using. The scenarios include the most frequent user tasks in the app and/or the user tasks in the app that are reported as problematic steps from a usability/accessibility perspective.
3) Each $t \in T$ is mapped to the nodes and edges of G in the MBT framework (details follow in Section VII-C).
4) The user context C in which the defined tested user scenarios T will be assessed is defined. Namely, the values of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ are set for the individual actions that can be invoked by the remote control.
5) The following set of verifications is performed for each $t \in T$:
   a) User scenario path $p(t)$ is constructed for t. If $p(t)$ does not exist, this fact indicates a UI design flaw. If this check is passed, then perform the following checks:
   b) If $p(t)$ is not a simple path, compute the node repetition $nr(p(t)) = \dfrac{|p(t)|+1}{nodes(p(t))}$. Then, $nr(p(t)) > nr_{threshold}$ may indicate possible UI design suboptimality. $nr_{threshold}$ is discussed in Section VI-A.
   c) $|p(t)| > |p(t)|_{threshold}$ may indicate possible UI design suboptimality. $|p(t)|_{threshold}$ is discussed in Section VI-A.
   d) $\mathcal{E}(p(t), C)$ is computed:
      i) $\mathcal{E}(p(t), C) = \infty$ (or division by 0) indicates that $p(t)$ is infeasible for a particular user in context C (typically, a limit for a user with a specific need).
      ii) $\mathcal{E}(p(t), C) > \mathcal{E}_{threshold}$ may indicate possible UI design suboptimality. $\mathcal{E}_{threshold}$ is discussed in Section VI-A.
6) To remove the UI design problems identified in the previous steps, the UI designer can edit G in the MBT environment by adding an edge (or a set of edges), adding a node (or a set of nodes), or generally updating the model. Then, the problematic scenarios can be reanalyzed (repeat step 5) until the defined verification rules are satisfied.
7) The adjustments in G can be transformed to a set of change requests for the UI development team to repair the detected problems or suboptimalities in the Smart TV app.

When step 7 results in a change in the UI, steps 1-7 can be repeated to verify the suitability of the changes from the usability perspective. The whole cycle of steps 1-7 can be repeated several times until the optimal result is achieved.

A. Initial Values of Thresholds

For the verification rules defined in Section VI, step 5, we set the following initial values of the thresholds. We set the value of $nr_{threshold}$ to 1.5, the value of $|p(t)|_{threshold}$ to 20 and the value of $\mathcal{E}_{threshold}$ to 25000. These values are based on our previous experience, and they are further adjusted based on feedback from the experiments in Section VII-D.

VII. Experimental Verification

We have verified our proposed approach in an experimental evaluation study consisting of the technical verification of the methods and experiment with a group of Smart TV users. The following sections detail the experimental procedures and the evaluation results.

A. Experiment Method

The experiments were conducted in a sequence of the following steps:

1) We selected an open source Smart TV app[3] (further referred as testing app) as an SUT to be analyzed by our specialized crawler to create user interaction model G.
2) We configured a special testbed setup for the experiments that consisted of Smart TV environment web simulator with an installed testing app with a special logging mechanism to capture user actions. In addition, this mechanism counts the exact time at which the user executes a particular action (represented by an edge or node of G) on the app and the remote control button that triggered the action.
3) We defined a set of four tested user scenarios T: one to be used as a training scenario for the experiment participants and three to collect the experimental data.

[2]Smart TV crawler download page https://github.com/bestoun/EvoCreeper

[3]https://github.com/daliife/Cinemup

The scenarios were deliberately defined in less detail (capturing only a generally defined user task, not a sequence of main screens and input actions to be visited/achieved on the app). The user scenarios used in the experiment are described in detail in Section VII-B.

4) As described in the method defined in Section VI, each $t \in T$ is mapped to the nodes and edges of G in the MBT environment.
5) For each $t \in T$, we ran the set of verification procedures defined in Section VI. We implemented the verification rules as a part of the MBT platform. The initial configurations of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ are presented in Section IV-C.
6) Concurrently, each $t \in T$ was implemented in the testbed by 25 independent users recruited from the students of a software testing course. The users were instructed to perform the user scenario as specified by t. The logging mechanism logged their activities on the app.
7) We compared the results obtained from the application of the verification rules (step 5) and the independent test by users (step 6). Namely, we compared the total time needed to accomplish the user scenarios and the length of the user scenario paths on the UI measured as the number of transitions, and we analyzed the length of individual user scenario paths invoked on the UI by the remote control buttons. The results are presented in Section VII-D.
8) We repeated step 6 again with another group of 24 participants. The details of the second verification are in Section VII-D.
9) Based on feedback from the comparison results and from the second experiment, we adjusted the configurations of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ. Additionally, we adjusted the values of the thresholds $nr_{threshold}$, $|p(t)|_{threshold}$ and $\mathcal{E}_{threshold}$. We present the updated values in Section VII-D.
10) We repeated step 5 with the adjusted configurations of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ.
11) Again, we compared the results obtained from the verification rules on the app and the independent test by the users to check for improvement in the method configuration. The details of this second verification follow in Section VII-D.

Regarding the details of the experiment participants, we recruited a group of sixty persons from the students of a software testing course: 49 of the participants successfully completed the experiment. There were nine females and 40 males, and the mean age was 23.6 years ($SD = 1.1$). Two participants were left-handed, two participants wear glasses for both long and short distances, 15 wear glasses for long distances, 32 do not need prescription glasses, and only one participant changes between glasses for reading and glasses for looking at a distance. In their routine work (not in the experiments, where the environment was standardized), ten participants regularly use a touchpad as a primary pointing device, one uses a trackpoint and 39 use a mouse.

The first group of 25 participants included three females and 22 males, and the mean age was 23.8 years ($SD = 1.1$). One participant was left-handed. Eleven of the participants wear glasses for long distance vision in the first group. The second group of 24 participants included six females and 18 males, and the mean age was 23.5 years ($SD = 1.2$). One participant was left-handed. Four participants wear glasses for long distance and two wear glasses for both long and short distance. The distribution of the pointing devices used in routine work was similar between groups. In each group, five participants use a trackpad and the others use a mouse.

In the experiments, we took the following measures to prevent the impact of a possible learning effect: (1) participants started the experiment with a training scenario, and the results from these scenarios were not taken into account in the evaluation of the experimental data, and (2) we randomized the sequence of user scenarios to be executed by each of the participants and maintained an overall equal distribution of these sequences.

B. User Scenarios in the Experiment

We considered the following user scenarios in the experiment:

1) Examine all photos from the given movie in the "Popular" section.
2) Count the number of movies in the category "TOP TV."
3) Check if there is a movie with given name in the category "TOP RATED."
4) Count the number of comedies in the category "TOP RATED." To determine if a movie is a comedy or not, use the movie metadata in its attributes.

Scenario 1 was used as a training scenario to allow the experiment participants to become familiar with the testing environment. The results of this scenario were not evaluated further. Scenarios 2, 3 and 4 were used to collect data to adjust the model parametrization and the thresholds used in the UI verification rules.

C. Implementation of the Proposed Automated UI Analysis and Testbed Setup

We implemented the proposed automated method in the development version of the used MBT platform[4]. In this environment, we created an abstract user scenario. We then mapped the steps of the abstract user scenario to the nodes N and edges E of the user interaction model G to compose a tested user scenario t.

During the assessment of t in G, the user scenario path $p(t)$ is found in G and, subsequently, the values $nr(p(t))$ and $\mathcal{E}(p(t), C)$ are computed. The parametrization of context C (configuration of the values of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ) is entered via two CSV files, to which a path is specified.

The computed results can be copied to the clipboard for further processing. For the experiment with the users (Step 6 of the experiment method described in Section VII-A), the Smart TV environment web simulator with the testing app

[4]http://still.felk.cvut.cz/oxygen/

TABLE II: Detailed results for individual input actions - experimental group A

input action	avg. time [ms]	$valid$	$invalid$	avg. time SD
LEFT	1063	83	5	873.98
RIGHT	975	1596	8	814.58
UP	687	4	0	250.58
DOWN	1173	36	0	1383.17
OK	2418	508	31	1739.04
BACK	1293	420	5	861.19

was deployed on a set of 20 workstations with the same hardware and operating system configuration to minimize possible bias caused by different hardware power or operating system configurations.

The software simulation of the remote control device was available in the software runtime environment. To minimize the possible bias in results caused by different layouts or the simulated remote control, the same layout was configured for each of the participants.

D. Experiment Results

During the first phase of the experiment, we collected data from 25 user participants (group A) who successfully completed the assigned user scenarios. In total, the users completed 75 test scenarios and produced 75 user scenario paths. As mentioned previously, we did not consider the training scenario. Regarding the individual steps of the analyzed user scenario paths, we analyzed 2696 path steps, i.e., the transitions between nodes of the app model N triggered by input actions invoked by pressing the UP, DOWN, LEFT, RIGHT, OK and BACK buttons on the remote control. To exclude excessively long transitions from the data processing, we set a threshold of 10 s. We considered user scenario path steps longer than 10 s to be biased, i.e., the user left the interaction with the UI for a certain time and then returned. In total, we excluded 49 steps, leaving 2647 steps in total to analyze. Table II presents a breakdown of the acquired data by individual input actions. In Table II, $valid$ stands for the number of valid actions, $invalid$ stands for the number of invalid actions, and SD stands for the standard deviation.

Table III compares the results from the experiment with results obtained from the automated UI verification conducted on the MBT platform for the initial parametrization of the app model I. Scenario IDs (ID of t in Table III) refer to the numbering in Section VII-B. Scenario 1 was used as a training scenario and is not included in the analysis. The left part of the table presents the results from the experiment with the group of users. Here, avg_time means the average time needed to execute the scenario in milliseconds. SD stands for the standard deviation, and avg_stp represents the average number of steps in the user scenario paths executed by the experiment participants. The middle part of Table III presents the results of the analysis for the scenarios performed using the verification rules proposed in this paper and implemented in the MBT platform. Here, we present $|p(t)|$ and $\mathcal{E}(p(t), C)$ in milliseconds for each t. The right part of Table III compares the experimental results of the user group with the results obtained from the analysis conducted by the proposed method. Here, $DIFF_{stp} = \frac{|p(t)|}{avg\_stp} \cdot 100\%$ and $DIFF_{time} = \frac{\mathcal{E}(p(t),C)}{avg\_time} \cdot 100\%$.

The relatively low correlation between $DIFF_{stp}$ and $DIFF_{time}$ indicates the suboptimality of the initial parametrization of the user interaction model. Additionally, $DIFF_{time}$ for scenario 4 indicates incorrect settings. For fewer steps (60 in the shortest path versus an average of 61 for the experiment participants), the $\mathcal{E}(p(t), C)$ computed by the proposed method is higher than avg_time. Clearly, an update of the user interaction model parametrization is needed in the second iteration of the experiment. We further analyze and discuss the results, including the differences between the data obtained from the experiment with the users and the data obtained from the proposed automated analysis ($DIFF_{stp}$, $DIFF_{time}$ and their correlation), in Section VII-E.

In the second iteration of the experiments, we collected data from another 24 user participants (group B) that successfully completed the assigned user scenarios. Group B was disjunctive from group A, and the participants were distributed randomly between the groups. In total, the users completed 72 test scenarios, producing 72 user scenario paths. Regarding the individual steps of the analyzed user scenario paths, we analyzed 2645 user scenario path steps. We kept the same threshold of 10 s to exclude excessively long transitions. In total, we excluded 59 steps, resulting in 2586 steps in total to analyze. Table IV presents a breakdown of the acquired data by individual input actions for this second iteration of the experiment. After the analysis of the data presented in Tables II, III, IV, and VI, we adjusted the model parametrization. The adjusted values are presented in Table V.

Based on the above analysis, we also updated the values of the thresholds for the UI verification rules presented in Section VI. The value of $nr_{threshold}$ was kept at 1.5, $|p(t)|_{threshold}$ was set to 100 and $\mathcal{E}_{threshold}$ was set to 100000. Table VI compares the results from the experiment with both groups with the results obtained from the automated UI verification performed on the MBT platform using the adjusted model parametrization (refer to Table V).

After the adjustment of the user interaction model parametrization, $DIFF_{time}$ improved compared to the results from the first iteration of the experiment (refer to Table III). The improvement in $DIFF_{stp}$ is not relevant to evaluate, as we compare the shortest path computed by the proposed method in the MBT environment with the paths taken in the app's UI by the participants from both experimental groups. However, the correlation between $DIFF_{time}$ and $DIFF_{stp}$ is more evident in this phase of the experiment, which indicates that the user interaction model parametrization was adjusted to be more accurate. We analyze and discuss the results further in Section VII-E.

E. Discussion

In this section, we discuss and analyze the experimental results in more depth. The first point to analyze is the significant difference between the results achieved by the experimental participants and the results produced by the proposed automated analysis in the case of scenario 3 (see Table VI). The average length of the user path in this scenario was 16 steps for both experimental groups, whereas the result

TABLE III: Results for tested user scenarios - experimental group A

ID of t	Experiment with users				Proposed method		Differences	
	avg_time [ms]	avg_time SD	avg_stp	avg_stp SD	$\mathcal{E}(p(t), C)$	$\|p(t)\|$	$DIFF_{time}$	$DIFF_{stp}$
2	28090	15310.52	27	15.89	20100	23	71.56%	85.19%
3	19164	8095.88	16	7.34	7300	7	38.09%	43.75%
4	91527	33128.95	61	18.67	97000	60	105.98%	98.36%

TABLE IV: Detailed results for individual input actions - experimental group B

input action	avg. time [ms]	$valid$	$invalid$	avg. time SD
LEFT	1178	93	3	1027.16
RIGHT	971	1540	11	745.51
UP	1251	5	0	1196.70
DOWN	1335	30	0	1525.18
OK	2179	482	44	1459.61
BACK	1243	436	1	709.43

TABLE V: Adjusted model parametrization

action	$\delta(a(e))$	$UC(a(e), C)$	$\mathcal{E}_{dev}(C)$	$\mathcal{E}_{env}(C)$
LEFT	1000	1.0	1.0	1.0
RIGHT	1000	1.0		
UP	1000	1.0		
DOWN	1250	1.0		
OK	2000	1.0		
BACK	1225	1.0		

achieved by the automated analysis was 7. The rationale behind this difference is that there were two possible ways to iterate the list of movies in the tested app, either from an initial position in the list to the right or from the initial position to the left. Most users intuitively started iterating the list to the right, which required more steps. The proposed method took the shortest path to accomplish the task by iterating the list to the left. For both experimental groups, only a few actions with the UP button were performed (see Tables II and IV) due to the nature of the tested user scenarios âĂŞ the UP button was not practically required to accomplish the task. Due to low the frequency of UP actions, these data are not considered in the experimental evaluation.

For the RIGHT button, the average times of actions in Tables II and IV are lower than those for the other buttons. The rationale behind this situation is that the RIGHT button was used in iterating the lists of the movies, which takes less time than needed to control other elements in the tested app. On the other hand, the average time for operations using the OK button is longer than that of the other buttons, which is expected because after pressing the OK button, the user usually moves to a new screen of the UI, where it takes time to choose the next action.

When analyzing the difference between the results of the experiment with the users and the results produced by the proposed automated analysis ($DIFF_{time}$ and $DIFF_{stp}$ in Tables III and VI), in both phases of the experiment, the difference itself is not the primary indicator. As the proposed automated analysis can find a better path through the UI, the difference will be present in the comparison. In fact, what is important is the correlation between the $DIFF_{time}$ and $DIFF_{stp}$. In the first phase of the experiment (Table III), this correlation is present but not so strong. However, this correlation significantly increases in the second phase of the experiment (Table VI), which indicates improvement in the configuration of the user interaction model.

Another point to discuss is the values of the thresholds for the UI verification rules presented in Section VI, namely, $nr_{threshold}$, $|p(t)|_{threshold}$ and $\mathcal{E}_{threshold}$. These thresholds can be set by the user based on judgment and experience with the developed app. However, some recommendations must be provided to potential users of the method. The results from the experiments showed that for Smart TV app, the values of the thresholds (especially $|p(t)|_{threshold}$) are significantly higher than the intuitively expected values for web, desktop or mobile app UI design. This difference can be explained by the relatively simple remote control device, which requires more user actions to reach particular elements of the UI, compare to web apps, for instance.

To compare the proposed approach with the available methods in the related areas, we start with manual usability testing of Smart TV apps. Compared to the proposed approach, which is automated, manual assessment of the usability of a smart TV app (e.g. [2], [3], [13]) might take more time and resources. From the performance viewpoint, the proposed automated approach makes the assessment process faster and hence, more repeatable after various changes in the developed smart TV app which might impact overall app development economics.

The proposed approach also differs from the previous proposals in the area of the automated usability testing of Smart TV apps based on a model created from the SUT UI. A comparable candidate here is an HSTM model by Cui et al. [19], based on a state machine and hierarchical structure of the app. In contrast to the approach proposed in this study, the HSTM based model is constructed by scanning the source code of the Smart TV app, whereas in our approach, UI screens are analyzed only. This fact might not impact the performance of the method itself; as it can be considered rather as an organizational constraint.

Additionally, an approach by Cui et al. is considered to be exhaustive as it will detect all the elements, including those that are not clickable. Hence, extensive filtering of those elements in the model is needed to generate an effective interaction model of the UI of the Smart TV app. This represents an extra step in the process and might impact its performance. Our approach does not require the source code of the app. Instead, we use a unique crawler to analyze the app's UI and detect the actual clickable elements, which, in turn, makes the proposed method more flexible.

To compare the crawler proposed in this study to the alternatives available in the literature, crawlers that can be found, focus on different domains than the Smart TV apps (e.g. web applications [20], thick-client app UIs [21] or mobile apps [22]–[24]). Hence, they do not support the goals of our study, and it is a difficult task to compare their performance, as the domain of their operation differs significantly.

TABLE VI: Results for tested user scenarios - experimental groups A and B

ID of t	Experiment with users				Proposed method		Differences	
	avg_time [ms]	avg_time SD	avg_stp	avg_stp SD	$\mathcal{E}(p(t), C)$	$\|p(t)\|$	$DIFF_{time}$	$DIFF_{stp}$
Experimental group A								
2	28090	15310.52	27	15.89	24250	23	86.33%	85.19%
3	19164	8095.88	16	7.34	8000	7	41.74%	43.75%
4	91527	33128.95	61	18.67	85275	60	93.17%	98.36%
Experimental group B								
2	32102	17728.78	30	12.19	24250	23	75.54%	76.67%
3	17937	9463.59	16	7.38	8000	7	44.60%	43.75%
4	85546	20019.32	60	14.31	85275	60	99.68%	100.00%

Regarding the models used by mentioned crawlers, differences are also present in comparison to our study. Nguyen *et al.* [31] are using an event-flow graph (EFG) as a model of the UI. Amalfitano *et al.* [23] employ a state machine as a model. These approaches are not applicable in the case of smart TV apps, which is caused by the different nature of the user's interaction with the app. To give an example, in a mobile app, the spatial distance between two icons (represented by two model states) is not relevant to the transition. In a smart TV app, this distance significantly matters and is expressed by a set of transitions. This difference leads to a different nature of the user interaction model.

Finally, regarding the model-checking approach for UI usability testing, the proposal of Harrison et al. [29] can be compared with our approach. However, Harrison et al. are using temporal logic as an underlying model of the SUT and focuses on the verification of the UI of medical devices. In comparison, our approach uses the User Interaction Model based on a directed graph, and the primary goal is to identify the possible UI design sub-optimalities, so these two approaches are conceptually similar only.

To conclude, we have identified no direct alternative to the approach proposed in this study and from this viewpoint, the proposed method is an original contribution to the filed of automated UI testing of Smart TV apps, based on automated creation of the User Interaction Model from the app UI.

VIII. Threats to Validity

In this section, we discuss issues that might affect the accuracy or objectivity of the results. For each issue, we also consider its possible impact and the actions we took to mitigate the impact.

The first possible concern is that the learning effect during the experiments might bias the results. We prevented the learning effect by two measures: (1) each participant started the interaction with the UI with a training user scenario that was not included in the evaluation results, and (2) each participant was presented a randomized sequence of user scenarios. We kept the distribution of the sequences of user scenarios equal between the experimental groups.

Another concern that might be raised regarding the experiments is the simulation of the Smart TV environment that was used instead of a real Smart TV device, which might influence the measured data and, as a consequence, the accuracy of the suggested user interaction model parameterization (constants UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ). However, the principle of the method and its use case is not affected by this possible limitation. For the Smart TV app, only the user interaction model parameterization constants have to be adjusted. Moreover, the same concern can be raised in the case of different types of remote controls. Here, the composition of the buttons on the remote control and general ergonomics of the device might influence the parametrization.

Additionally, the accuracy of UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ can be influenced by the set of user scenarios and analyzed apps. Achieving perfect and exact parametrization values is not a realistic task and is not a reasonable goal. The proposed method works within a certain tolerance given by the thresholds $nr_{threshold}$, $|p(t)|_{threshold}$ and $\mathcal{E}_{threshold}$, which can be adjusted by the users of the method.

IX. Conclusion

The proposed combination of the SUT user interaction model reconstructed from an actual Smart TV application with a set of verification rules aimed to assess the feasibility and efficiency of user tasks (being the major quality characteristic in usability testing) is, to the best of our knowledge, an original attempt in the field, as such a study has not been published previously.

In contrast to the manual usability testing techniques, the proposed method is automated; therefore, the proposed technique is faster and the possibility of human-made mistakes is lower. On the other hand, if the method is not configured correctly, the findings might be misleading. We minimized this effect by conducting experiments in which we adjusted values of the configuration constants UC, $\mathcal{E}_{dev}$, $\mathcal{E}_{env}$ and δ based on the results of independent test with a group of users.

References

[1] J. S. Dumas, J. S. Dumas, and J. Redish, *A practical guide to usability testing*. Intellect books, 1999.

[2] A. Ingrosso, V. Volpi, A. Opromolla, E. Sciarretta, and C. M. Medaglia, "Ux and usability on smart tv: A case study on a t-commerce application," in *Proc. Int. Conf. on HCI in Bus.* Springer, 2015, pp. 312–323.

[3] I. Alam, S. Khusro, and M. Naeem, "A review of smart tv: Past, present, and future," in *Proc. Int. Conf. on Open Source Syst. & Technol. (ICOSST)*. IEEE, 2017, pp. 35–41.

[4] T. Kim, S. Choi, and H. Bahn, "A personalized interface for supporting multi-users in smart tvs," *IEEE Trans. Consum. Electron.*, vol. 62, no. 3, pp. 310–315, 2016.

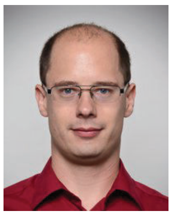
[5] J. Kim, S. Kim, S. Park, and J. Hong, "Home appliances controlling through smart tv set-top box with screen-mirroring remote controller," in *Proc. Int. Conf. ICT Convergence (ICTC)*, Oct 2013, pp. 1009–1012.

[6] J. Kim, E. Jung, Y. Lee, and W. Ryu, "Home appliance control framework based on smart tv set-top box," *IEEE Trans. Consum. Electron.*, vol. 61, no. 3, pp. 279–285, Aug 2015.

[7] S. Y. Chun and C. Lee, "Applications of human motion tracking: Smart lighting control," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. Workshops*, June 2013, pp. 387–392.

[8] M. R. Cabrer, R. P. D. Redondo, A. F. Vilas, J. J. P. Arias, and J. G. Duque, "Controlling the smart home from tv," *IEEE Trans. Consum. Electron.*, vol. 52, no. 2, pp. 421–429, May 2006.
[9] L. Jalal, M. Anedda, V. Popescu, and M. Murroni, "Internet of things for enabling multi sensorial tv in smart home," in *Proc. IEEE Broadcast Symp. (BTS)*. IEEE, 2018, pp. 1–5.
[10] D. Vavilov, A. Melezhik, and I. Platonov, "Healthcare application of smart home user's behavior prediction," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan 2014, pp. 323–326.
[11] C. Fan, Y.-K. Wang, and J.-R. Chen, "Home sleep care with video analysis and its application in smart tv," in *Proc. IEEE 3rd Global Conf. Consum. Electron. (GCCE)*, Oct 2014, pp. 42–43.
[12] E. Erkan, H. R. OzÃğalÄśk, and S. YÄślmaz, "Designing a smart security camera system," in *Proc. 23rd Signal Process. Commun. Appl. Conf. (SIU)*, May 2015, pp. 1705–1708.
[13] D.-H. Shin, Y. Hwang, and H. Choo, "Smart tv: are they really smart in interacting with people? understanding the interactivity of korean smart tv," *Behaviour & information technology*, vol. 32, no. 2, pp. 156–172, 2013.
[14] H. Okada and R. Fujioka, "Automated methods for webpage usability & accessibility evaluations," *Advancesin Human Computer Interaction, In-Tech Publishing, chapter21*, pp. 351–364, 2008.
[15] A. Gimblett and H. Thimbleby, "Applying theorem discovery to automatically find and check usability heuristics," in *Proc. 5th ACM SIGCHI Symp. on Eng. interactive Comput. Syst.* ACM, 2013, pp. 101–106.
[16] R. Cassino and M. Tucci, "Developing usable web interfaces with the aid of automatic verification of their formal specification," *Jour. of Visual Languages & Computing*, vol. 22, no. 2, pp. 140–149, 2011.
[17] R. Cassino, G. Tortora, M. Tucci, and G. Vitiello, "Sr-task grammars: a formal specification of human computer interaction for interactive visual languages," in *Proc. IEEE Symp. on Human Centric Comput. Lang. and Environ.* IEEE, 2003, pp. 195–197.
[18] J. Nielsen, "Ten usability heuristics, available from: https://www.nngroup.com/articles/ten-usability-heuristics/," 1995, checked 2016-01-10.
[19] K. Cui, K. Zhou, H. Song, and M. Li, "Automated software testing based on hierarchical state transition matrix for smart tv," *IEEE Access*, vol. 5, pp. 6492–6501, 2017.
[20] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Trans. Web*, vol. 6, no. 1, pp. 3:1–3:30, Mar. 2012.
[21] A. Memon, I. Banerjee, and A. Nagarajan, "Gui ripping: Reverse engineering of graphical user interfaces for testing," in *Proc. 10th Working Conf. on Reverse Eng. (WCRE)*. IEEE, 2003, pp. 260–269.
[22] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A gui crawling-based technique for android mobile application testing," in *Proc. IEEE int. conf. on Softw. testing, verification and validation workshops (ICSTW)*. IEEE, 2011, pp. 252–261.
[23] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "Mobiguitar: Automated model-based testing of mobile apps," *IEEE Software*, vol. 32, no. 5, pp. 53–59, 2015.
[24] P. Wang, B. Liang, W. You, J. Li, and W. Shi, "Automatic android gui traversal with high coverage," in *Proc. Fourth Int. Conf. on Commun. Syst. and Netw. Technol. (CSNT)*. IEEE, 2014, pp. 1161–1166.
[25] B. N. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "Guitar: an innovative tool for automated testing of gui-driven software," *Automated software engineering*, vol. 21, no. 1, pp. 65–105, 2014.
[26] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media, 2013.
[27] P. Godefroid and K. Sen, "Combining model checking and testing," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Springer, 2018, ch. 19, pp. 613–650.
[28] S. A. Seshia, N. Sharygina, and S. Tripakis, "Modeling for verification," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Springer, 2018, ch. 3, pp. 75–106.
[29] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon, "Verification of user interface software: the example of use-related safety requirements and programmable medical devices," *IEEE Trans. Human-Mach. Syst.*, vol. 47, no. 6, pp. 834–846, 2017.
[30] B. S. Ahmed and M. Bures, "Testing of smart tv applications: Key ingredients, challenges and proposed solutions," in *Proc. Future Technol. Conf. (FTC)*, K. Arai, R. Bhatia, and S. Kapoor, Eds. Cham: Springer Int. Publishing, 2019, pp. 241–256.
[31] B. N. Nguyen, B. Robbins, I. Banerjee, and A. Memon, "Guitar: an innovative tool for automated testing of gui-driven software," *Automated Software Engineering*, vol. 21, no. 1, pp. 65–105, 2014.

Miroslav Bures received his Ph.D. at Czech Technical University in Prague. His research interests are model-based testing, path-based testing, data consistency testing and combinatorial interaction testing, effective test automation (test automation architectures, assessment of automated testability, economic aspects) and quality assurance methods for Internet of things solutions.



Miroslav Macik received his Ph.D. at Czech Technical University in Prague. He currently works at the same institution as a researcher in the field of Human-Computer Interaction. His research focuses on model-based design and evaluation, haptic interaction, accessibility.

Bestoun S. Ahmed obtained his Ph.D. from University Sains Malaysia (USM) in 2012. Currently, he is a senior lecturer at the department of mathematics and computer science, Karlstad University, Sweden. His main research interest include Combinatorial Testing, Search Based Software Testing (SBST), and Applied Soft Computing.

Vaclav Rechtberger is PhD. student in Software testing Intelligent Lab (STILL), dept. of Computer Science and Engineering, Czech technical University in Prague. His focus is Model-based Testing, test automation and testing of Internet of Things systems.

Pavel Slavik is full Professor of Computer Science and member of HCI group at Czech Technical University in Prague. His fields of interest are visualization, usability and accessibility. He served in the past as an IPC member for several HCI conferences.