Robot-Assisted Smartphone Performance Testing

Teemu Kanstrén*, Pekka Aho*, Arttu Lämsä*, Henar Martin†, Jussi Liikka*, Miska Seppänen*
*VTT, Oulu, Finland
email:firstname.lastname@vtt.fi
†ESG GmbH, Germany
email: henar.martin.rodriguez@gmail.com

***Abstract*—This paper describes experiences and lessons learned in applying an approach of using a physical robot for testing overall smartphone device performance based on user profiles. The process consists of capturing user actions, abstracting them to usage profiles, transforming these into test models, and generating test cases from the models. The goal is to support performance testing of touch screen devices and applications in a realistic test environment. To achieve this, the tests are based on real-world generated user profiles and executed using a real physical robot, simulating the actual user. Our performance testing targets different attributes of performance such as response times, power and resource usage, and software/hardware aging. Use of different hardware and software configurations in the test scenarios is considered. This work has been performed in close collaboration with industry partners in robotics provider and user industries.**

I. Introduction

Various characteristics have to be taken into account when testing embedded devices, which are in practice hybrid hardware/software (HW/SW) systems with complex interactions between different parts. A specific case are modern smartphones that can be seen as miniaturized computers running complex applications on top of a largely SW based platform. As such, emphasis in their testing is often put on the software applications running on them.

However, the overall user experience when using such applications is also dependent on the overall performance of the system, including HW and SW configurations. The end user observes the effects in various ways, e.g., response times and power consumption. Various types of users, e.g., power users, elderly people, businessmen and teenagers, may have different user expectations, resulting in variance of the user experience even across the same HW/SW configuration.

To test the relevant performance characteristics of smartphones for different users, we need to be able to capture how users use their devices, identify different types of users, and use this information to accurately simulate these users while collecting the performance measurements on the device. Accurately performing such interactions and measurements at the device level requires that we must be able to interact with it similar to a real user, and we must be able to capture measurements with minimal probe-effect, i.e., minimize the side effects caused by the test environment.

In this paper we present a process for robot-assisted smartphone performance testing, addressing these requirements. User profiles are created automatically based on captured observations of actual smartphone use. These profiles are transformed into suitable test models to generate test inputs for the system under test (SUT). A physical robot is used to interact with the device similar to a real user, and measurements are captured external to the system, minimizing the probe-effect.

The contribution of this paper is in providing an overall process of 1) capturing actual user profiles from actual smartphone use, 2) using a physical robot for automated smartphone testing, and 3) using the resulting test data to analyse product performance. Additionally, we present experiences and lessons learned in implementing such an environment.

Possible application areas of such techniques include profiling product performance and resource consumption under different configurations and user profiles, and optimizing this performance. Different views on performance in this case include response times, power and resource usage, and HW/SW aging in different configurations and scenarios. This paper is based on the results of a project performed in close collaboration with industry partners on improving methods and finding new application areas for robot-assisted testing of smartphones.

The rest of the paper is structured as follows. Section II presents background concepts. Section III describes how we build user profiles and use these for test generation. In Section IV we discuss our experiences in using the approach, the possible application areas, and benefits of our testing approach. Finally, conclusions end the paper.

II. Background

A. Terminology

Our prototyping environment targets Android based mobile devices and uses some Android specific terminology. One central term in this context is an *activity*. An activity groups several user interface views into one. These grouped views typically represent a single action performed by the user, such as writing an email.[1]

We use the term *probe-effect* to describe the impact a measurement has on the system being observed. For example, when specific SW probes are installed on a system to provide measurements or to control the system, they will always alter the system behaviour in some way, minor or major. A typical example is the performance impact, where the probes themselves take some of the resources of the system to perform their task.

978-1-4799-8757-3/15/$31.00 ©2015 IEEE

We use *Markov models* as a basis to model the user behaviour. A Markov-model is a stochastic model that assumes the Markov-property. In a *n*th order stochastic model the current state depends on the *n* previous states in a non-deterministic way. The Markovian property defines that the probabilistic choice of next state is only dependent on the previous state (first-order), not on the path through which the state was reached. In our case, we use first-order Markov Models.

B. Related Work

Profiling deployed SW for testing has been discussed extensively in [2] from the conformance testing perspective. Creating models from such profiling data is typically called specification mining. For example, [7] present an approach for behavioral specification mining based on sets of predetermined patterns observed over the collected data. In our case, we extend the use of such monitoring data as a basis for smartphone user profiling and performance testing.

The summarization of user sessions into probabilistic user models and their use for test generation is discussed in [5]. The benefit shown in [5] is reduction in the number of test cases required to reach required coverage, and increased probability of finding issues due to increased coverage. The models used in [5] are Event-Flow-Graphs, which have the disadvantage that they are not state sensitive as nodes represent events and their edges represent the relationships between them. State is not included. Our approach based on Markov Models overcomes this problem, meaning that the current open activity is used as model state, defining which part of the Markov model is enabled next. For example, the test engine can open an email application from the Android launch screen but not from inside the email activity itself.

Beyond test generation, user profile information has also been applied to post-mortem failure analysis as discussed in [8]. In this case, actual failure scenarios are repeated on the user device in order to collect data from the failure and to use it for debugging purposes. In [8] the system described for this is SW based, adding extra probe-effects of the system under test. Our robot-based approach makes also possible to apply such techniques without the intrusiveness of SW probes, and with minimized probe-effect.

Similar to our work, running tests based on user profiles on mobile devices while using automated mechanical tools to assist the test execution is discussed in [9]. There, a servomotor is used to move the phone to provide some movement related input (e.g., device tilt). Actual user interaction is stimulated using the Android Debug Bridge (ADB). These functions are then used to assess the performance of the developed hardware elements (memory chips in this case). Three types of user profiles are created manually; Normal user, Smart user, and Businessman. In our case, the generation of different types of user profiles is automated and the resulting models are used for producing more diverse test sets. For the mechanical part, we target a broader overall user interaction automation (beyond just tilt) without SW instrumentation and probe-effect.

In addition to basic performance testing of response time latencies, we also provide means for accurate resource consumption analysis more broadly. A related approach to this is presented in [10], where detailed resource usage data is collected from device use with software based probes for measurement and test execution. In this regard, we provide additional accuracy by avoiding the need for intrusive measurement components in the device itself, and for providing accurate replay at actual physical device level and not just at the SW level.

Ways to reduce the probe-effect when executing tests and profiling mobile SW have been discussed in [11], where the SW is executed in an emulator environment with the probes embedded in the virtual machine instead of the profiled application. Such an approach is less intrusive on the application but still highly intrusive on its operational environment. In relation to this, we also avoid the platform software probe-effect.

Finally, a robot-based remote testing platform for mobile applications is presented in [12]. It allows a tester to interact with a mobile device remotely through controlling a robot that has a mechanical moving arm and finger for interaction. Test scripts can be written in the form of keywords to input text in specific locations, and to press icons. Optical Character Recognition (OCR) and image recognition techniques are used to find the intended locations for interaction. We extend the applicability of this type of approach to wider set of systems, to be more accurate in terms of user profiling and provide detailed experiences in application of these to different types of performance testing.

III. Behaviour Modelling and Test Generation

In this section, we describe the probabilistic models, the system architecture, and the test generation process of our approach.

A. Markov model generation

In our Markov-Model, the transitions are paths leading from one Android activity to another. That is, based on a given probability, the user may move between the different activities in the smartphone user interface. The activity that is selected at a specific point of time depends only on an event triggered by carrying out an action (e.g., pressing a button or selecting a menu option) on the user interface of the activity that is currently running. This is the Markovian property, where the previous events are not taken into account.

Figure 1 shows an extract of one of our Markov Models. In order to characterize a Markov Model the following parameters must be defined [13]:

- The number of states in the model (N) being
$$S = \{S_1, S_2, ..., S_N\} \quad (1)$$
the individual states.
- The number of distinct observation symbols per state (M) being
$$V = \{V_1, V_2, ...V_M\} \quad (2)$$

978-1-4799-8757-3/15/$31.00 ©2015 IEEE

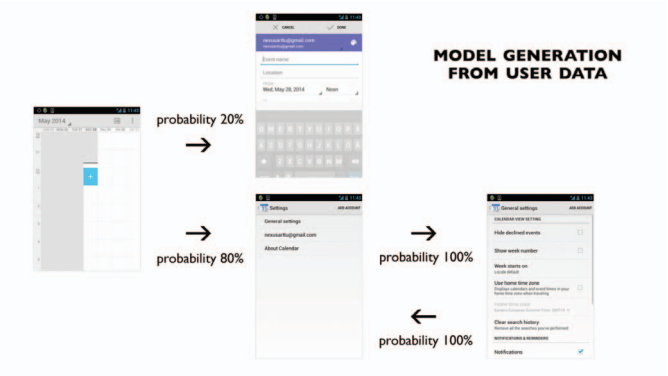

Fig. 1: An example of a generated Markov model

the individual symbols.

- The state transition probability distribution

$$A = a_{ij} : a_{ij} = P[q_{t+1} = \frac{S_j}{q_t} = S_i], 1 \leq i, j \leq N \quad (3)$$

- The observation symbol probability distribution in the state j,

$$B = b_j(k).B_j(k) = P[v_k at \frac{t}{q_t} = S_j], \quad 1 \leq j \leq N, 1 \leq k \leq M. \quad (4)$$

  The sum of all probabilities out of a state in each case is 1.

- The initial state distribution

$$\pi_i = P[q_1 = S_i], 1 \leq i \leq N \quad (5)$$

Given a sequence of observations captured from the user interactions $O = [O_1, O_2, ..., O_T]$, the problem that must be solved is the computation of the parameters of the Markov Model $\lambda = [A, B, \pi]$ that maximizes the probability $P(\frac{O}{\lambda})$. This is carried out using the Baum-Welch method [13].

As a result, the model generation is based on how many times the user has switched from one activity to another. For example, if the user has switched from activity A to activity B 5 times out of 20 and from A to C 15 times out of 20, the probabilities are A ->B 25% and A ->C 75%. The set of touch events used to change from one activity to another is stored along with the transition, allowing for the test generator to reproduce this path.

To create the model, the collected data is separated for each user and periods of usage. A specific period of usage is identified by separating events based on time gaps in the use of the device. That is, once the idle period exceeds a given threshold, the current usage scenario is considered to have ended. Every chain of activities is considered to start from the launch screen (activity) of the Android OS. The end of the chain is considered the last activity before the observed ending timeout. The test generator inserts the activity to open the launch screen as the final step each time.

Events are stored from user touch events (read from the Android device /dev/input/eventX) and system log (logcat). Both logs are timestamped using the Android system clock. The system log identifies activity changes, which are synchronized with touch events using timestamps. These together form the basis for the states and transition paths in the Markov model.

There can be various paths from one activity to another, and all paths taken by the user are stored as potential paths in the Markov model between the associated activities. Figure 2 shows an example of four paths in using an Email application. The first going from the Launch screen activity to the main Email activity, the second one from the Email main activity to the compose email activity, the third from compose email to list of sent emails, and finally back to the Launcher screen.

The first path consists of a single touch event on the Email icon. The second path consists of a single touch on the compose message icon. The third path is more interesting, and consists of several touch events on the virtual keyboard on the screen. Each path in the Markov model is stored exactly as recorded, starting from opening the activity until another activity has been opened instead.

That is, inside a single path (Markov transition) there is no change from how the user has interacted with the application. Instead, all paths from one activity to another taken by the user are stored and one of these is chosen randomly to move through the Markov model. In case of path 3 here, it means repeating the exact typing of the same message. This ensures that the actions taken are valid (e.g., if we randomized path events one could end up randomly choosing to close the email keyboard and then trying to type on it after closing it).

Fig. 2: An example of paths in Markov model

In our case a component called MarkovAnalyzer was created to produce both a single summarizing model for all users as well as a separate model for each user. The generated models contained descriptions of the Android activities used, the transition probabilities between them and the touch event paths leading from one activity to another.

B. Architecture

Our test system architecture is illustrated in Figure 3. It has three main parts; mobile clients, the server and the test agent. The mobile clients are used for collecting the device usage data. The server collects, stores and converts this data into Markov model suitable for test generation. Finally, the test agent generates test cases from the model and executes them by controlling the testing robot.

The mobile client was developed for Android 4.2 and the devices used for collecting the data were LG Google Galaxy Nexus 4 smartphones. The mobile client collected the following data and stored it into the memory of the device:

978-1-4799-8757-3/15/$31.00 ©2015 IEEE

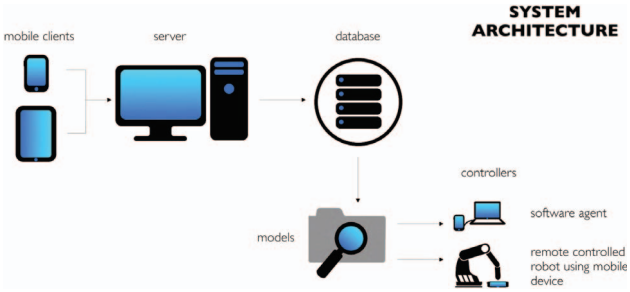

Fig. 3: System architecture

- Android activity change events - timestamp, name of activity and information whether the activity was brought to foreground or sent to background.
- Screen touch events - timestamp, x- and y-coordinates, orientation.
- Hard button events - timestamp, id of the button pressed (e.g., volume up/down), information whether the button was pressed down or released.
- Screenshots - whole screen and partial screenshots, including capture time, activity name, and coordinates of the partial screenshot.
- Icons of the activites used.

The collected data is sent to the server once a day. If no network connection to the server is available, the data is buffered until delivered. The server stores the data and transforms it into a Markov model (see Figure 1).

C. Test Generation

Test generation from the Markov model uses random walks through the Markov model, guided by the probabilities that describe the transitions. Each transition taken in the Markov model is mapped to a test step, which consists of the test robot performing a sequence of touches for the transition path on the device screen at the matching coordinates. This also includes the timing of the profiled user (delay between each action). There are several possible paths from one state to the other, due to different interaction sequences in those paths. For example, the user may have written different email messages while using the *write email* activity at different times (producing different paths), or they might have clicked on different parts of the email icon when opening the email activity (a different path again). As noted, When several such paths are available, the choice between them is made randomly.

The generated tests are stored in text files, where each line represents a step in a test case. A line contains the X,Y-coordinates and a delay following a single touch event. The test agent then controls the test robot using this information as the set of steps for the test case. Beyond basic type touch events, the user can also perform swipes. Swipes are gestures where the user places a finger on the screen and drags it in one direction. These are identified in the traces as several touch events with very small delays in between. As the delays are too small and too close apart for human to perform in other ways, they are interpreted as swipe gestures. For the robot execution, they are represented as a set of commands where the artificial finger is set on the screen, moved and lifted.

We used two types of robots for our experiments as shown in Figure 4. The first one (Robot A in Figure 4) was used as a prototyping environment. The second one (Robot B in Figure 4) over a remote connection in our robotics provider lab, as an industrial scale robot for final performance testing. The main difference is that the industrial robot can handle higher speeds, has specific function support (e.g., pinch and zoom), is more accurate and has a high definition camera for external monitoring and for machine vision support.

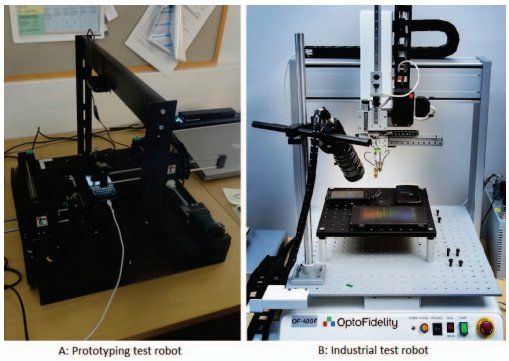

Fig. 4: Touch-screen testing robots

Before executing the test cases, the robot has to be calibrated because of the differences between the coordinate spaces of the robot and the smartphone. The recorded touch events are described in terms of screen pixels. The test robot itself uses millimeter precision events to move its 'finger'. These need to be calibrated to match each other. To do this, first we select two points diagonally on the screen of the smartphone. When the robot touches these points, the smartphone is set to record the coordinates where the touch event is observed in its own coordinate system on the screen. These coordinate pairs are then inputted to the robot control software, which uses these to create a transformation between the two coordinate spaces. The assumption in this case is that the smartphone is placed in line with the coordinate axes of the robot.

IV. Experiences and Discussion

To evaluate the approach, we initially collected six months of data from 19 users during the year 2013. The collected data was used to generate Markov models to describe the different types of users and these models were used to generate test cases that were executed using the robot.

A. Experiences

1) Hardware: Graphical user-interface (GUI) based testing is typically slow due to the overhead of all the work needed to create and update the GUI by the operating system. In our robot-based approach, further delays are added due to the capabilities of the robot hardware itself. It takes time for the robot to move around its arms and to perform actions. For fast use cases, it may not be able to accurately repeat the use case at the same speed as the actual user did. This is not as big

978-1-4799-8757-3/15/$31.00 ©2015 IEEE

an issue for high-grade industrial robots (such as [15] shown as Robot B in Figure4), which have more powerful hardware and better designed mechanisms, but it is still an issue to be considered.

Partially this may be mitigated by having a more intelligent control server, which is able to buffer more commands and wait for the previous ones to be completed. However, this also requires having some added intelligence in the robot itself being able to report back to the control server on the results of the actions, such as when a given operation has been finished. More advanced features (e.g., two-way communications, machine vision and image recognition) are required on both sides to be able to synchronize with the control server and to carry out the use cases with changes in the operational status and performance of the system. This should be considered in the design of the overall test automation system, as this type of information (e.g., delays, speed of use, potential issues) is valuable in itself, providing more detailed information on the delays and overall responsiveness of the device and applications.

Where the robot based testing approach may not scale to very large test sets by itself due to slowness of GUI testing, the user-profile based test generation approach can be a very useful tool to mitigate this issue. Since we know from the different user profiles which types of users use the different types of actions and activities most, we can also use the profiles to guide the testing to focus the most on those parts.

Finally, mobile device hardware and interaction mechanisms are constantly changing as new mechanisms, e.g., curved displays, voice control commands, and gesture controls, are introduced. As such, new techniques are required to address new types of interaction mechanisms for realistic probe-effect free testing.

2) Accuracy: While we calibrate between the screen coordinates and the robot-coordinates, there can still be issues related to the differences in coordinate spaces. For example, if the user touch event occurs at the edge of a button, it may be shifted just enough in the repeating scenario to miss the actual mark. In our experience, such instances are very rare, but highlight how important is the careful calibration of the robot coordinates with the recorded coordinates.

Additionally, it is possible to record 'extra' events during the recording phase, e.g., recording an accidental touch of the screen by the hand holding the phone. Unless we can exactly replicate the behavior of the operating system in each case, automatically determining which of these touches were intended, which ones accidental and most importantly which ones the operating system ignored due to some specific multi-touch rules, is not feasible for an external recording entity. This can cause problems, although again such issues are rare in our experience. Possible solutions could include recording common patterns to activate specific actions in specific states of the system under test (SUT) and use these to filter out extra information when anomalies are observed.

On the other hand, in some cases it might be useful to use the robot to accurately mimic the exact user behaviour, including the unintended touch events as those can also in practice cause issues, which is useful information for product development. This is a topic we plan to investigate further as part of our future work in using user profiles for smartphone testing.

3) Test Oracles and Dynamic Tests: Different types of test oracles for different types of scenarios are required. As we are mainly concerned with performance testing in this case, the main test oracle to use is to measure performance attributes such as latencies in user interface and in different functions, or measuring power and resource consumption of the device.

For testing that needs to make more specific assertions we would need more detailed test oracles, such as being able to assert that a specific activity opens as a result of given stimuli, the stimuli being generated by the tester as based on the Markov model. However, such information is also required in performance testing even if not as accurately. To measure latencies in the user interface, we need to be able to identify which activities are opened at which times.

A second part where this is required is in providing dynamic user interface traversal for test cases. If we simply base our test scripts on delays between events, these will fail given the first difference in system response times. That is, when the performance fluctuates the model may get off the path if it is unable to respond to changes in latencies and just keeps clicking after hard-coded delays.

B. Discussion

Considering the interaction with the different elements of the user interface and with different touch intensivities, we can use the robot to test how well the device responds to different levels of touch sensitivity. An approach for this is to use different types of 'fingers' on the robot arm to interact with the device. A bare finger or a finger with different types of gloves can be simulated using different heads.

By using the robot we are able to accurately measure how different operating conditions affect the performance of the device and applications over long term. For example, profiling battery performance for mobile devices is an important aspect for many device manufacturers. Realistic estimates for this would require having users repeat the same usage scenarios continually. Performing such testing manually is tedious and also error prone in re-executing the exact same scenarios every time. The robot enables carrying out a single scenario or generating several similar scenarios based on the user profile (and systematically repeating those as needed).

Beyond simply measuring power consumption, the robot assisted approach can also reveal interesting information about the performance of the device in various configurations. For example, when executing the same test scenarios using the same device with different battery levels and the battery charger plugged in and unplugged, we noticed that the timings of the applications on the device changed based on whether the charger is plugged in or not. When the charger was plugged in, the performance of the device was faster, even with the exact same settings on the device. Using manual testing, analysing

978-1-4799-8757-3/15/$31.00 ©2015 IEEE

such issues was harder as we could not be sure whether there was a difference in the conditions due to limitations of human perception. Using the robot with automated measurements, the results are much more accurate.

In addition to power consumption, other types of resource measurements could be made as well. As the robot repeat the same operations continuously, it is possible to observe memory leaks and similar issues that causing performance degradation on the device and applications over long term usage. As the robot repeats the same tests and measures the performance, we can be sure these issues are not caused by the test framework itself (since there is no software instrumentation) and that the user interaction remains consistent. This type of approach can also be applied to a number of different hardware configurations testing for durability, performance and other elements (e.g., as memory in [9]).

C. Potential Improvements and Future Work

A basic use for image capture and analysis in this type of testing is using them for comparison to identify when a given Markovian state is achieved. However, they can also be very useful for reporting such as illustrated by the figures in this paper. The coverage of the test runs could be analyzed visually (e.g., the unvisited views could be tracked).

More generally, with regards to image recognition from the adaptability perspective, an interesting extension would be also to be able to adapt to different icon sets and applications installed in the device (e.g., activity trigger to test is described in a different coordinate systems, is in a different screen in the launcher, or uses different default icons from a different manufacturer). This is not difficult for a human tester but requires integration of several advanced techniques for the robot-based environment, such as image recognition and modelling of general smartphone operating system logic to search for the triggering icons in different potential locations. Similarly dynamic aspects of applications should be considered, such as changing lists of sent or received emails or web-page advertisement blocks.

In this paper we have mostly focused on generating test inputs based on user profiles, and using these as a basis for measurements in non-functional testing. To extend our test approach into the functional testing domain, more accurate test oracles and test models are required. For example, intelligent machine vision based evaluation or specific GUI based test specification languages (e.g., [16]) could be used. Specific test oracles for mobile application user interfaces, such as discussed in [17], could be integrated to provide more extensive coverage in functional domain.

Generally, improvements in analysis of the results have strong potential. The analysis should allow us to provide more information to support product design and problem area identification. For example, comparison of robot-performed test cases and the traces produced in previous test iterations could be used to gain information on how the different configurations and evolution of the HW/SW versions impact the different properties, such as performance of the system over its evolution. Combining this with other product information can also provide support for more detailed analytics on product and user experience optimization.

V. Conclusions

Robot-assisted testing is still a new and emerging topic, and a lot of work remains to make it more widely industry applicable in terms of cost and performance. The work we have presented in this paper provides a foundation to take these requirements further. While with the work presented in this paper we can see using these approaches successfully for non-functional testing such as different aspects of performance testing, extension into more detailed conformance testing space is still a challenge. In the future, we can also see that robot-assisted testing has to be able to address new types of emerging interaction techniques such as voice and gesture based controls.

References

[1] *Activity — Android Developers*, http://developer.android.com/reference/android/app/Activity.html, referenced 13 Nov 2014.

[2] S. Elbaum and M. Diep *Profiling Deployed Software: Assessing Strategies and Testing Opportunities*, IEEE Transaction on Software Engineering, April 2005, vol. 31, no. 4, pp. 312-327.

[3] A. Machiry, R. Tahiliani, M. Naik, *Dynodroid: An Input Generation System for Android Apps*, Proc. of 9th ESEC/FSE Meeting, Saint Petersburg, Russia, August 18-26, 2013

[4] W. Choi, G. Necula,and K. Sen,*Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning*, SIGPLAN Notes, 2013, Vol. 48, No. 10, pp. 623-640

[5] P. Brooks and A. M. Memon *Automated GUI Testing Guided by Usage Profiles*, Proc. 22nd IEEE/ACM Int'l. Conf. on Automated Software Engineering (ASE), 2007, pp. 333-342.

[6] D. Amalfitano et al., *MobiGUITAR - A Tool for Automated Model-Based Testing of Mobile Apps*, IEEE Software, 2014, to appear, doi:10.1109/MS.2014.55.

[7] G. Reger, H. Barringer and D. Rydeheard, *A Pattern-Based Approach to Parametric Specification Mining*, Proc. 28th IEEE/ACM Int'l. Conf. on Automated Software Engineering (ASE), 2013, pp. 658-663.

[8] S. Joshi and A. Orso *SCARPE: A Technique and Tool for Selective Capture and Replay of Program Executions*, Proc. IEEE Int'l. Conf. on Software Maintenance (ICSM), 2007, pp. 234-243.

[9] G. Canfora et al. *A Case Study of Automating User Experience-Oriented Performance Testing on Smartphones*, Proc. IEEE Int'l. Conf. on Software Testing, Verification and Validation (ICST), 2014, pp. 66-69.

[10] T. Ohmann et al. *Behavioral Resource-Aware Model Inference*, Proc. 29th IEEE/ACM Int'l. Conf. on Automated Software Engineering (ASE), 2014, pp. 19-30.

[11] C-H. Tu et al. *Performance and Power Profiling for Emulated Android Systems*, ACM Transactions on Design Automation of Electronic Systems, vol. 19, no. 2, March 2014.

[12] K. B. Dhanapal et al. *An Innovative System for Remote and Automated Testing of Mobile Phone Applications*, Service Research and Innovation Institute Global Conference, 2012, pp. 44-54.

[13] A. Rabiner, *Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, 1989, Vol. 77, No. 2, pp. 257-286.

[14] Bitbar *Mobile App Testing on Android and iOS Devices*, http://testdroid.com/, referenced 3 Oct 2014.

[15] Optofidelity *Optofidelity Test Automation*, http://www.optofidelity.com/, referenced 3 Oct 2014.

[16] F. Zaraket et al. *GUICop: Specification-based GUI Testing*, Proc. IEEE 5th Int'l. Conf. on Software Testing, Verification and Validation (ICST), 2012.

[17] R. N. Zaeem, M. R. Prasad, S. Khurshid, *Automated Generation of Oracles for Testing User-Interaction Features of Mobile Apps*, Proc. IEEE Int'l. Conf. on Software Testing, Verification, and Validation (ICST), 2014, pp. 183-192.

978-1-4799-8757-3/15/$31.00 ©2015 IEEE