Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 284 – 291

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

Impact of execution modes on finding Android failures

Inês Coimbra Morgado[a,b,*], Ana C. R. Paiva[a,b]

[a]*Department of Informatics Engineering, Faculty of Engineering of University of Porto, Porto, Portugal*
[b]*INESC TEC Porto, Porto, Portugal*

Abstract

The iMPAcT tool combines the benefits of existing user recurring behaviour (User Interface Patterns) on mobile applications to facilitate the test automation of Android mobile applications. It uses an automatic exploration process combined with reverse engineering to identify the existing user interface patterns on a mobile application and then tests those patterns with generic test strategies (designated Test Patterns). The Test Patterns are defined in a catalogue that can be reused for testing other applications. However the results obtained by the iMPAcT tool depend on the exploration mode and on the order in which the test strategies are applied. This paper describes an experiment conducted to evaluate the impact of using different exploration modes and of changing the order by which UI patterns are searched and subsequently tested on the failures found and on the number of events fired.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Mobile Testing; UI Patterns; Reverse Engineering; Android; Test Automation; Case Study

1. Introduction

Smartphones have been gaining a strong participation in our daily lives. Furthermore, the number of mobile applications available has exceeded one million and the number of downloads has exceeded fifty billion[1].

This huge number of applications increases the rivalry among suppliers that need to ensure that their applications work correctly as thoroughly as possible if they want them to make a stand. Even though there are several approaches focused on test automation[2–6], the peculiarities of the mobile world, such as new development concepts, like activities, new interaction gestures and limited memory, make mobile testing a challenging activity[7,8]. The World Quality Report 2014-15[9] mentions that the greatest challenge for mobile testing is the lack of the right testing processes and methods, followed by insufficient time to test and the absence of in-house mobile test environments. Thus, it is extremely important to automate mobile testing.

Reverse engineering[10] is a technique used to aid in test automation as it provides information about the AUT. When reverse engineering a mobile application, dynamic techniques may be more appropriate than static ones due to the event-based nature of mobile applications: in this kind of applications most of the content is produced dynamically,

* Corresponding author. Tel.:+351225081400, fax:+351225081440
E-mail address: pro11016@fe.up.pt

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2016.04.127

e.g., the information presented on the screen depends on the order of the events previously fired, which is extremely difficult to analyse statically.

There are also some studies showing the usefulness of using patterns for mobile testing. In 2009, Erik Nilsson[11] identified some recurring problems when developing an Android application and the User Interface (UI) patterns that could help solve them. In 2013, Sahami Shirazi *et al.*[12] studied the layout of Android applications trying, among other goals, to verify if these layouts presented any UI Patterns. They concluded that 75.8% of unique combinations of elements appeared only once in the application. This study was conducted taking into consideration a static analysis of the layout and its elements. There is also some literature on the presence of UI Patterns on mobile applications, such as[13].

One of the main problems of dynamic reverse engineering is the dependence between the order in which the application is explored and the results obtained. The exploration may be random or guided, *i.e.*, follow an exploration algorithm, such as depth-first.

The iMPAcT tool[14] combines the benefits of existing UI Patterns and reverse engineering to ease mobile test automation. The tool uses dynamic reverse engineering to detect the presence of UI patterns and then tests them using generic test strategies, called Test Patterns (more details in Section 3). The results obtained by the tool depend both on the order in which the different events are executed and on the order the different patterns are identified and tested. This paper reports the results obtained by the iMPAcT tool when using different exploration modes and different Test Patterns testing orders.

The remaining of the paper is structured as follows. Section 2 presents some related work. Section 3 describes the approach implemented in the iMPAcT Tool. Section 4 presents the case study and corresponding results. Section 5 presents the drawn conclusions.

2. Related Work

Model based testing (MBT) approaches generate test cases from models according to coverage criteria. Even though MBT generates test cases automatically, the effort invested in building the model should not be neglected. To reduce this effort it is possible to use reverse engineering approaches to extract part of an existing application model and work from there.

Software reverse engineering has long been a field of research. There are several approaches that extract models from desktop[15–17] and web[18–20] applications. Regarding the mobile world, to the best of our knowledge, there is no approach focusing Windows Phone applications and only a handful of approaches dealing with iOS application, in the last years.

Even though reverse engineering approaches can be static (when they analyse the source code), dynamic (when they analyse the application at run time) or a combination of both (hybrid), the event-based nature of mobile applications makes the dynamic and hybrid approaches more common. Nevertheless, there are some, like Batyuk *et al.*[21] who identify possible security vulnerabilities like unwanted user access, by applying static approaches.

Regardless of the techniques used, the purpose of reverse engineering in the context of mobile applications is to obtain a model of the application and/or to test it. The work of Yang *et al.*[22] is an example of the first as it presents a hybrid approach: an initial static phase identifies the possible events to be fired and a second dynamic phase explores the application by firing those events and analysing their effects on the application. An example of the latter is the work of Amalfitano *et al.* in 2012[7] which is similar to the exploration phase of the approach presented in this paper and in 2013[23], in which they generate test cases but follow a dynamic approach with reflection and code replacement techniques.

Mobile application testing (or mobile testing) has been gaining interest by researchers because it presents additional challenges when compared to the testing of other types of applications, such as, web or desktop[8].

Mobile testing can be performed automatically and the market already offers several automatic options for test case execution, including the two official ones for Android: Espresso[1] and UI Automator[2]. The main difference between

[1] https://developer.android.com/training/testing/ui-testing/espresso-testing.html
[2] https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html

these frameworks is that Espresso only enables interacting with the application under testing while UI Automator enables interacting with the device and all its applications and settings.

According to Gao *et al.* [24], the main mobile testing purposes are: UI testing, mobile connectivity testing, usability testing and mobility testing; and the four main testing approaches are: emulation-based testing, device-based testing, cloud testing and crowd based testing.

There are several works focused on mobile testing. For instance, Amalfitano *et al.* [25] automate crash testing on Android applications, Machiry *et al.* [26] generate inputs to test the application and Costa *et al.* [27] generate test cases from a pattern-based model of the application and Nasim *et al.* [28] identify malicious code automatically injected in the application's source code.

Usually tests can not be exhaustive and, thus, it is necessary to select which tests to perform or to select a subset of the overall behaviour to test. There are some recent approaches that only test recurring behaviour on software applications, the so called UI Patterns [27,29–31].

Even though the presence of patterns can be useful for testing mobile applications [11,12], sometimes, different approaches focus on different types of patterns. For instance, Hu and Neamtiu [32] associate the notion of pattern with possible failure classes, *i.e.*, typical places where failures can occur (activities, events); Batyuk *et al.* [21] define a pattern as a set of malicious accesses; Amalfitano *et al.* [23] consider three types of patterns: GUI patterns, event patterns and model patterns; and Costa *et al.* [27] consider UI Test Patterns.

The patterns from Amalfitano *et al.* and from Costa *et al.* have some similarities with the ones presented in this paper. Amalfitano *et al.* define GUI and event patterns which represent the behaviour based on the GUI of the application and on Android system events, respectively. The UI Patterns presented in this paper can be related either to the GUI or to the system's events like detecting the presence of a side drawer or detecting if a certain resource is being used. Thus, the patterns from Amalfitano *et al.* are closely related to patterns defined in this paper (UI Patterns). The definition of Test Patterns used by Costa *et al.* is the same as the one used on this paper.

3. Approach

The iMPAcT tool automates the testing of recurring behaviour (UI patterns) present on Android mobile applications. Beyond the description provided in this paper it is possible to find more details of the iMPAcT tool in [14,33].

The testing approach supported by the iMPAcT tool is depicted in Figure 1 and has four main characteristics:

1. the goal is to test recurring behaviours, *i.e.*, UI patterns;
2. the whole process is completely automatic;
3. it is an iterative process combining automatic exploration, reverse engineering and testing;
4. the reverse engineering process is fully dynamic, *i.e.*, the source code is never accessed and no code instrumentation is required.

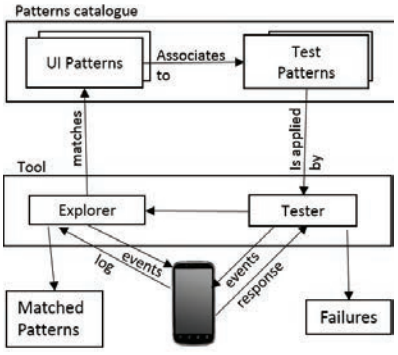

Fig. 1. Block Diagram of the Architecture of the Approach [34]

The approach follows an iterative process in which each iteration is composed of three phases:

1. Exploration: the screen is read, *i.e.*, the set of elements visible on the screen is identified along with their properties (clickable, enabled, position, ...) and the events that can be fired. From those events, one is randomly chosen and fired;
2. Pattern Matching: after an event is fired, the iMPAcT tool identifies the UI Patterns that are present on the screen. The UI Patterns to be identified are defined in a catalogue;
3. Tester: When a UI Pattern is matched, the corresponding test strategy (Test Pattern) in the catalogue is applied. Whenever a failure is found it is reported.

Identification of events

The decision of which event to execute is random. However, the set of events available for execution depends on the exploration mode set by the user:

1. *execute once*: only events that have not yet been executed are considered;
2. *prioritise not executed*: The choice is initially made among the events that have not yet been fired. However, it is possible to run an event a second time if all available events have already been fired and if this event is necessary to provide access to a screen that still has not executed events;
3. *prioritise list items and not executed*: this mode is similar to the previous one with two differences: events executed on elements belonging to a list have priority relatively to the remaining ones, and the "click on the App button" event is only selected when there are no more events to fire. The main advantage of this mode is that it tries to fire all possible events of each screen before going back, which is usually the result of clicking on the App button.

Stop Condition

When no event is available the *back button* is pressed in order to try to reach the previous screen. When this action takes the exploration to the home screen of the device, *i.e.*, exits the application, the exploration stops.

Patterns Catalogue

The catalogue of patterns contains the UI Patterns to be identified and the corresponding Test Patterns to test them. This catalogue is defined only once and may be reused for any Android mobile application. A Pattern (both UI and Test) is formally defined as a set of tuples <G, V, A, C, P>, in which:

G is the ID (goal) of the pattern;
V is a set of pairs [variable, value] relating input data with the variables involved;
A is the sequence of actions to perform;
C is the set of checks to perform;
P is the precondition (boolean expression) defining the conditions in which the pattern should be applied.

In order to apply a pattern, the tester needs to provide (configure) the value for each variable (in V). It is possible to use the same pattern several times with different configurations. As such, applying a pattern consists in: if the pre-condition (P) holds, a sequence of actions (A) is executed with the corresponding input values (V). In the end, a set of checks (C) is performed. When identifying a pattern (Pattern Matching), the preconditions result indicates if the tool should try to identify the pattern and the result of the checks indicates whether or not the pattern is present. When applying a Test Pattern (Tester), the preconditions result indicates if the corresponding test strategy should be applied and the result of the checks indicates whether or not the pattern is correctly implemented.

In this paper two patterns are used. The first one is the side drawer pattern: the UI Pattern detects the presence of a side drawer also know as navigation drawer, and the corresponding Test Pattern verifies if it takes up the full height of the screen. The second one is the Orientation Pattern: the UI Pattern verifies if it is possible to rotate the screen and the corresponding Test Pattern verifies 1) if rotating the screen does not make any widgets disappear and 2) if rotating the screen does not make any user input disappear. More details may be found in [34].

4. Case Study

As explained in Section 3, it is possible to select the exploration mode (1-execute once, 2-prioritise not executed, 3-prioritise list items and not executed) as well as to define in which order the patterns are identified and, thus,

tested. Altering these settings may change the results obtained: different failures may be detected and a different number of events may be executed. The percentage of executed events is calculated according to the number of events identified by the tool during the exploration. It is not possible to know the total number of possible events for a mobile application because some events may not be reached while exploring.

The goal of these experiments is to compare the results obtained by three different exploration modes and two different testing orders. Thus, during the experiment, we apply three different exploration modes combined with two different testing orders, *i.e.*, test the side drawer before orientation and vice-versa. This makes a total of six different executions modes for each application.

Tables 1 and 2 present the results obtained when exploring the Tomroid[3] and Book Catalogue[4] applications, respectively. Each of the six execution modes were run several times for each application since the results obtained depend on the events fired (which are chosen randomly). As such, the data presented in Tables 1 and 2 is an average of the different executions:

- orientation failures: average of failures detected in the orientation pattern. There is no column for the failures related to the side drawer pattern because there were no significant differences detected;
- execution time: average of the time taken in each execution;
- events identified: maximum number of events identified in the execution mode;
- % of events: average percentage of executed events over events identified in each execution;
- % of events over execution mode: the average percentage of executed events over the maximum number of events identified by executions of the same exploration mode and testing order;
- % of events over exploration mode: the average percentage of executed events over the maximum number of events identified by executions of the same exploration mode (regardless of testing order);
- % of events over all executions: the average percentage of executed events over the maximum number of events identified by all executions (regardless of testing order or exploration mode).

Table 1. Results obtained when applying different execution modes to Tomdroid

Testing Order	Orientation failures	Execution Time	Events identified	% of events	% events over execution mode	% events over exploration mode	% events over all executions
exploration mode 1: execute only once							
SD-O	1	2m 4s 410ms	76	52.7	33.6	33.6	24
O-SD	2.75	2m 8s 402ms	51	67.6	54	36.2	26
exploration mode 2: priority to not executed							
SD-O	1.75	5m 45s 466ms	106	85.6	63	63	63
O-SD	2.75	5m 50s 584ms	89	76.3	75	63	63
exploration mode 3: priority to not executed and list items							
SD-O	2	5m 39s 951ms	89	82.2	74.5	65.6	62.5
O-SD	2.5	5m 47s 346ms	101	76.8	61.1	61.1	59.15

By analysing the results obtained on the Tomdroid application present in Table 1, it is possible to state that:

- when the orientation pattern is tested before the side drawer pattern the number of failures detected increases;
- the exploration modes 2 and 3 take more time than exploration mode 1;
- the percentage of executed events during exploration mode 2 is higher than with exploration mode 1;
- the results in terms of time and % of events over exploration mode are very similar;
- all the results of the exploration mode 2 and exploration mode 3 are very similar *i.e.*, in the case of Tomdroid it is indifferent which of these modes is used.

The difference in the execution time between exploration mode 1 and exploration modes 2 and 3 is to be expected as the first one executes a much lower number of events (identifies less events and executes a smaller percentage of

[3] https://play.google.com/store/apps/details?id=org.tomdroid&hl=en
[4] https://play.google.com/store/apps/details?id=com.eleybourn.bookcatalogue&hl=en

the events). The difference in the overall results between mode 1 and modes 2 and 3 shows that the two last modes enable a more thorough exploration of the application, which leads to more failures being found.

Moreover, a more thorough analysis of the report makes it possible to conclude that, when testing the orientation pattern before testing the side drawer pattern, it detects the failure: "when rotating the screen after opening the *More Options* menu, one of the available options disappears"as depicted in Figure 2.

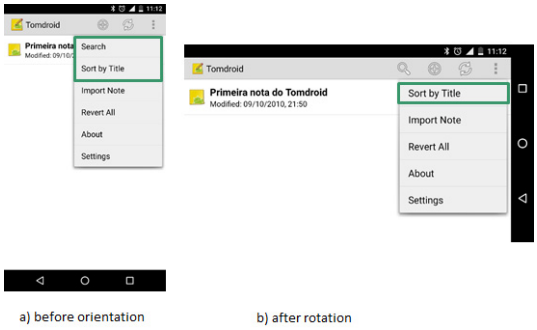

Fig. 2. The *More Options* menu of Tomdroid before and after rotation

Table 2. Results obtained when applying the different exploration modes and pattern orders to Book Catalogue

Testing Order	Orientation failures	Execution Time	Events identified	% of events	% events over execution mode	% events over exploration mode	% events over all executions
exploration mode 1: execute only once							
SD-O	5.75	4m 27s 362ms	141	45.6	25	25	5.4
O-SD	9	5m 15s 798ms	141	50.5	23.3	23.3	6.5
exploration mode 2: priority to not executed							
SD-O	13.25	13m 24s 297ms	372	54	34.3	34.3	19.5
O-SD	12.75	8m 32s 318ms	230	52.7	34.4	20.8	12.1
exploration mode 3: priority to not executed and list items							
SD-O	23.75	34m 1s 170ms	653	79.1	52.2	52.2	52.2
O-SD	33.25	32m 13s 346ms	418	83.6	76.1	48.7	48.7

By analysing the results obtained on the Book Catalogue application, it is possible to state that:

- the number of failures found in the orientation pattern is, generally, higher when testing the orientation first;
- the time of exploration does not vary with the testing order but varies with exploration modes. The exploration time of exploration 3 is seven times the amount of the time for exploration mode 1;
- the different percentages of executed events do not vary with the testing order but are affected by the exploration mode;
- the percentage of executed events (over events identified in each execution) is similar in the first two exploration modes (around 50%);
- exploration mode 3 identifies more events and presents higher percentages of executed events than the other modes.

Taking this into consideration it is possible to conclude that the exploration mode 3 provides a more thorough exploration as it identifies more events and still executes a higher percentage of events. It also detects more failures. However, it takes much longer to execute than the other exploration modes.

5. Conclusions

The iMPAcT tool provides an automatic way of exploring and testing Android applications. It relies on a catalogue of UI patterns and the corresponding test strategies (Test Patterns) that can be applied to any Android mobile application. However, the results depend both on the exploration mode and on the testing order. This paper presents a case study to assess the impact of these variations. With this case study it is possible to conclude that: 1) the testing order affects the number of failures detected but it does not significantly affect the number of events identified nor the percentage of events executed nor the exploration time;

2) the two latter exploration modes (2 and 3) provide a more thorough exploration and, thus, detect more events; 3) when dealing with bigger applications, exploration mode 3 enables a more thorough exploration than the other modes; 4) For smaller applications the difference between the modes 2 and 3 is not that significant; 5) the exploration mode affects the overall time and number of events identified but it does not necessarily affect the number of failures found.

The main advantage of this testing approach is the automation. However, it is only capable of testing the UI patterns within the pattern catalogue. Thus, it is important to improve such catalogue with additional patterns in the future. Moreover, we also intend to improve the visualisation of the results. At this moment the tool saves the fired events in a log file and shows the patterns and failures found.

Acknowledgements

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the FCT Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.

References

1. Ingraham, N.. Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio. 2013. URL http://goo.gl/z3RprB.
2. Alalfi, M.H., Cordy, J.R., Dean, T.R.. Automating Coverage Metrics for Dynamic Web Applications. In: *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE. ISBN 978-1-61284-369-8; 2010, p. 51–60. doi:10.1109/CSMR.2010.21. URL http://goo.gl/WkQ7Y0.
3. Arlt, S., Bertolini, C., Schäf, M.. Behind the Scenes: An Approach to Incorporate Context in GUI Test Case Generation. In: *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2011)*. Washington, DC, USA; 2011, p. 222–231.
4. Moreira, R.M.L.M., Paiva, A.C.R.. PBGT tool: an integrated modeling and testing environment for pattern-based GUI testing. In: *29th ACM/IEEE international conference on Automated software engineering (ASE 2014)*. New York, New York, USA: ACM Press. ISBN 9781450330138; 2014, p. 863–866. doi:10.1145/2642937.2648618. URL http://dl.acm.org/citation.cfm?id=2642937.2648618.
5. Nabuco, M., Paiva, A.C.R.. Model-Based Test Case Generation for Web Applications. In: *14th International Conference on Computational Science and Applications (ICCSA 2014)*. 2014, .
6. Aho, P., Suarez, M., Memon, A., Kanstren, T.. Making GUI Testing Practical: Bridging the Gaps. In: *12th International Conference on Information Technology - New Generations (ITNG 2015)*. Las Vegas, NV, USA. ISBN 978-1-4799-8828-0; 2015, p. 439–444. doi:10.1109/ITNG.2015.77. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7113512.
7. Amalfitano, D., Fasolino, A.R., Tramontana, P., De Carmine, S., Memon, A.M.. Using GUI ripping for automated testing of Android applications. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. New York, New York, USA: ACM Press. ISBN 9781450312042; 2012, p. 258–261. doi:10.1145/2351676.2351717. URL http://dl.acm.org/citation.cfm?id=2351676.2351717.
8. Muccini, H., di Francesco, A., Esposito, P.. Software testing of mobile applications: Challenges and future research directions. In: *7th International Workshop on Automation of Software Test (AST 2012)*. Zurich, Switzerland: IEEE. ISBN 9781467318228; 2012, p. 29–35. doi:10.1109/IWAST.2012.6228987. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6228987.
9. Capgemini, , Sogeti, , Hp, . World Quality Report 2014-15. Tech. Rep.; 2014. URL http://goo.gl/jzN2aA.
10. Chikofsky, E., Cross, J.. Reverse Engineering and Design Recovery: a Taxonomy. *IEEE Software* 1990;**7**(1):13–17. doi:10.1109/52.43044. URL http://dx.doi.org/10.1109/52.43044.
11. Nilsson, E.G.. Design patterns for user interface for mobile applications. *Advances in Engineering Software* 2009;**40**(12):1318–1328. doi:10.1016/j.advengsoft.2009.01.017. URL http://www.sciencedirect.com/science/article/pii/S0965997809000428.
12. Sahami Shirazi, A., Henze, N., Schmidt, A., Goldberg, R., Schmidt, B., Schmauder, H.. Insights into layout patterns of mobile user interfaces by an automatic analysis of android apps. In: *5th ACM SIGCHI symposium on Engineer-*

ing interactive computing systems. ACM. ISBN 978-1-4503-2138-9; 2013, p. 275–284. doi:10.1145/2480296.2480308. URL http://dl.acm.org/citation.cfm?id=2494603.2480308.
13. Neil, T.. *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps*. Sebastopol, Canada: O'Reilly Media, Inc.; 2nd ed.; 2014.
14. Coimbra Morgado, I., Paiva, A.C.R.. The iMPAcT Tool: Testing UI Patterns on Mobile Applications. In: *30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*. 2015, .
15. Rohatgi, A., Hamou-Lhadj, A., Rilling, J.. An Approach for Mapping Features to Code Based on Static and Dynamic Analysis. In: *2008 16th IEEE International Conference on Program Comprehension*. IEEE; 2008, p. 236–241. doi:10.1109/ICPC.2008.35. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4556137.
16. Grilo, A.M.P., Paiva, A.C.R., Faria, J.P.. Reverse engineering of GUI models for testing. In: *The 5th Iberian Conference on Information Systems and Technologies (CISTI '10)*; July. IEEE; 2010, p. 1–6. URL http://goo.gl/bXcIy.
17. Coimbra Morgado, I., Paiva, A.C.R., Pascoal Faria, J.. Reverse Engineering of Graphical User Interfaces. In: *The Sixth International Conference on Software Engineering Advances (ICSEA '11)*; c. Barcelona. ISBN 9781612081656; 2011, p. 293–298.
18. Sacramento, C., Paiva, A.C.R.. Web Application Model Generation through Reverse Engineering and UI Pattern Inferring. In: *9th International Conference on the Quality of Information and Communications Technology (QUATIC 2014)*. Guimarães, Portugal: IEEE. ISBN 978-1-4799-6133-7; 2014, p. 105–115. doi:10.1109/QUATIC.2014.20. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6984098.
19. Marchetto, A., Tonella, P., Ricca, F.. Under and Over Approximation of State Models Recovered for Ajax Applications. In: *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE; 2010, p. 236–239. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5714441.
20. Maezawa, Y., Washizaki, H., Honiden, S.. Extracting Interaction-Based Stateful Behavior in Rich Internet Applications. In: *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE; 2012, p. 423–428. doi:10.1109/CSMR.2012.53. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6178915.
21. Batyuk, L., Herpich, M., Camtepe, S.A., Raddatz, K., Schmidt, A.D., Albayrak, S.. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. In: *2011 6th International Conference on Malicious and Unwanted Software*. IEEE. ISBN 978-1-4673-0034-6; 2011, p. 66–72. doi:10.1109/MALWARE.2011.6112328. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6112328.
22. Yang, W., Prasad, M.R., Xie, T.. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications. In: *16th International Conference on Fundamental Approaches to Software Engineering (FASE'13)*. Rome, Italy; 2013, p. 250–265. doi:10.1007/978-3-642-37057-1_19. URL http://link.springer.com/chapter/10.1007/978-3-642-37057-1_19.
23. Amalfitano, D., Fasolino, A.R., Tramontana, P., Amatucci, N.. Considering Context Events in Event-Based Testing of Mobile Applications. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. IEEE. ISBN 978-0-7695-4993-4; 2013, p. 126–133. doi:10.1109/ICSTW.2013.22. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6571621.
24. Gao, J., Bai, X., Tsai, W.T., Uehara, T.. Mobile application testing - a Tutorial. *Computer* 2014;**47**(2):46–55. doi:10.1109/MC.2013.445.
25. Amalfitano, D., Fasolino, A.R., Tramontana, P., De Carmine, S., Imparato, G.. A toolset for GUI testing of Android applications. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. ISBN 978-1-4673-2312-3; 2012, p. 650–653. doi:10.1109/ICSM.2012.6405345. URL http://goo.gl/gPDU9z.
26. Machiry, A., Tahiliani, R., Naik, M.. Dynodroid: an input generation system for Android apps. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*. New York, New York, USA: ACM Press. ISBN 9781450322379; 2013, p. 224. doi:10.1145/2491411.2491450. URL http://dl.acm.org/citation.cfm?id=2491411.2491450.
27. Costa, P., Paiva, A.C.R., Nabuco, M.. Pattern Based GUI Testing for Mobile Applications. In: *9th International Conference on the Quality of Information and Communications Technology (QUATIC 2014)*. Guimarães, Portugal: IEEE. ISBN 978-1-4799-6133-7; 2014, p. 66–74. doi:10.1109/QUATIC.2014.16. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6984094.
28. Nasim, F., Aslam, B., Ahmed, W., Naeem, T.. Uncovering Self Code Modification in Android. In: *First International Conference on Codes, Cryptology, and Information Security, C2SI 2015*. Rabat, Morocco: Springer; 2015, p. 297–313. doi:10.1007/978-3-319-18681-8_24. URL http://link.springer.com/chapter/10.1007/978-3-319-18681-8_24.
29. Cunha, M., Paiva, A.C.R., Ferreira, H.S., Abreu, R.. PETTool: A pattern-based GUI testing tool. In: *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*; vol. 1. San Juan, PR: IEEE. ISBN 9781424486663; 2010, p. V1–202 – VI–206. doi:10.1109/ICSTE.2010.5608882. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5608882.
30. Moreira, R.M.L.M., Paiva, A.C.R., Memon, A.. A pattern-based approach for GUI modeling and testing. In: *24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)*. Passadena, CA: IEEE. ISBN 978-1-4799-2366-3; 2013, p. 288–297. doi:10.1109/ISSRE.2013.6698881. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6698881.
31. Monteiro, T., Paiva, A.C.R.. Pattern Based GUI Testing Modeling Environment. In: *Sixth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013)*. Luxembourg, Luxembourg: IEEE. ISBN 978-0-7695-4993-4; 2013, p. 140–143. doi:10.1109/ICSTW.2013.24. URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6571623.
32. Hu, C., Neamtiu, I.. Automating gui testing for android applications. In: *The 6th International Workshop on Automation of software test (AST '11)*. New York, New York, USA: ACM Press. ISBN 9781450305921; 2011, p. 7. doi:10.1145/1982595.1982612. URL http://dl.acm.org/citation.cfm?id=1982595.1982612.
33. Coimbra Morgado, I., Paiva, A.C.R.. Testing approach for mobile applications through reverse engineering of UI patterns. In: *Sixth International Workshop on Testing Techniques for Event BasED Software*. 2015, .
34. Coimbra Morgado, I., Paiva, A.C., Faria, J.P.. Automated Pattern-Based Testing of Mobile Applications. In: *2014 9th International Conference on the Quality of Information and Communications Technology*. Guimarães, Portugal: IEEE. ISBN 978-1-4799-6133-7; 2014, p. 294–299. doi:10.1109/QUATIC.2014.47.