2014 9th International Conference on the Quality of Information and Communications Technology

Automated Pattern-Based Testing of Mobile Applications

Inês Coimbra Morgado, Ana C. R. Paiva, and João Pascoal Faria
INESC-TEC
Department of Informatics Engineering, Faculty of Engineering, University of Porto,
rua Dr. Roberto Frias, 4200-465 Porto, Portugal

***Abstract*—This paper presents an approach for testing mobile applications using reverse engineering and behavioural patterns. The goal of this research work is to ease the testing of mobile applications by automatically identifying and testing behaviour that is common in this type of applications, *i.e.*, behaviour patterns. The approach includes a tool to automatically explore an Android application. This tool also identifies patterns in the behaviour of the application and apply tests previously associated with those patterns. The final results of this research work will be a catalogue of behavioural patterns and the tool which will output a report on the matched patterns and another one on the testing of those patterns.**

***Keywords*—Reverse engineering; Testing; Android; Patterns**

I. Introduction

Since the release of the iPhone in 2007 [1] and of the first Android smart phone in 2008 [2], smart phones have started to greatly increase their mobile sales. In fact, in 2013 both Android's *Google Play* and Apple's *App Store* surpassed the one million available applications and fifty billion downloads threshold [3]. This market dimension makes it extremely important to ensure the quality of an application as it generates a high level of competitiveness and thus for one to get popular it must be as flawless as possible. Furthermore, there has also been an increase of business critical mobile applications, such as mobile banking applications, which makes it even more important to ensure its functional correctness.

Mobile applications have, as any other type of application, their own quirks regarding testing, such as the high amount of different events that need to be tested. An event in a mobile application can be of two types: an user interface (UI) event, *i.e.*, an event provoked by an interaction of the user with the application, such as tapping a widget; or a system event, *i.e.*, an event provoked by something external to the application, such as the detection of a new available network, an incoming call or message and the modification of the orientation of the phone (from vertical to horizontal, for instance). Each of these events may or may not have an impact on the application behaviour. For instance, an incoming call changes the currently *running* activity to the *stopped* state and tapping a widget may modify the state of data of the currently *running* activity or even close it to open a new one.

One of the main focus of mobile testing is user interface (UI) testing, more precisely Graphical UI (GUI) testing, as this is the source of interaction with the user and where most errors occur. Two ways of automating testing are automating the test case generation process or automating the test case execution. One of the most popular techniques for automatic test case generation, which is the focus of this work, is Model-based Testing (MBT) [4], *i.e.*, a model of the application's behaviour is necessary as input and the output is a test suite. This technique has two main problems: the effort required in building the model as it is usually not available and the combinatorial explosion of test cases that are generated [5]. The first problem will be addressed using reverse engineering techniques and the second by focusing on testing recurring behaviour, *i.e.*, behaviour patterns. A pattern is a recurring solution for a recurring problem in a certain context. A behavioural pattern reproduces a recurring behaviour situation, for instance, login or master/detail. Examples of behavioural patterns in Android applications are screen rotation or the starting of a new activity.

Software reverse engineering was defined in 1990 by Chikofsky and Cross [6] as "the process of analysing a subject system to (1) identify the system's components and interrelationships and (2) to create representations of the system in another form or at a higher level of abstraction".

Even though nowadays reverse engineering is considered helpful in several areas [7], such as testing, it initially surfaced associated with software maintenance as it eases system comprehension. This was considered extremely important as over 50% of a systems development is occupied with maintenance tasks [8], [9], [10] and over 50% of maintenance is dedicated to comprehending the system [11], [12]. Reverse engineering has also proved to be useful, for instance, in coping with the Y2K problem, with the European currency conversion and with the migration of information systems to the web and towards the electronic commerce [13]. With the exploration of reverse engineering techniques, its usefulness grew from software maintenance to other fields, such as verification and validation and security analysis.

As far as we know, none of the reverse engineering approaches applied on mobile applications tries to take advantage of the existence of behavioral patterns on the application to facilitate their task. Mobile applications present behavioural patterns which can ease the modelling and, thus, the testing task. As such, this approach explores a mobile application in order to identify behaviour patterns and to test these patterns. During the exploration, whenever an occurance of a behavioural pattern is detected a pre-defined test strategy will be applied to test it. Even though it is necessary to manually specify a catalogue of behavioural patterns and the corresponding test strategies, the catalogue is common to every application and so this effort is only necessary once. This will

978-1-4799-6133-7/14 $31.00 © 2014 IEEE
DOI 10.1109/QUATIC.2014.47

be one of the results of this work.

There have already been some studies that show that patterns can be useful for testing mobile applications. In 2009, Erik Nilsson [14] identified some recurring problems when developing an Android application and the UI design patterns that could help solve them. If these patterns have an associated behaviour then it is possible to identify the pattern by the automatic detection of the behaviour. In 2013, Sahami Shirazi *et al.* studied the layout of Android applications trying, among other goals, to verify if these layouts presented any patterns. They concluded that 75.8% of unique combinations of elements appeared only once in the application. Nevertheless, this study was conducted taking into consideration a static analysis of the layout and its elements while different combination of elements may represent the same behaviour and, thus, the same pattern.

The remaining of this document is structured as follows. Section II presents the state of the art on reverse engineering and patterns applied to mobile applications. Section III presents the approach and research methodology. Section IV presents the past work and preliminary results. Section V presents the work to be developed along with the results that are expected. Section VI presents the drawn conclusions.

II. State-of-the-art

This Section presents a research on the state of the art on mobile reverse engineering. In order to ease the analysis of the approaches, an ontology based on the one presented by Cornelissen *et al.* [15] was defined with the top-level concepts:

- *goal*: what is the main purpose of the approach? This corresponds to *activity* from Cornelissen *et al.*;
- *target*: what is the target platform in which the approach works? This corresponds to *target* from Cornelissen *et al.*;
- *method*: does it use a static, dynamic, or hybrid method? Cornelissen *et al.* only consider dynamic approaches so this aspect was not considered;
- *technique*: what are the techniques applied? This corresponds to *method* from Cornelissen *et al.*;
- *extracted information*: what type of information is extracted? This aspect did not appear in the Cornelissen *et al.*'s ontology;
- *output*: how is the obtained information represented to the user? This aspect did not appear in the Cornelissen *et al.*'s ontology;
- *validation*: how is the proposed approach validated? This corresponds to *Evaluation* from Cornelissen *et al.*.

Table I presents the result of the classification of the approaches according to the ontology. Further details on the definition of this ontology can be found in [16].

Goal

Joorabchi *et al.* [22] and Yang *et al.* [23] are the only ones who claim their goal is to obtain a model of the application's user interface. Naturally other approaches also have this purpose, like Amalfitano *et al.*'s [18], but Joorabchi *et al.* and Yang *et al.* make no effort of trying to test the application or to generate event sequences. Their main goal is to obtain a finite state machine representing the behaviour of the application. According to Yang *et al.* [23] the main difference between their approach and Joorabchi *et al.*'s [22] is that the latter has no means of identifying which GUI elements are actionable and which events these elements support.

Target

Joorabchi *et al.* [22] are the only ones who target iOS applications, while the remaining approaches target Android applications.

Method

Table I shows a clear predominance of dynamic and hybrid approaches, which is to be expected given the event-based nature of mobile applications. In fact, only Batyuk *et al.* [20] follow a static approach, trying to identify possible security vulnerabilities, such as unwanted access of user data.

Technique

All the dynamic approaches apply instrumentation, even if it is to the Android SDK framework [27], like Machiry *et al.* [26], or to the virtual machine where the application is being run, like Hu *et al.* [25]. Unlike their work in [28], in which Amalfitano *et al.* targeted rich internet applications, in [17] and [18] they apply instrumentation to the application under analysis itself. Anand *et al.* [24] apply instrumentation both to the application under analysis and to the Android SDK framework. It is also verifiable that most approaches, specially the most recent ones, opt for an hybrid approach in order to benefit from both worlds. Amalfitano *et al.*'s work in 2011 [17] aimed at automatically generating test cases and detecting crashes by crawling the application. In 2011 Amalfitano *et al.* [17] used the instrumentation to obtain analysable logs in order to detect the origin of a crash. In 2012 [18], with the same idea in mind, they applied Memon's GUI Ripper [29], [30] to Android applications, which rips the application and then analyses each part separately simulating user events. The main difference in the results was their new capability of detecting some bugs besides crashes. In 2013 [19] they decided to add an analysis of the behaviour of the application in the presence of system events. To do so they opted again to crawl the application and replaced the Android Sensor framework with an *ad hoc* version to ease the task of injecting this new type of events.

Extracted Information

Both Anand *et al.* [24] and Jensen *et al.* [21] obtain event sequences to test the application with a hybrid approach. However, Anand *et al.* aim at obtaining a set of event sequences with the higher coverage percentage possible and Jensen *et al.*'s idea is to find a feasible path that enables the testing of a given code statement that has not yet been reached by other testing techniques, *i.e.*, Jensen *et al.*'s approach complements other approaches which do not present 100% coverage.

Output

As most approaches focus on verifying and validating applications it would be expected to find a tendency of the

TABLE I. CLASSIFICATION OF THE MOBILE REVERSE ENGINEERING APPROACHES ACCORDING TO THE ONTOLOGY

		Papers									
Aspect	Classification	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]
Goal	V&V	x	x	x	x	x			x	x	x
	Model Recovery						x	x			
Target	Android	x	x	x	x	x		x	x	x	x
	iOS						x				
Method	Static				x						
	Dynamic	x	x							x	x
	Hybrid			x		x	x	x	x		
Technique	Crawling	x		x			x	x			x
	Ripping		x								
	Instrumentation	x	x	x					x	x	x
	Parsing							x			
	Event Handling					x				x	
	Event Simulation	x	x	x			x	x		x	x
	Pattern Identification				x					x	
	Clustering	x									
	Test Generation	x	x	x						x	
	Code Injection						x				
	Code Replacement			x			x				
	Reflection			x			x				
	Comparison of Interface	x					x				
	Data Mining				x						
	Concolic Execution					x			x		
Extracted Information	Crash Detection	x	x	x							
	Bugs (not only crashes)		x	x						x	
	Event Sequence					x			x		
	Runtime Behaviour						x	x			
	Inputs										x
	Malicious Functionalities				x						
Output	Test Suit	x	x	x							
	Call Graph					x			x		
	Finite State Machine						x	x			
	Report				x					x	x
Validation	Case Study	x	x	x	x	x	x	x	x	x	x
	Comparison With Other Approaches/Tools		x	x				x		x	x
	Evaluation		x	x				x			x

type of output towards test suites. However, this does not happen. In fact, there is no tendency towards any of the types. Nevertheless, it is important to note the model recovery approaches opt by modelling their applications with a finite state machine.

Validation

All approaches do case studies with at least one application, half of them compare their results with the ones from other approaches [23], existing tools [18], [26], with data obtained by manual analysis [25], [26] or even their own previous work [19] and almost half the approaches do some kind of evaluation, with coverage analysis begin the most popular.

There are only two mobile reverse engineering approaches dealing with patterns: Amalfitano *et al.* [19] and Batyuk *et al.* [20]. Amalfitano *et al.* define a set of event sequences to test situations like an incoming call. The term pattern is used because these event sequences can be applied to any application. Batyuk *et al.* [20] apply pattern identification to detect malicious intents of the application. However, none of these approaches try to take advantage of the existence of behavioural patterns in the application to facilitate their task, which is one of the goals of the approach described in this document. In fact, none of the software reverse engineering approaches who aimed at pattern identification focus on behavioural patterns.

Moreover, only Amalfitano *et al.* [19] and Machiry *et al.* [26] consider the effect of system events. However, Amalfitano *et al.*'s goal is to generate a test suite and to detect crashes and Machiry *et al.* attempt at recovering input data to enable testing, while the approach in this document will output a report on matched patterns and where they are found as well as a testing report.

A. Conclusions

The approach here presented aims at using reverse engineering to ease the behavioural patterns identification process and to use these patterns to ease the automation of the testing process. This will enable the testing of part of the application without the effort of building a model and without the risk of having a test case explosion as the test case generation will only be focused on the patterns found during the exploration and each pattern has a well-defined test strategy associated with it.

III. RESEARCH OBJECTIVES AND METHODOLOGICAL APPROACH

This Section presents the research objectives and the methodological approach as well as the research hypothesis and the how it will be validated.

A. Research Objectives

The main research objectives of this work are to construct a catalogue of behavioural patterns and to develop a tool that automatically explores a mobile application and identifies and tests behavioural patterns during the exploration.

B. Methodological Approach

The main steps of this approach consist in: 1) defining a catalogue of mobile GUI patterns to identify and the corresponding test strategy; 2) applying a hybrid reverse engineering approach to automatically explore mobile applications; 3) identifying patterns on the fly and applying the predefined test; 4) storing the information regarding all the explored behaviour; 5) producing a test report and a matched patterns report.

The reverse engineering process to be applied is based on Yang *et al.*'s approach [23], *i.e.*, static analysis will be used to identify which event handlers are associated with each widget in order to dynamically exercise them. A deep study on which frameworks or tools are best is still being undertaken. A possible static analysis tool is WALA [31], which is the one used by Yang *et al.* [23] and extracts a call graph of the application, and possible solutions for the dynamic exploration and exercising of widgets are Robotium [32] and Monkey Runner [33].

Finally, after each step of the exploration in which a pattern (from the previously defined catalogue) is identified, the corresponding test strategy is applied and a report is produced. The testing phase consists in injecting the events defined in the test strategy associated with the behavioural pattern matched and in analysing the responses of the system, comparing them with the ones defined in the test strategy. Figure 1 depicts the approach.

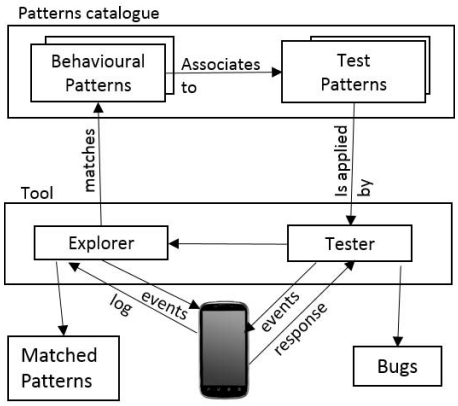

Fig. 1. Block Diagram of the Architecture of the Approach

C. Research Hypothesis

Mobile applications have generic recurrent behaviour, independent of their specific domain, that may be tested automatically by combining reverse engineering with testing within an iterative process.

D. Validation

The validation of the approach is divided in three main aspects. Firstly, as the goal of the approach is to test the application, some errors will be introduced in a set of applications in order to verify if the approach detects them. Secondly, an analysis of the percentage of the code explored and tested will be undertaken. Finally, the results will be compared with the ones from other approaches, such as the ones of Yang *et al.* [23] and Amalfitano *et al.* [19]. This comparison will mainly be on the reverse engineering results.

In order to do so, a case study will be conducted preferably on applications also used to validate other mobile reverse engineering approaches. Thus, the implementation of the approach will focus Android smartphones applications.

IV. Past work and preliminary results

A. Catalogue of Behavioural Patterns

One of the most important aspects of this approach is the definition of the patterns catalogue. Even though it is always possible to improve the catalogue, at this moment some behaviour patterns have already been identified. These are some examples:

- screen rotation - when the screen is rotated, the information on the screen should remain unaltered even thoug its placement may change
- menu appearance - the information on the screen should remain unaltered except for the new menu (the menu itself also provides new exploration and test paths);
- incoming call - when the call ends the application should go back to the same state it was in upon the incoming call;
- appearance of a keyboard - when pressing the keys, the content of the selected text box should change accordingly.

B. GUI Reverse Engineering for Visual and Formal Models

Previous work on reverse engineering has already been conducted in Desktop applications. In [34], [35] an approach for recovering part of a formal model of a Desktop application was presented. This approach automatically explored the application using the Microsoft Windows accessibility framework UI automation [36] and following a depth-first exploration algorithm. It was able to successfully extract GraphML models on the navigation and on properties changes of the application, as well as part of a formal Spec# model.

C. Patterns to disambiguate extracted models

Often the behaviour of applications is modelled as a finite state machine (FSM), which may arise an ambiguity problem, *i.e.*, when from the same state, the same event leads to two different states.

In [35], a machine learning technique called Inductive Logic Programming [37], [38] was used in order to solve these ambiguities. The main idea was to identify ambiguities in the

model and match them to previously defined patterns which indicate how to solve that same ambiguity and would modify the model accordingly. In order to apply ILP, all states, transitions and patterns were expressed in a declarative language, Prolog [39]. Even though the Prolog code representing the states and the transitions can be automatically derived from the model, the patterns, alike every pattern identification approach, have to be manually defined. Nevertheless, they are reusable. The approach followed is summarised in Figure 2.

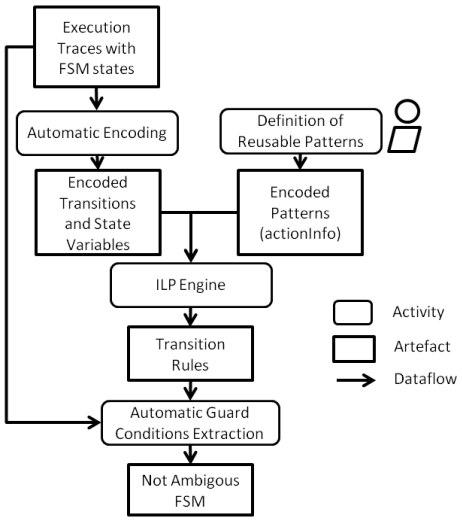

Fig. 2. UML Activity Diagram of the Architecture of the ILP approach to disambiguate FSM models

This experiment shows a succesfull application of a pattern-matching technique, which could be used in the work presented in this document. However, due to scalability issues, other options are being analysed.

V. Future work and expected results

It is possible to divide the future work in near future and otherwise. For the near future, the most important steps to take are: 1) explore and decide the best framework to ease the application exploration and data collection; 2) select applications which are known to have at least one of the patterns; 3) manually explore the application in order to ease the development of the tool and to test the identification of that pattern; 4) apply the test pattern when the behaviour pattern is found.

At this point, it is expected to have a tool that extracts information along a manual exploration of an application. This manual exploration is stopped when a pattern is found and the test pattern is applied. The tool will produce a report whenever the pattern is found stating the result of the test.

Once these steps are concluded, it is important to: 1) test with other applications adding errors on the patterns to fully test the approach; 2) develop the automatic exploration of the application; 3) identify other patterns.

At the end of the development, the tool is expected to automatically explore the application and to identify several behaviour patterns. The case study to be conducted should verify that the tool identifies the patterns, testing them and producing the correct report.

VI. Conclusions

This document presents the current state of the art on reverse engineering and, more specifically, on mobile reverse engineering. Furthermore, it describes the approach to be followed in this PhD work as well as the research methodology associated with it. The approach described automatically explores and tests a mobile application. The testing process will be based on the identification of behavioural patterns on its GUI. In order to achieve this, a reverse engineering approach will be followed. The process will be based on an automatic and dynamic exploration of the applications GUI. However, in order to ease this exploration, a static analysis will identify the widgets that can be exercised and how they can be exercised, *i.e.*, it identifies the widgets which have event handlers associated. Furthermore, behavioural patterns are to be identified in order to enable the testing of the corresponding behaviour on the fly. In summary, the contributions of the work will be: 1) a reverse engineering approach to identify occurrences of behavioral patterns in mobile applications, 2) an on-the-fly testing approach based on the application of test patterns associated with behavioral patterns identified in the mobile application. In this approach the effort is associated with the assembly of the patterns catalogue, which is common to every application, *i.e.*, once defined it can simply be reused. As such, it has a better cost-benefit ratio than MBT approaches and it enables the detection of more (and different) errors than monkey testing tools.

Acknowledgments

This work is financed by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within the project FCOMP-01-0124-FEDER-020554 and the PhD scholarship SFRH/BD/81075/2011.

References

[1] CrunchBase, "iPhone," Jan. 2014. [Online]. Available: http://www.crunchbase.com/product/iphone

[2] M. Wilson, "T-Mobile G1: Full Details of the HTC Dream Android Phone," Jan. 2014. [Online]. Available: http://goo.gl/6vqI4E

[3] N. Ingraham, "Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio," Dec. 2013. [Online]. Available: http://goo.gl/z3RprB

[4] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Nov. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1200168

[5] A. C. R. Paiva and R. M. Vidal, "Automated specification-based testing of graphical user interfaces," Ph.D. dissertation, Faculdade de Engenharia da Universidade do Porto, 2006. [Online]. Available: http://sigarra.up.pt/feup/pt/publs_pesquisa.FormView?P_ID=23628

[6] E. Chikofsky and J. Cross, "Reverse Engineering and Design Recovery: a Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990. [Online]. Available: http://dx.doi.org/10.1109/52.43044

[7] G. Canfora and M. Di Penta, "New Frontiers of Reverse Engineering," in *Future of Software Engineering*, Minneapolis, 2007, pp. 326 – 341.

[8] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," *Communications of the ACM*, vol. 21, no. 6, pp. 466–471, Jun. 1978. [Online]. Available: http://dl.acm.org/citation.cfm?id=359511.359522

[9] I. Sommerville, *Software Engineering*, 5th ed. Addison-Wesley, 1995.

[10] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=612986.613032

[11] T. A. Standish, "An Essay on Software Reuse," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 494–497, Sep. 1984. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5010272

[12] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, Jun. 1989. [Online]. Available: http://dl.acm.org/citation.cfm?id=97118.97124

[13] H. A. Muller, J. H. Jahnke, D. B. Smith, and M.-A. Storey, "Reverse engineering: a roadmap," in *Proceedings of the conference on The future of Software engineering - ICSE '00*. New York, New York, USA: ACM Press, May 2000, pp. 47–60. [Online]. Available: http://dl.acm.org/citation.cfm?id=336512.336526

[14] E. G. Nilsson, "Design patterns for user interface for mobile applications," *Advances in Engineering Software*, vol. 40, no. 12, pp. 1318–1328, Dec. 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997809000428

[15] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, Sep. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4815280

[16] I. Coimbra Morgado, "Automated Pattern-Based Testing of Mobile Applications [Thesis Proposal]," Faculdade Engenharia Universidade Porto, Tech. Rep. April, 2014. [Online]. Available: http://paginas.fe.up.pt/%7epro11016/files/PhDThesisProposal.pdf

[17] D. Amalfitano, A. Fasolino, and P. Tramontana, "A GUI Crawling-Based Technique for Android Mobile Application Testing," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 252–261. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5954416

[18] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "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, Sep. 2012, p. 258. [Online]. Available: http://dl.acm.org/citation.cfm?id=2351676.2351717

[19] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci, "Considering Context Events in Event-Based Testing of Mobile Applications," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, Mar. 2013, pp. 126–133. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6571621

[20] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A.-D. Schmidt, and S. Albayrak, "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, Oct. 2011, pp. 66–72. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6112328

[21] C. S. Jensen, M. R. Prasad, and A. Mø ller, "Automated testing with targeted event sequence generation," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis - ISSTA 2013*. New York, New York, USA: ACM Press, Jul. 2013, p. 67. [Online]. Available: http://dl.acm.org/citation.cfm?id=2483760.2483777

[22] M. E. Joorabchi and A. Mesbah, "Reverse Engineering iOS Mobile Applications," in *2012 19th Working Conference on Reverse Engineering*. IEEE, Oct. 2012, pp. 177–186. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6385113

[23] W. Yang, M. R. Prasad, and T. Xie, "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, pp. 250–265. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-37057-1_19

[24] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*. New York, New York, USA: ACM Press, Nov. 2012, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2393596.2393666

[25] C. Hu and I. Neamtiu, "Automated GUI Testing on the Android Platform," in *The 22nd International Conference on Testing Software and Systems (ICTSS '10)*. Natal, Brazil: ACM Press, May 2010, pp. 67–72. [Online]. Available: http://dl.acm.org/citation.cfm?id=1982595.1982612

[26] A. Machiry, R. Tahiliani, and M. Naik, "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, Aug. 2013, p. 224. [Online]. Available: http://dl.acm.org/citation.cfm?id=2491411.2491450

[27] G. Android, "Get the Android SDK," 2014. [Online]. Available: http://developer.android.com/sdk/index.html

[28] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "Reverse Engineering Finite State Machines from Rich Internet Applications," in *The 15th Working Conference on Reverse Engineering (WCRE '08)*. IEEE, Oct. 2008, pp. 69–73. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656395

[29] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing," in *The 10th Working Conference on Reverse Engineering (WCRE '03)*, 2003. [Online]. Available: http://www.cs.umd.edu/ atif/papers/MemonWCRE2003.pdf

[30] D. R. Hackner and A. M. Memon, "Test case generator for GUITAR," in *Companion of the 13th international conference on Software engineering (ICSE Companion '08)*, ser. ICSE Companion '08. New York, New York, USA: ACM Press, 2008, p. 959. [Online]. Available: http://dl.acm.org/citation.cfm?id=1370175.1370207 http://doi.acm.org/10.1145/1370175.1370207

[31] T. J. Watsoon, "Wala," Jan. 2014. [Online]. Available: http://wala.sourceforge.net/wiki/index.php/Main_Page

[32] Google, "robotium," Jan. 2014. [Online]. Available: https://code.google.com/p/robotium/

[33] G. Android, "Monkey Runner," Jan. 2014. [Online]. Available: http://developer.android.com/tools/help/monkeyrunner_concepts.html

[34] I. Coimbra Morgado, A. C. R. Paiva, and J. a. Pascoal Faria, "Reverse Engineering of Graphical User Interfaces," in *The Sixth International Conference on Software Engineering Advances (ICSEA '11)*, no. c, Barcelona, 2011, pp. 293–298.

[35] I. Coimbra Morgado, A. C. R. Paiva, J. Pascoal Faria, and R. Camacho, "GUI Reverse Engineering with Machine Learning," in *Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'12)*, Zurich, Switzerland, 2012, pp. 27–31.

[36] R. Haverty, "New accessibility model for Microsoft Windows and cross platform development," *SIGACCESS Access. Comput.*, no. 82, pp. 11–17, 2005. [Online]. Available: http://doi.acm.org/10.1145/1077238.1077240

[37] S. Muggleton, "Inductive logic programming," in *Proceedings of the 1st Conference on Algorithmic Learning Theory*, 1990, pp. 43–62.

[38] S. H. Muggleton and L. D. Raedt, "Inductive Logic Programming: Theory and Methods," *Journal of Logic Programming*, vol. 19,20, pp. 629–679, 1994.

[39] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, 4th ed. Berlin, Germany: Springer-Verlag New York Berlin Heidelberg, 2003.