Automated Testing of Mobile Applications: A Systematic Map and Review

Abel Méndez-Porras[1], Christian Quesada-López[2], and Marcelo Jenkins[2]

Department of Computer Science, Costa Rica Institute of Technology[1]
Center for ICT Research, University of Costa Rica, San Pedro, Costa Rica[2]
amendez@itcr.ac.cr, cristian.quesadalopez@ucr.ac.cr, marcelo.jenkins@ecci.ucr.ac.cr

Abstract. Context: Software testing is a costly and time-consuming activity. Automated testing of mobile applications is considered complex and difficult. Indeed, several factors such as a variety of inputs (user, context, and environment) that a mobile application normally requires, and the heterogeneity of the technologies make automated testing not a trivial task. Two of the major challenges for automated testing are creation of the appropriate test cases scenarios and to decide in which devices to perform the testing. **Objective:** This paper reports on a systematic map and review. Automated testing approaches for mobile applications, testing techniques, and empirical assessment are identified, mapped, and characterized. We investigate the major challenges in automated testing of mobile applications. An analysis and synthesis of these studies is conducted. **Method:** A systematic mapping and systematic literature review research method has been conducted for identifying and aggregating evidence about automated testing of mobile applications. **Results:** A total 83 studies were identified. The results were tabulated and synthesized to provide recommendations to practitioners about automated testing of mobile applications. The main approaches identified were model-based testing (30%), capture/replay (15.5%), model-learning testing (10%), systematic testing (7.5%), fuzz testing (7.5%), random testing (5%) and scripted based testing (2.5%). **Conclusions:** In recent years, the number of proposals for automated software testing of mobile applications has increased. In 40% of the studies, the testing techniques use GUI-based models of the application. Further research is needed, in order to improve the creation of effective and efficient models for automated testing of mobile applications.

Keywords: random testing, model-based testing, model-learning testing, systematic testing, capture/replay, scripted-based, systematic literature review

1 Introduction

Mobile applications become strategic business tools and have changed the business landscape. They have also expanded to meet the communication and information sharing needs of social networks. Nowadays, there are a variety of new scenarios and applications for mobile devices. Indeed, users and developers expect reliability, ease of use, performance and security. Unfortunately, the quality of mobile applications is lower than expected. According to Amalfitano and Fasolino [1], this lack of quality is due to the very rapid development process in which software testing is neglected or carried out in a superficial way.

Software testing of mobile applications is not a trivial task due to several factors such as the variety of inputs that a mobile application normally requires and the heterogeneity of the technologies used to implement them. The variety of inputs that a mobile application normally requires (such as user input, input context and environment) makes it very difficult to design appropriate test cases to detect faults. Several studies state the importance of the context in which mobile applications are executed to ensure quality [2, 3, 4]. Performing testing on multiple platforms is needed in order to ensure quality.

To face the growing demand for high quality mobile applications, developers must spend more effort and attention on the processes of software development. In particular, software testing and automated testing play a strategic role in ensuring the quality of these applications. Understanding how automated testing of mobile applications is carried out in the practice is needed. Our study makes the following contributions:

– It identifies the main approaches for automated testing of mobile applications and the main research trends over time.
– It analyzes the available evidence on automated testing of mobile applications regarding its usefulness and accuracy.

These results will help professionals make informed decisions based on evidence about the selection of automated testing approaches. The remainder of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the research design of the systematic map and review. Section 4 and Section 5 present the results of the map and review. Finally, Section 6 discusses the results and Section 7 concludes the paper.

2 Related work

In this section, a review of secondary studies that were conducted in mobile automated testing is presented. Muccini et al. [2] conducted a literature survey on new lines of research in software test automation for mobile applications. In this paper, the authors conclude that mobile applications are different than traditional software. They also state that the challenges for automated testing of mobile applications seems to be related to the context and the mobility of applications. They state that the performance, security, reliability, and energy used are strongly affected by the variability of the environment in which the mobile device moves. This study is limited to only 18 studies. Kirubakaran and Karthikeyani [5] conducted a literature survey (21 studies) where they confirm previous results in [2].

Amalfitano et al. [6] analyzed challenges and open problems in the field of software testing for the Android platform. They described guidelines, models, techniques, and technologies for software testing in Android applications. They developed a comprehensive explanation of unit, integration, and system testing for Android applications. They also addressed the issue of software testing regarding to non-functional requirements: performance testing, stress testing, security testing, compatibility testing, usability testing and accessibility testing. Finally, they have listed tools and frameworks for automated testing in Android.

The main difference with our work is that these studies are not systematic literature reviews.

3 Research design

Secondary studies aggregate evidence from primary studies [7]. To increase the validity of the results, it is important to be systematic when evidence is analyzed. A systematic review (SLR) is a well-defined methodology that enables the identification, evaluation, and interpretation of all available and relevant evidence related to a specific research question, subject matter, or event of interest [8]. SLRs collect and synthesize empirical evidence from different sources [9]. Systematic mapping studies (SMS) provide an overview of a research area and identify the quantity and type of research and results available within it [10]. Mapping and review questions have different aims. A SMS question aims at structuring a research area and shows how the studies are distributed. A SLR question is related to the evidence and provides recommendations based on the strength of this evidence [10]. The following sections detail the protocol for the SMS and the SLR according to the guidelines proposed in [9, 10, 11] and considering the recommendations of [12, 13].

3.1 Map and review questions

A systematic map and literature review was conducted in order to identify, map, analyze, and characterize automated testing in the context of mobile applications. Table 1 shows the questions related to systematic map and review.

3.2 Search strategy

The search strategy aim is to find all relevant primary studies for answering research questions. First, the search string is defined and relevant databases are selected. Then, the studies' inclusion and exclusion criteria and procedure are defined.

The definition of the search string was based on population, intervention, comparison, and outcome (PICO) [14]. The search terms used in the map and systematic review were constructed using the following strategy [15]: (1) major terms were derived from the research questions by identifying the population, intervention and outcome, (2) alternative spellings and synonyms were identified

Table 1. Mapping study questions and review study questions

Mapping questions	**Review questions**
MQ1 Which are the main journals and conferences for automated testing of mobile applications?	RQ1 What are the challenges of automated testing of mobile applications?
MQ2 Which are the main authors for automated testing techniques research?	RQ2 What are the different approaches for automated testing of mobile applications?
MQ3 How is the frequency of papers distributed according to their testing approach?	RQ3 What is the most used experimental method for evaluating automated testing of mobile applications?

based a reference set of relevant papers. The reference set was created including the articles that are relevant to be included in the review. The reference set consisted of the following references: [16, 17, 18, 19, 4, 20, 21, 22, 23, 24]. (3) Besides, alternative terms were included via expert knowledge in the field, (4) the string was constructed with the Boolean OR for alternative spellings and the AND to link PICO categories. Finally, (5) the search string was piloted in several runs in order to reduce the quantity of noise of the results. An excerpt from the search string is shown below: ("mobile apps" OR "android apps") AND ("systematically testing" OR "automated testing") AND ("framework" OR "tool")[1].

The papers were searched based on title, abstract and keywords. The search string was used to search the following four digital libraries: *SCOPUS* (<scopus.com>), *ISI Web of Science* (<www.isiknowledge.com>), *IEEE Xplore* (<ieeexplore.ieee.org>), and *Engineering Village* (<www.engineeringvillage.org>).

The search was run on the four digital libraries many times iteratively. In each iteration the outcomes were analyzed and the search string was refined. This cyclical process stopped when the resulting studies were related to the testing of mobile applications. The goal of this preliminary check was to assess the effectiveness of the search string.

The selection of primary studies was conducted in three stages. In stage 1, applying the search string to the digital libraries performed an automated search. In stage 2, duplicates were identified and removed. In stage 3, we applied the inclusion/exclusion criteria to each study. The criteria are related to the availability of the study, their focus on automated testing, techniques and tools, and automation[2].

Two of the authors evaluated each study according to the inclusion and exclusion criteria. The selection procedure was conducted following these steps[3]: Two

[1] Full search string is available in:
http://eseg-cr.com/research/2014/map_and_review.pdf

[2] Full inclusion/exclusion criteria are available in:
http://eseg-cr.com/research/2014/map_and_review.pdf

[3] Full selection procedure is available in:
http://eseg-cr.com/research/2014/map_and_review.pdf

authors read the titles and abstracts separately. The papers were categorized as follows: (1) Accepted (2) Rejected and (3) Not defined.

The search was conducted (piloted, refined, and finished) during the first semester of 2014, and the quality assessment, extraction, mapping and analysis of the data on the second part of 2014. The preliminary results were documented and reviewed in this period.

3.3 Study quality assessment

Quality assessment is concerned with the rigor of the study and the applicability of the automated testing techniques for mobile applications. The quality of the papers was evaluated as regards to their objectives, context, test case derivation and execution, automation process, automation tool, tool availability, formal specification, empirical validation, findings, and conclusions criteria[4]. Studies were assessed using a checklist. The minimum evaluation that the study may receive is zero and the maximum is 31.

3.4 Data extraction

Based on the selection procedure, the required data for answering the map and review questions were extracted. This information is additional to the study quality criteria. A data extraction form was created and filled in for each study. The data extraction form has three different sections in which the following information was captured:

- The first section captures general information about the paper such as: data extractor, title, author, year, journal, conference, study identifier.
- The second section extracts data about the objectives, automated testing techniques, context, test cases creation and execution, tool support, finding and conclusions.
- The third section extracts information about the empirical validation.

The data extraction process was conducted by the principal author and validated by the second author in a random sample of studies. Extracted data was summarized in separate tables and later consolidated by both authors in a single file. Based on the extracted data, the analyses for the map and the review questions were conducted.

3.5 Analysis

The most advanced form of data synthesis is meta-analysis. However, this approach assumes that the synthesized studies are homogeneous [7]. Meta-analysis is not applied in this review because varieties of model and evaluation approaches

[4] Full quality assessment checklist is available in: http://eseg-cr.com/research/2014/map_and_review.pdf

have been discussed in the selected papers. Dixon-Woods et al. [15, 25] describe the content analysis and narrative summary as approaches to integrate evidence. Content analysis categorizes data and analyzes frequencies of categories transferring qualitative into quantitative information [25]. Some categories relevant for this review were identified in selected studies according to the recommendations in [10]. Narrative summaries report findings and describe them. Evidence in diverse forms can be discussed side by side [25]. This approach is suitable to synthesize the evidence of the selected papers of this review.

All studies were evaluated using the proposed section 3.3. Analyze was focused on the 44 studies with the highest rigor (tables 2 and 4).

3.6 Threats to validity

This section analyzes the threats to the validity for this study and the actions undertaken to mitigate them. Relevant limitations were the authors' bias, the exclusion of relevant articles, and the generality of the results. **Search process**: It is based on four digital libraries relevant in software engineering. **Study selection**: It is based on title and abstract. The inclusion and exclusion criteria were defined, evaluated, and adjusted by two researchers in order to reduce bias.

Quality assessment: a single researcher conducted the quality assessment checklist. Detailed criteria were defined to make the evaluation as easy as possible. The review protocol was peer-reviewed to assure the clarity of the quality criteria. **Data extraction**: a single researcher conducted the data extraction using a detailed extraction form to make the results as structured as possible. The review protocol was peer-reviewed to assure good understandability and clarity for the extracted information. After the extraction, a single reviewer checked the extraction forms to review if something was missing. Test-retest often suggested for single research was not conducted [26]. **Generalization of the results**: it is limited by the generalizability of the studies included in the review. The context information was extracted according to [27] to identify relevant context information in software engineering research.

4 Mapping results

4.1 Number of identified studies

We obtained 248 studies and after removing duplicates, 129 studies remained. Two authors applied the inclusion/exclusion criteria on the 129 studies independently and then compared results. Throughout the detailed review further articles were discarded, leaving 83 primary studies[5] being included in the analysis of this map and review.

[5] Study quality assessment and study references are available in: http://eseg-cr.com/research/2014/map_and_review.pdf

4.2 Publication (Mapping Question 1 and 2)

The principal authors are: Domenico Amalfitano, Anna Rita Fasolino and Porfirio Tramontana. They have published four studies together. Moreover, Kon Kim Haeng and Iulian Neamtiu have published four studies. Mukul R. Prasad, Sergiy Vilkomir and LIU Zhi-fang have published three studies.

Table 3 shows top six forums for automated testing of mobile applications. The main forum where studies about automated testing of mobile applications were published was ICSE where 9 studies were published.

4.3 Trends over time (Mapping Question 3)

Figure 1 shows the distribution of selected primary studies published before July 26, 2014. The 44 studies reporting the highest evaluation are shown, the evaluation was performed following the procedure described in section 3.3. The number of publications has increased in recent years. In the last three years, many approaches have employed model-based testing technique. The aim is to create a model of the application under test. Creating a good representation model of the application under test is one of the main tasks of the analyzed studies (40%). The main approaches identified were model-based testing (30%), capture/replay (15.5%), model-learning testing (10%), systematic testing (7.5%), fuzz testing (7.5%), random testing (5%) and scripted based testing (2.5%).

5 Review results

This section describes the results related to the systematic review questions presented in section 3.1. The results of each study were tabulated and analyzed. Techniques for automated testing of mobile applications have been classified into model-based testing, capture/replay, model-learning testing, systematic testing, fuzz testing, random testing, scripted-based testing techniques [28, 29, 30].

Model-based testing builds a model of the applications being tested and use this model to generate test cases. GUI models can either be constructed manually or generated automatically. Capture/replay can be captured events while the test cases are executed and then replay them automatically. Model-learning testing builds a model of the GUI application in conjunction with a testing engine and guide the generation of user input sequences based on the model. Systematic testing automatically and systematically generates input events to exercise applications. Fuzz testing generates a large number of simple inputs to application. Random testing generates random sequences of user input events for the application being tested. Scripted-based testing requires manually writing test cases before execute automatically.

Table 2 provides an overview of studies that follow the techniques described above. In this and all forthcoming tables the studies are ordered from highest to lowest rigor based on quality criteria.

Table 4 shows the approaches focused on platforms, web services and infrastructure and others. This group is based on platforms that support real devices

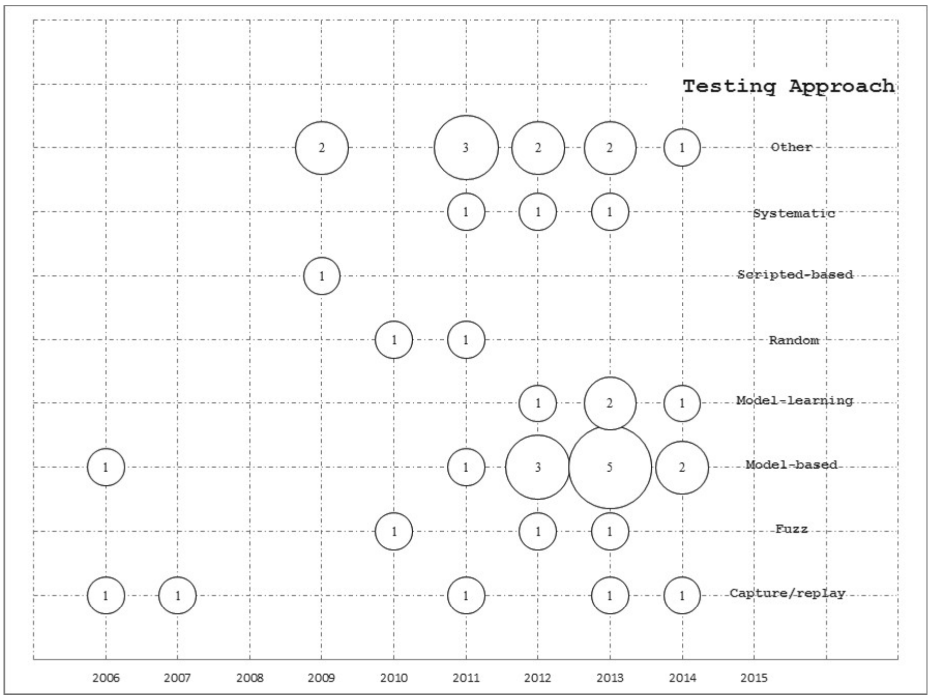

Fig. 1. Distribution of studies published per year and techniques

that can be accessed locally or through the cloud. The categories were created using the technique de keywording proposed in [10].

6 Discussion

In this section we try discussed the three research questions. **RQ1. What are the challenges of automated testing of mobile applications?** Variety of context events, fragmentation in both software and hardware, and resource limitation in mobile device are discussed.

Variety of context events. Abowd et al. [31] define a context as: "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is including the user and applications themselves".

The behavior of smart mobile devices is highly interactive, according to (S55). Mobile applications are affected by this interaction. Users consistently perform actions on their applications and the applications respond to these actions. Automating this type of testing scenario is complex.

According to (S17), a mobile application receives user GUI input and environmental context events. User GUI input such as keyboard events and touch events. Environmental context events that include physical context events and social context events. Physical context events obtained from the built-in GPS receiver, Bluetooth chips, accelerometer sensor, humidity sensor, magnetic sensor and network. Social context events, like nearby MSN friends, the current activity the user is up to, and even the user's mood. In (S33), these events are called "context events". The mobile application must accept and react to the changing context events as inputs to produce appropriate outputs. Modeling the context events of mobile applications for automated testing is one of the major challenges currently.

[H]

Table 2: Overview of the studies according to the employed technique

Study	Year	Approach	Evaluation	Context	Data source	Outcome	Rigor
S35	2013	Model-based testing	Case study	Android	8 real-word applications	A security testing method of Android applications which aims at detecting data vulnerabilities based on the intent mechanism	29
S42	2013	Model-learning testing	Case study	Android	10 apps from F-Droid open applications market	The approach for complex apps, outperforms both random and L^*3-based testing	29
S7	2013	Model-based testing	Case study	Android	8 open-source applications	Orbit efficiently extracted high-quality models, fully automatically	26
S24	2012	Systematic Testing	Case study	Android	5 real-word applications	ACTEve is significantly more efficient than the naive concolic execution technique	26
S25	2014	Model-based testing	Case study	Android	6 applications	QUANTUM found a total of 22 bugs, using a total of 60 tests for 6 apps	26
S27	2013	Model-based testing	Case study	Android	5 real-word applications	On average, a test case consists of 10 events. 53% of the events are connector events	26
S39	2013	Model-learning based	Case study	Android	50 open-source applications and 1,000 free applications on Google Play	Dynodroid found 9 bugs in 7 of the 50 apps, and 6 bugs in 5 of the top 1,000 free apps on Google Play	26
S50	2014	Model-learning based	Case study	Android	4 open-source applications	MobiGUITAR reported 10 new bugs	26
S64	2013	Capture/replay	Case study	Android	100 real-world applications in Google Play	The approach can be applied successfully for repeatability, bug reproducibility, and execution time-warping	26
S70	2013	Model-based testing	Case study	Android	25 popular Android applications	The exploration techniques achieve 59.39-64.11% activity coverage and 29.53-36.46% method coverage	26
S9	2011	Model-based testing	Sample example	Android	Sample example	Effectiveness and usability of A^2T^2 is showed	25
S81	2012	Model-learning testing	Sample example	Android	One real-word application	For an application revealed 4 undocumented bugs, automatically detected in less than 5 hours	25
S57	2012	Model-based testing	No clear description	Android	No clear description	A framework systematically generates test cases from the formal specification of the mobile platform and performs conformance testing with the generated test cases	24
S72	2012	Model-based testing	No clear description	No clear description	No clear description	An approach applying Model-based Testing (MBT) and Dynamic Feature Petri Nets (DFPN) to define a test model from which an extensive test suite can be derived	24
S17	2010	Adaptive random testing	Case study	Android	6 real-word applications	The technique can both reduce the number of test cases and the time required to expose first failure when compared with random technique	23
S29	2014	Model-based testing	Case study	Android	3 real-word applications	DroidCrawle automatic traverse the GUIs with a well- designed traversal algorithm	23
S30, S08	2011	Random testing	Case study	Android	10 popular applications in the Android Market	The technique reported activity, event and type error that have already been reported and new errors	23
S33	2013	Model-based testing	Case study	Android	5 real Android applications published in the Android Market	Effectiveness of technique was showed in terms of code coverage and method coverage	23
S38	2013	Fuzz Testing	Case study	Android	3 video-playing apps are tested using DroidFuzzer on 3 video data formats	14 bugs are found and a vulnerability is verified	23
S43	2009	Scripted-based testing	Case study	Java	Applications were deployed on several mobile devices	Hermes automates testing of a Java Midlet application that may run on multiple heterogeneous devices	23
S55	2007	Capture/replay	Case study	Symbian	3 testing teams are founded and there are 2 testing engineers in each team	Automated tests find more bugs than manual testing	23
S66	2013	Systematic testing	Case study	Android	3 applications	An approach focuses on inter-application communication of those messages that are directed to activities	23
S15, S6	2012	Fuzz Testing	Sample example	Android	Unspecified	No results are showed	22
S21	2011	Capture/replay	Sample example	iOS and Android	One mobile project	22 test cases were created. Approximately 80% of execution time saving achieved	22
S28	2014	Capture/Replay	Sample example	iOS	A bike sharing prototype application	An automated usability testing tool for iOS applications	22
S74	2012	Model-based testing	Sample example	Android	Google's Notepad application	A unit-based test approach was presented for testing conformance of life cycle dependent properties of mobile applications, using assertions	22
S4	2010	Fuzz Testing	Case study	Symbian	5 retail handsets based on Symbian OS	MAFIA generated more than 13,000 malformed inputs that successfully crash multiple Symbian OS retail phones	21
S13	2006	Model-based testing	Sample example	Java applications	A sample application	No results are showed	21
S77	2006	Capture/replay	No clear description	Symbian	Many applications and two Nokia 6680 mobile phones	A experimental testing system was developed for CB3G applications and to automate repeated test execution in multiple mobile phones at the same time	21
S82	2011	Systematic testing	No clear description	Android	No clear description	A security validation service that analyzes smartphone applications submitted to app markets and generates reports that aid in identifying security and privacy risks	21

Table 3. Publications in journals and conferences

Journal or Conference	# Study
International Conference on Software Engineering (ICSE)	9
Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (LNCS)	6
Communications in Computer and Information Science (CCIS)	5
ACM International Conference Proceeding Series (ACM-ICPS)	4
International Workshop on Automation of Software Test (AST)	3
International Conference on Software Testing, Verification and Validation Workshops (ICSTW)	3

[H]

Table 4: Approaches focused on platforms, web services and infrastructure

Study	Year	Approach	Evaluation	Context	Data source	Outcome	Rigor
S41	2014	Malware detectors	Case study	Android	500 applications from Google Play Store	FLOWDROID achieves 93% recall and 86% precision, greatly outperforming the commercial tools IBM AppScan Source and Fortify SCA	26
S58	2013	Spectrum-based fault localization technique	Case study	Android	4 open-source applications	The collected results show that the used instrumentation entails an average time overhead of 5.75% (with standard deviation σ=2.49)	26
S73	2012	Online platform for automating UI testing	No clear description	Android	Testdroid has attracted over 1000 developers and has conducted over 20000 tests	An online platform for automating UI testing on a variety of mobile handsets	24
S31	2011	Automated testing of service-oriented distributed applications	Case study	Android	5 smartphones are connected to a server	A decentralized testing automation framework that accomplishes distributed test sequencing and control for service-oriented applications was presented	23
S61	2009	Performance testing	Case study	Java	Two java applications	A performance unit test method for mobile applications using MObilePBDB was proposed and the mobile performance unit test tool PJUnit was proposed	23
S05, S10	2011	Domain-Specific Modeling Language	No clear description	Multiplatform	No information available	MATeL can be easily adapted and/or extended, plugged into any industrial test bed as well	22
S11	2012	Automation testing of mobile applications directly on the mobile devices	Sample example	Symbian	Sample application	MobTAF has been proposed which performs automation testing of mobile applications directly on the mobile devices	22
S32, S46	2013	Testing as a Service	No clear description	Android	No clear description	The high-level architecture of TaaS solution and the detailed architecture of the CTOMS platform were described	21
S60	2009	Performance testing in Test Driven Development	No clear description	Java	No clear description	A mobile performance unit testing tool that supports the functional testing in the development process of unit testing environment and also supports performance unit testing generation and performance automation	21
S69	2011	Touchscreen meta-testing model	No clear description	Android	No clear description	Five test scenarios pertaining to the presented meta-test model	21

Fragmentation in both software and hardware. Fragmentation is the main problem related to mobile devices, and it is manifested in both software and hardware, in (S51, S73, S17) this problem is discussed. Applications behave differently on each device, and usability and performance are affected. Mobile operating systems are still in a rapid iterative upgrade process. According to S55, the diversity of mobile devices and platforms is reducing the reusability and maintainability of test cases.

Changing mobile ecosystem. According to (S83), mobile applications offer different services through an end-to-end mobile ecosystem, composed of mobile devices, mobile networks and application servers. These components exhibit significant variations in their behavior. These changes in context are perceived by mobile applications and affect their function. They need to be tested against variations exhibited by each one of the above-mentioned components.

According to (S83), mobile networks are comprised of many factors outside the control of the application, such as network connectivity, bandwidth, and congestion. Mobile applications should be evaluated to ensure that they behave properly when they are subjected to such network conditions. Testing the application against a variety of network conditions is required.

Resource limitation in mobile device. Resource limitation is still a topic that we should consider to ensure the quality of mobile applications. Device limitations vary in terms of memory, processing power, screen type, battery level, storage capacity, platform version and input method, according to (S12). Limited resources on a mobile device can influence the testing process. The testing tool or process consumes resources, and it may affect the normal execution of the mobile applications themselves, according to (S17).

RQ2. What are the different approaches for automated testing of mobile applications? In the literature, there are many approaches to address the automated testing of mobile applications. In section 5 they are presented.

There are some tools available to download on line, below some of them are discussed. Dynodroid (S39) uses model learning and random testing techniques for generating inputs to mobile applications. It is based on an "observe-select-execute" principle that efficiently generates a sequence of relevant events. It generates both user interface and system events, and seamlessly combines events from humans and machines. In MobiGUITAR (S50) the application model is created using an automated reverse engineering technique called GUI Ripping. Test cases are generated using the model and test adequacy criteria. A^3E (S70) uses model based testing techniques. This tool systematically explores applications while they are running on actual phones. A^3E uses a static, taint-style, dataflow analysis on the application bytecode in a novel way, to construct a high-level control flow graph that captures legal transitions among activities (application screens). It then uses this graph to develop an exploration strategy named Targeted Exploration, which permits fast, direct exploration of activities, including activities that would be difficult to reach during normal use. It also developed a strategy named Depth-first Exploration that mimics user actions for exploring activities and their constituents in a slower, but more systematic, way.

SwiftHand (S42) generates sequences of test inputs for Android applications. It uses machine learning to learn a model of the application during testing, uses the learned model to generate user inputs that visit unexplored states of the application, and uses the execution of the applications on the generated inputs to refine the model. Also, the testing algorithm avoids restarting the application during testing.

Strategies for record and replay have also been explored. In (S64) RERAN is described. It permits record-and-replay events for the Android smartphone platform. It addresses challenges of sophisticated GUI gestures by directly capturing the low-level event stream on the phone, which includes both GUI events and sensor events, and replaying it with microsecond accuracy. This approach allows RERAN to capture and playback GUI events, i.e., touchscreen gestures (e.g., tap, swipe, pinch, zoom), as well as those from other sensor input devices (e.g., accelerometer, light sensor, compass).

We found other tools that offer attractive solutions for mobile application testing, but we did not know if these were available via the Internet. In (S25) QUANTUM is described. It is a framework for authoring test oracles for checking user-interaction features of mobile applications, in an application agnostic manner. The framework supports model-driven test suite generation where each generated test includes both the test sequence to execute and the corresponding assertions to check (as test oracles). The framework uses its built-in, extensible library of oracles (for various user-interaction features) and a model of the user-interface for generating a test suite to comprehensively test the application against user-interaction features. The framework supports the following user-interaction features: double rotation, killing and restarting, pausing and resuming, back button functionality, opening and closing menus, zooming in, zooming out and scrolling. In (S07) Orbit is described. It uses a model based testing technique. It is based on a grey-box approach for automatically extracting a model of the application being tested. A static analysis of the application source code is used to extract the set of user actions supported by each widget in the GUI. Next, a dynamic crawler is used to reverse engineer a model of the application, by systematically exercising extracted actions on the live application.

In several studies, proposed frameworks and platforms for mobile application testing appear. However, very few of these proposals are available to be used. Here we will mention some of these platforms or frameworks that provide concrete solutions for testing of mobile applications but we did not know if this is available via the Internet. In (S82) AppInspector is described. It is an automated security validation system. In (S73) Testdroid is described. It is an online platform for automating user interface testing.

Although there are many proposals for automated testing of mobile applications, few tools are available for download via the Internet or be using in the cloud.

RQ3. What is the most used experimental method for evaluating automated testing of mobile applications?

Of the 40 studies included in the review, 23 studies reported empirical validation through a case study, in 8 studies reported a sample example and 9 studies did not reported a clear description of the validation process. empirical used.

The presented case studies, there are very ambitious and well organized studies in the field. There are small examples that claim to be case studies. In many cases, the information reported case study is insufficient to classify its stringency.

7 Conclusions

Mobile phones receive events of the Internet (such as email or social network notifications), events from the external environment and sensed by sensors (such as GPS, temperature, pressure), events generated by the platform of hardware (such as battery and external peripheral port), events typical of mobile phones (such as phone calls and text messages) and events produced through of the GUI by users. The techniques discussed in this paper focus primarily on user events and very few address the remaining events. Techniques for generating test cases integrating as many events as possible still need to be developed. These techniques should emulate the changing contexts in which mobile applications operate.

In 40% of the studies, the technique used is based on the creation or use of a model of the GUI application. This model is used to construct a suite of test cases to test the application. There is still more research needed to create models that are effective and efficient for automated testing of mobile applications.

Testing of mobile applications can leverage the strengths of cloud computing, such as parallel and distributed systems, virtualization, and software services. Cloud computing allows application instances to run independently on a variety of operating systems and connect them through web services. Research sites could share real devices by reducing the cost of purchased equipment. The execution of parallel test cases can reduce the time required for testing. However, few approaches are currently leveraging the benefits of cloud computing for testing.

There are many proposed tools for mobile application testing but many of them are not available on the web for download. Others tools can be downloaded, but in our opinion, they are complex to use and their user manuals are somewhat deficient. Some tools are very specific and only serve to test a few features of the applications.

Systematic literature review of evidence needs to be performed to determine the experimental designs and the use of metrics to provide empirical evidence in studies related to testing of mobile applications.

For future work we would like to set the search string to achieve include more studies in the systematic literature review. In addition, we would like to make a wider analyze on each of the techniques described in the studies. Also, we would like to make a case study to evaluate and compare the tools that are available according to the studies analyzed.

Acknowledgments

This research was supported by the Costa Rican Ministry of Science, Technology and Telecommunications (MICITT). Our thanks to the Empirical Software Engineering (ESE) Group at University of Costa Rica.

References

[1] Amalfitano, D., Fasolino, A., Tramontana, P., De Carmine, S., Memon, A.: Using gui ripping for automated testing of android applications. In: 2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings. (2012) 258–261 cited By (since 1996)20.
[2] Muccini, H., Di Francesco, A., Esposito, P.: Software testing of mobile applications: Challenges and future research directions. In: 2012 7th International Workshop on Automation of Software Test, AST 2012 - Proceedings. (2012) 29–35 cited By (since 1996)1.
[3] Wang, Z., Elbaum, S., Rosenblum, D.: Automated generation of context-aware tests. In: Software Engineering, 2007. ICSE 2007. 29th International Conference on. (2007) 406–415
[4] Amalfitano, D., Fasolino, A., Tramontana, P., Amatucci, N.: Considering context events in event-based testing of mobile applications. In: Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013. (2013) 126–133 cited By (since 1996)0.
[5] Kirubakaran, B., Karthikeyani, V.: Mobile application testing-challenges and solution approach through automation. (2013) 79–84 cited By (since 1996)0.
[6] Amalfitano, D., Fasolino, A., Tramontana, P., Robbins, B.: Testing android mobile applications: Challenges, strategies, and approaches. Advances in Computers **89** (2013) 1–52 cited By (since 1996)1.
[7] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B.: Experimentation in Software Engineering. Springer (2012)
[8] Kitchenham, B.: Procedures for performing systematic reviews. Technical report, Departament of Computer Science, Keele University (2004)
[9] Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report (2007) http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf.
[10] Petersen, K., Feldt, R., Shahid, M., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the Evaluation and Assessment in Software Engineering (EASE'08), Bari, Italy (June 2008) 1–10
[11] Biolchini, J., Mian, P.G., Natali, A.C.C.: Systematic review in software engineering. Technical Report RT-ES 679/05, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil (May 2005)
[12] Dybå, T., Dingsøyr, T., Hanssen, G.K.: Applying systematic reviews to diverse study types: An experience report. In: ESEM, IEEE Computer Society (2007) 225–234
[13] Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. The Journal of Systems and Software **80**(80) (2007) 571–583
[14] Pai, M., McCulloch, M., Gorman, J.D., Pai, N., Enanoria, W., Kennedy, G., Tharyan, P., Colford, J.M.: Systematic reviews and meta-analyses: an illustrated, step-by-step guide. Natl Med J India **17**(2) (2004) 86–95
[15] Petersen, K., Wohlin, C.: Context in industrial software engineering research. In: ESEM. (2009) 401–404
[16] Zaeem, R., Prasad, M., Khurshid, S.: Automated generation of oracles for testing user-interaction features of mobile apps. (2014) 183–192 cited By (since 1996)0.
[17] Amalfitano, D., Fasolino, A., Tramontana, P., Ta, B., Memon, A.: Mobiguitar – a tool for automated model-based testing of mobile apps. Software, IEEE **PP**(99) (2014) 1–1
[18] Azim, T., Neamtiu, I.: Targeted and depth-first exploration for systematic testing of android apps. ACM SIGPLAN Notices **48**(10) (2013) 641–660 cited By (since 1996)0.
[19] Yang, W.b., Prasad, M., Xie, T.: A grey-box approach for automated gui-model generation of mobile applications. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7793 LNCS** (2013) 250–265 cited By (since 1996)3.
[20] Gomez, L., Neamtiu, I., Azim, T., Millstein, T.: Reran: Timing- and touch-sensitive record and replay for android. In: Proceedings - International Conference on Software Engineering. (2013) 72–81 cited By (since 1996)1.
[21] Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S., Stavrou, A.: A whitebox approach for automated security testing of android applications on the cloud. In: 2012 7th International Workshop on Automation of Software Test, AST 2012 - Proceedings. (2012) 22–28 cited By (since 1996)2.
[22] Kaasila, J., Ferreira, D., Kostakos, V., Ojala, T.: Testdroid: Automated remote ui testing on android. In: Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia, MUM 2012. (2012) cited By (since 1996)0.
[23] Gilbert, P., Chun, B.G., Cox, L., Jung, J.: Vision: Automated security validation of mobile apps at app markets. In: MobiSys'11 - Compilation Proceedings of the 9th Int. Conf. on Mobile Systems, Applications, and Services and Co-located Workshops - 2011 Workshop on Mobile Cloud Computing and Services, MCS'11. (2011) 21–25 cited By (since 1996)8.
[24] Jiang, B., Long, X., Gao, X.: Mobiletest: A tool supporting automatic black box test for software on smart mobile devices. (2007) cited By (since 1996)0.
[25] Dixon-Woods, M., Agarwal, S., Jones, D., Young, B., Sutton, A.: Synthesising qualitative and quantitative evidence: a review of possible methods. J Health Serv Res Policy **10**(1) (Jan 2005) 45–53
[26] Petersen, K.: Measuring and predicting software productivity: A systematic map and review. Information & Software Technology **53**(4) (2011) 317–343
[27] Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14, New York, NY, USA, ACM (2014) 38:1–38:10
[28] Choi, W., Necula, G., Sen, K.: Guided gui testing of android apps with minimal restart and approximate learning. ACM SIGPLAN Notices **48**(10) (2013) 623–639 cited By (since 1996)0.
[29] Machiry, A., Tahiliani, R., Naik, M.: Dynodroid: An input generation system for android apps. (2013) 224–234 cited By (since 1996)5.
[30] Nguyen, B., Robbins, B., Banerjee, I., Memon, A.: Guitar: an innovative tool for automated testing of gui-driven software. Automated Software Engineering (2013) 1–41
[31] Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. HUC '99, London, UK, UK, Springer-Verlag (1999) 304–307