Automated Testing of Mobile Applications: A Systematic Map and Review

Abel Méndez-Porras, Christian Quesada-López, and Marcelo Jenkins

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

This results will help professionals to 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. The research design of the systematic map and review is described in Section 3. The results of the map and review are presented in Section 4 and Section 5. Finally, Section 6 discusses the results and Section 7 concludes the paper.

2 Related work

In this section, a review of secondary studies that have been 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, authors conclude that mobile applications are different than traditional software. They also state that the challenges for automated testing of mobile applications seem 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 with 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 have been 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 is 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 base on
<table>
<thead>
<tr>
<th>Mapping questions</th>
<th>Review questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ1 Which are the main journals and conferences for automated testing of mobile applications?</td>
<td>RQ1 What are the challenges of automated testing of mobile applications?</td>
</tr>
<tr>
<td>MQ2 Which are the main authors for automated testing techniques research?</td>
<td>RQ2 What are the different approaches for automated testing of mobile applications?</td>
</tr>
<tr>
<td>MQ3 How is the frequency of papers distributed according to their testing approach?</td>
<td>RQ3 What is the most used experimental method for evaluating automated testing of mobile applications?</td>
</tr>
</tbody>
</table>

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”).

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.

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

---

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 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 Fusolino 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
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.
<table>
<thead>
<tr>
<th>Study</th>
<th>Year</th>
<th>Approach</th>
<th>Evaluation</th>
<th>Context</th>
<th>Data source</th>
<th>Outcome</th>
<th>Rigor</th>
</tr>
</thead>
<tbody>
<tr>
<td>S35</td>
<td>2013</td>
<td>Model-based test</td>
<td>Case study</td>
<td>Android</td>
<td>8 real-world applications</td>
<td>A security testing method of Android applications which aims at detecting data vulnerabilities based on the intent mechanism</td>
<td>29</td>
</tr>
<tr>
<td>S42</td>
<td>2013</td>
<td>Model-learning</td>
<td>Case study</td>
<td>Android</td>
<td>10 apps from P-Droid open</td>
<td>The approach for complex apps, outperforms both random and L*-based testing</td>
<td>29</td>
</tr>
<tr>
<td>S7</td>
<td>2013</td>
<td>Model-based testing</td>
<td>Case study</td>
<td>Android</td>
<td>8 open-source applications</td>
<td>Orbit efficiently extracted high-quality models, fully automatically</td>
<td>26</td>
</tr>
<tr>
<td>S24</td>
<td>2012</td>
<td>Systematic Testing</td>
<td>Case study</td>
<td>Android</td>
<td>5 real-world applications</td>
<td>ACTe is significantly more efficient than the naive concord execution technique</td>
<td>26</td>
</tr>
<tr>
<td>S25</td>
<td>2014</td>
<td>Model-based test</td>
<td>Case study</td>
<td>Android</td>
<td>6 applications</td>
<td>QUANTUM found a total of 22 bugs, using a total of 60 tests for 6 apps</td>
<td>26</td>
</tr>
<tr>
<td>S27</td>
<td>2013</td>
<td>Model-based test</td>
<td>Case study</td>
<td>Android</td>
<td>5 real-world applications</td>
<td>On average, a test case consists of 10 events. 53% of the events are connector events</td>
<td>26</td>
</tr>
<tr>
<td>S29</td>
<td>2014</td>
<td>Model-learning</td>
<td>Case study</td>
<td>Android</td>
<td>50 open-source applications and 5,000 free applications on Google Play</td>
<td>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</td>
<td>26</td>
</tr>
<tr>
<td>S50</td>
<td>2013</td>
<td>Capture/replay</td>
<td>Case study</td>
<td>Android</td>
<td>4 open-source applications</td>
<td>MobileGUTAR reported 20 new bugs</td>
<td>26</td>
</tr>
<tr>
<td>S64</td>
<td>2013</td>
<td>Capture/replay</td>
<td>Case study</td>
<td>Android</td>
<td>100 real-world applications</td>
<td>The approach can be applied successfully for repeatability, bug re-producibility, and execution time-warping</td>
<td>26</td>
</tr>
<tr>
<td>S70</td>
<td>2013</td>
<td>Model-based test</td>
<td>Case study</td>
<td>Android</td>
<td>25 popular Android applications</td>
<td>The exploration techniques achieve 39.39-64.11% activity coverage and 29.53-36.46% method coverage</td>
<td>26</td>
</tr>
<tr>
<td>S9</td>
<td>2011</td>
<td>Model-based test</td>
<td>Sample example</td>
<td>Android</td>
<td>Sample example</td>
<td>Effectiveness and usability of A^2T^2s is showed</td>
<td>25</td>
</tr>
<tr>
<td>S81</td>
<td>2012</td>
<td>Model-learning</td>
<td>Sample example</td>
<td>Android</td>
<td>One real-world application</td>
<td>An approach systematically generates test cases from the formal specification of the mobile platform and performs conformance testing with the generated test cases</td>
<td>25</td>
</tr>
<tr>
<td>S57</td>
<td>2012</td>
<td>Model-based test</td>
<td>Sample example</td>
<td>Android</td>
<td>No clear description</td>
<td>A framework systematically generates test cases from the formal specification of the mobile platform and performs conformance testing with the generated test cases</td>
<td>24</td>
</tr>
<tr>
<td>S72</td>
<td>2012</td>
<td>Model-based testing</td>
<td>No clear description</td>
<td>Android</td>
<td>No clear description</td>
<td>The technique can both reduce the number of test cases and the time required to expose first failure when compared with random testing</td>
<td>24</td>
</tr>
<tr>
<td>S80</td>
<td>2012</td>
<td>Random testing</td>
<td>Case study</td>
<td>Android</td>
<td>3 real-world applications</td>
<td>DroidCrawle automatically traverses the GUIs with a well-designed traversal algorithm</td>
<td>23</td>
</tr>
<tr>
<td>S80</td>
<td>2011</td>
<td>Random testing</td>
<td>Case study</td>
<td>Android</td>
<td>10 popular applications</td>
<td>The technique reported activity, event, and type error that haven’t been reported in the Android Market</td>
<td>23</td>
</tr>
<tr>
<td>S35</td>
<td>2012</td>
<td>Model-based test</td>
<td>Case study</td>
<td>Android</td>
<td>5 real-world applications</td>
<td>Effectiveness of technique was showed in terms of code coverage and published in the Android Market</td>
<td>23</td>
</tr>
<tr>
<td>S38</td>
<td>2013</td>
<td>Pan Testing</td>
<td>Case study</td>
<td>Android</td>
<td>3 video-playing apps are 14 bugs are found and a vulnerability is verified</td>
<td>Automated tests find more bugs than manual testing and there are 2 testing engineers in each team</td>
<td>23</td>
</tr>
<tr>
<td>S43</td>
<td>2009</td>
<td>Scripted-based</td>
<td>Case study</td>
<td>Java</td>
<td>Applications were deployed Hermes automates testing of a Java Midlet application that may run on multiple heterogeneous devices</td>
<td>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</td>
<td>23</td>
</tr>
<tr>
<td>S55</td>
<td>2007</td>
<td>Capture/replay</td>
<td>Case study</td>
<td>Symbian</td>
<td>3 testing teams are founded Automated tests find more bugs than manual testing and there are 2 testing engineers in each team</td>
<td>Automated testing of mobile applications</td>
<td>23</td>
</tr>
<tr>
<td>S66</td>
<td>2013</td>
<td>Systematic testing</td>
<td>Case study</td>
<td>Android</td>
<td>3 applications</td>
<td>An approach focuses on inter-application communication of these messages that are directed to activities</td>
<td>23</td>
</tr>
<tr>
<td>S15, S6</td>
<td>2012</td>
<td>Pan Testing</td>
<td>Sample example</td>
<td>Android</td>
<td>Unspecified</td>
<td>No results are showed</td>
<td>22</td>
</tr>
<tr>
<td>S21</td>
<td>2011</td>
<td>Capture/replay</td>
<td>Sample example</td>
<td>iOS</td>
<td>One mobile project</td>
<td>22 test cases were created. Approximately 80% of execution time saving achieved</td>
<td>22</td>
</tr>
<tr>
<td>S28</td>
<td>2014</td>
<td>Capture/Replay</td>
<td>Sample example</td>
<td>Android</td>
<td>iOS</td>
<td>An automated usability testing tool for iOS applications</td>
<td>22</td>
</tr>
<tr>
<td>S74</td>
<td>2012</td>
<td>Model-based test</td>
<td>Sample example</td>
<td>Android</td>
<td>Google’s Notepad application</td>
<td>A unit-based test approach was presented for testing conformance of life cycle dependent properties of mobile applications, using assertions</td>
<td>22</td>
</tr>
<tr>
<td>S4</td>
<td>2010</td>
<td>Pan Testing</td>
<td>Case study</td>
<td>Symbian</td>
<td>5 retail handsets based on Symbian</td>
<td>MAPIA generated more than 13,000 malformed inputs that successfully crash multiple Symbian O/S retail phones</td>
<td>21</td>
</tr>
<tr>
<td>S13</td>
<td>2006</td>
<td>Model-based testing</td>
<td>Sample example</td>
<td>Java applications</td>
<td>No clear description</td>
<td>A security testing service that analyzes smartphone applications submitted to app markets and generates reports that aid in identifying security and privacy risks</td>
<td>21</td>
</tr>
<tr>
<td>S77</td>
<td>2006</td>
<td>Capture/replay</td>
<td>Sample example</td>
<td>Many applications and two Symbian</td>
<td>A security testing service that analyzes smartphone applications submitted to app markets and generates reports that aid in identifying security and privacy risks</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>S82</td>
<td>2011</td>
<td>Systematic testing</td>
<td>No clear description</td>
<td>Android</td>
<td>No clear description</td>
<td>A security testing service that analyzes smartphone applications submitted to app markets and generates reports that aid in identifying security and privacy risks</td>
<td>21</td>
</tr>
</tbody>
</table>
Table 3. Publications in journals and conferences

<table>
<thead>
<tr>
<th>Journal or Conference</th>
<th># Study</th>
<th>Approach</th>
<th>Evaluation</th>
<th>Context</th>
<th>Data Source</th>
<th>Outcome</th>
<th>Rigor</th>
</tr>
</thead>
<tbody>
<tr>
<td>International Conference on Software Engineering (ICSE)</td>
<td>9</td>
<td>Fault detection in mobile applications</td>
<td>No clear description</td>
<td>No clear description</td>
<td>Google Play Store</td>
<td>FLOWDROID achieves 93% recall and 86% precision, greatly outperforming the commercial tools IBM AppScan Source and Fortify SCA</td>
<td>26</td>
</tr>
<tr>
<td>Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (LNCS)</td>
<td>6</td>
<td>Fault localization in mobile applications</td>
<td>No clear description</td>
<td>No clear description</td>
<td>Android 4 open-source applications</td>
<td>The collected results show that the used instrumentation entails an average time overhead of 5.75% (with standard deviation $\sigma = 2.49$)</td>
<td>26</td>
</tr>
<tr>
<td>Communications in Computer and Information Science (CCIS)</td>
<td>5</td>
<td>UI testing in mobile applications</td>
<td>No clear description</td>
<td>No clear description</td>
<td>Android Testdroid has attracted over 1000 developers and has conducted over 20000 tests</td>
<td>An online platform for automating UI testing on a variety of mobile handsets</td>
<td>24</td>
</tr>
<tr>
<td>ACM International Conference Proceeding Series (ACM-ICPS)</td>
<td>4</td>
<td>Service-oriented mobile application testing</td>
<td>Case study</td>
<td>Android 5 smartphones are connected to a server</td>
<td>A decentralized testing automation framework that accomplishes distributed test sequencing and control for service-oriented applications was presented</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>International Workshop on Automation of Software Test (AST)</td>
<td>3</td>
<td>No clear description</td>
<td>Case study</td>
<td>Android</td>
<td>Malware detectors</td>
<td>FLOWDROID achieves 93% recall and 86% precision, greatly outperforming the commercial tools IBM AppScan Source and Fortify SCA</td>
<td>26</td>
</tr>
<tr>
<td>International Conference on Software Testing, Verification and Validation Workshops (ICSTW)</td>
<td>3</td>
<td>No clear description</td>
<td>Case study</td>
<td>Android</td>
<td>Spectrum-based fault localization technique</td>
<td>The collected results show that the used instrumentation entails an average time overhead of 5.75% (with standard deviation $\sigma = 2.49$)</td>
<td>26</td>
</tr>
<tr>
<td>Journal of Software Quality Engineering</td>
<td>2</td>
<td>No clear description</td>
<td>Case study</td>
<td>Android</td>
<td>Online platform for automating UI testing</td>
<td>An online platform for automating UI testing on a variety of mobile handsets</td>
<td>24</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
<td>2</td>
<td>No clear description</td>
<td>Case study</td>
<td>Android</td>
<td>Automated testing of service-oriented applications</td>
<td>A decentralized testing automation framework that accomplishes distributed test sequencing and control for service-oriented applications was presented</td>
<td>23</td>
</tr>
<tr>
<td>Journal of Software Testing, Inspection, and Enhancement</td>
<td>2</td>
<td>Performance testing in mobile applications</td>
<td>Case study</td>
<td>Java</td>
<td>Performance testing in Test Driven Development</td>
<td>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</td>
<td>21</td>
</tr>
<tr>
<td>Journal of Software Engineering in Practice</td>
<td>2</td>
<td>Domain-Specific Modeling Language</td>
<td>No clear description</td>
<td>Multiplatform</td>
<td>MATeL can be easily adapted and/or extended, plugged into any industrial test bed as well</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
<td>2</td>
<td>Automation testing of mobile applications directly on the mobile devices</td>
<td>Sample example</td>
<td>Symbian</td>
<td>MobTAF has been proposed which performs automation testing of mobile applications directly on the mobile devices</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
<td>2</td>
<td>No clear description</td>
<td>Sample example</td>
<td>Symbian</td>
<td>Testing as a Service</td>
<td>The high-level architecture of TaaS solution and the detailed architecture of the CTOMS platform were described</td>
<td>21</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
<td>2</td>
<td>Performance testing in mobile applications</td>
<td>No clear description</td>
<td>Java</td>
<td>Performance testing in Test Driven Development</td>
<td>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</td>
<td>21</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
<td>2</td>
<td>Touchscreen meta-testing model</td>
<td>No clear description</td>
<td>Android</td>
<td>No clear description</td>
<td>Five test scenarios pertaining to the presented meta-test model</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 3. Publications in journals and conferences
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. A³E (S70) uses model based testing techniques. This tool systematically explores applications while they are running on actual phones. A³E 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. 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


