Software Quality Journal (2019) 27: 149–201
https://doi.org/10.1007/s11219-018-9418-6

Automated functional testing of mobile applications: a systematic mapping study

Porfirio Tramontana[1] · **Domenico Amalfitano**[1] · **Nicola Amatucci**[1] · **Anna Rita Fasolino**[1]

Published online: 24 October 2018

Abstract

Context Testing is a critical and costly activity in the life cycle of a mobile application, due to the growing request of new applications and to the rapid evolution of mobile devices and frameworks. Testing automation may represent an effective solution to improve the quality of mobile applications and to reduce testing costs. *Objective* We have performed a systematic mapping study to find, analyze, and classify papers in the scientific literature that are related to the automation of functional testing of mobile applications with the aim to provide a classification scheme useful for researchers and practitioners to have a clear view of the state of the art and to easily find existing solutions to their issues. *Method* We have conducted the study on the basis of a set of 18 research questions. Search queries have been formulated and applied to 7 search engines and the resulting papers have been filtered by considering sets of inclusion and exclusion criteria. The selected papers have been systematically classified and, in addition, a bibliometric analysis has been performed. *Results* A systematic map including 131 papers has been obtained and is publicly available. The papers have been classified on the basis of the supported testing activities, the characteristics of the techniques and tools they present, and the evaluation methodologies adopted to validate them. The bibliometric analysis has allowed the identification of the most active researchers, the most attractive venues, and the most influential papers. *Conclusions* The analysis of the systematic mapping has allowed the identification of some research trends and gaps in this field of study. For example, we have observed a strong prevalence of Android-based approaches, a lack of contributions from industry, and the absence of specific venues and journals focused on mobile testing automation.

Keywords Mobile applications · Testing automation · Functional testing · Systematic mapping

✉ Porfirio Tramontana
ptramont@unina.it

Extended author information available on the last page of the article.

1 Introduction

The widespread diffusion of smartphones and other mobile devices makes the mobile applications market very dynamic and profitable. The quality and, in particular, the reliability of such applications may be key factors that determine their success. The rapidity with which mobile applications have to be evolved to maintain their appeal and to be adapted to the characteristics of new devices makes testing and quality assurance very important and critical activities.

Manual testing of mobile applications may be a very costly activity both in terms of time and resources, as well as an extremely boring, repetitive, and error-prone activity. For example, there is a large fragmentation of mobile systems and devices, as witnessed by the recent study of OpenSignal that has found in August 2015 the existence of more than 24,000 different types of devices supporting Android[1]. A consequent issue is related to the need to repeat the same tests on a very large number of different devices and execution environments. Not surprisingly, a growing interest in mobile testing automation techniques and tools has been demonstrated by the industry. For example, as regards the Android framework, three technologies supporting testing activities have been developed and distributed since the first versions of the framework in 2008 (i.e., the Monkey tool, which is capable of automatically triggering random event sequences, the MonkeyRunner scripting language, and the InstrumentationTestCase library by means of which the tester can implement automatically executable test cases), and they have contributed to the diffusion of the Android framework. Moreover, both Google and Amazon have recently released some cloud services supporting the automated testing of Android applications (i.e., Android Robo Test from Google[2] and the built-in Fuzz Test from Amazon[3]).

A great interest in mobile testing automation has been recorded in the literature, too. The first papers related to testing automation of Symbian applications have been published in 2006 (Delamaro et al. 2006), while the first paper related to Android smartphones dates back to 2010 (Liu et al. 2010a) and the first secondary papers discussing challenges, approaches, and future directions of mobile testing automation have been published in 2012 and 2013 (Muccini et al. 2012; Amalfitano et al. 2013b; Dubinsky and Abadi 2013; Kirubakaran and Karthikeyani 2013).

Nowadays, there is a large fragmentation of papers focused on different aspects of mobile testing automation including functional testing, security testing, usability testing, context-awareness testing, and energy efficiency assessment (Zein et al. 2016). This fragmentation, together with the continuous proposal of new techniques and prototypes of tools, makes it difficult for researchers and practitioners to have a clear view of the state of the art in mobile testing automation.

Systematic mapping studies represent a well-known mean to shed light on a wide area of research by systematically classifying all the contributions in literature with respect to a given set of categories. According to Petersen et al., a software engineering systematic map is a defined method to build a classification scheme and structure a software engineering field of interest (Petersen et al. 2008). Systematic mapping studies are different from systematic literature reviews that are focused on a more qualitative review of the contributions

[1] https://opensignal.com/reports/2015/08/android-fragmentation/

[2] https://firebase.google.com/docs/test-lab/robo-ux-test

[3] http://docs.aws.amazon.com/devicefarm/latest/developerguide/test-types-built-in-fuzz.html

found in literature. From this point of view, systematic mapping may represent the starting point for systematic literature reviews (Kitchenham et al. 2009).

In this paper, we present a systematic mapping study centered on techniques and tools supporting the automation of functional testing activities on mobile applications.

According to ISO 29119 Software Testing Standard (2013), testing of the functional characteristics of the software under test is referred as functional testing, while testing of the other quality characteristics is referred as non-functional testing. In this study, we intend as functional testing all the activities related to the verification of the correct execution of the whole application under test or of some of its parts, with respect to its functional requirements. It has been distinguished by non-functional testing that is driven by the quality requirements of the applications, such as its security, privacy, usability, performance, and energy efficiency.

There are two main reasons behind the selection of this specific research area in this study. The first reason is the relative lack of similar studies in literature.

In fact, some secondary papers in literature deal with broader research areas, such as the study of Holl and Elberzhager (2016) that focused on quality assurance of mobile applications; the one of Zein et al. (2016) that deals with security, privacy, usability, and context awareness testing (and found only 29 works directly related to testing automation); the one of Sahinoglu et al. (2015) that considers both functional and non-functional testing activities; and the one of Ahmad et al. (2018) that considers development challenges (including testing) related both to native mobile applications, to mobile web applications, and to hybrid applications. Other secondary studies such as the one of Corral et al. (2015) and the one of Mendez-Porras et al. (2015b) are updated to 2013 and 2014, respectively. The work of Mendez-Porras et al. (2015b) is the most similar to ours since its specific topic is mobile testing automation. With respect to this paper, we have followed a more accurate methodology and to perform a more detailed analysis and classification of the contributions in literature, taking into account the papers published in the last 3 years, too.

The second reason is the fundamental importance of testing automation activities in the context of mobile applications. In fact, even the automation of a single testing task, such as the generation of test cases, their execution or the oracle evaluation may produce a remarkable reduction of the costs of the testing process (Crispin and Gregory 2009). Consequently, testing automation may make feasible the execution of complex and effective testing processes that are instead too much expensive for manual testing approaches. In addition, the automation of functional testing activities may enable the automation of quality assurance activities, as it can be observed in several works in literature. Automatically generated functional test suites can be reused in the context of compatibility testing of mobile applications with respect to different combinations of devices and operating system versions (Vilkomir and Amstutz 2014; Vilkomir et al. 2015; Zhang et al. 2015a). Another example is represented by the approach proposed by Behrouz et al. (2015) that exploits test cases generated by static and dynamic analysis techniques (including random testing techniques) to explore the execution scenarios of an Android application and to measure the energy consumption. Finally, Canfora et al. (2013) have developed a system to measure some user experience parameters on real devices by exploiting techniques supporting the automatic execution of test cases.

The systematic mapping study presented in this paper has been carried out on the basis of the guidelines proposed by Petersen et al. (2008). First of all, four goals have been formulated that aim at (1) the classification of the works in literature in terms of their support to testing automation, (2) the evaluation of the characteristics of the proposed techniques and tools, (3) the classification of the proposed techniques and tools in terms of how they have

been evaluated and compared with the state of the art, and (4) the identification via bibliometric analyses of the most prolific authors, institutions, and countries, the most influential publications, and the venues and journals that more frequently have included papers related to this topic. By means of the GQM approach (Basili et al. 1994), a number of research questions and metrics related to the proposed goals have been defined.

The literature research has been carried out on seven different search engines with a set of search queries that have been designed by improving the ones proposed by previous secondary works in literature and validated with respect to a set of relevant papers. The 4509 papers retrieved by search engines have been filtered on the basis of inclusion and exclusion criteria obtaining a set of 131 relevant articles. Each of these articles has been analyzed in depth by the authors in order to extract the information necessary to fill the map. The systematic map has been analyzed in order to provide a detailed description of state of the art in this research field with its emerging trends and existing gaps. The systematic map is available online at https://goo.gl/678T5P for validation purposes: our intent is to periodically update it in the future.

The remainder of the paper is structured in the following way: Section 2 provides a survey of the secondary works in literature that are related to the mobile testing automation area; the adopted research methodology is described in detail in Section 3, while the results of the systematic mapping study are presented in Section 4. Threats to the validity of the presented study are discussed in Section 5 while further discussions about the way to identify the most relevant contributions found in literature, the emerging trends, and the current research gaps are included in Section 6. Finally, conclusions are reported in Section 7.

2 Related work

Several works in the last years have studied challenges, research directions, and trends in mobile application testing, usually based on informal surveys of existing literature and tools.

In particular, Muccini et al. (2012) have summarized the main challenges and research directions in mobile testing automation, while Amalfitano et al. (2013b) have provided a wider view on this field in 2013, by focusing on open issues from different testing perspectives. Another similar work has been presented in 2013 by Kirubakaran and Karthikeyani (2013), that discussed about challenges and solutions provided by testing automation techniques. Also in 2013, Dubinsky and Abadi (2013), in the context of the Workshop on Mobile Development Lifecycle, have collected from participants a list of 45 challenges that they have organized in three research questions for planning future researches on mobile testing automation. Arzensek and Hericko (2014) and Saad and Awang Abu Bakar (2014) in 2014 have focused their contributions respectively on the criteria to select the characteristics of a mobile application testing tool and on the features of the existing commercial testing tools. Gao et al. (2014) have provided a large overview of mobile testing and tools up to 2014. Good surveys of mobile applications automated testing techniques are also included in the related work sections of some recent papers (Moran K et al. 2016; Mao et al. 2016). Finally, in 2018, Ahmad et al. (2018) have identified the challenges of both native, web, and hybrid mobile applications by means of an empirical study and have found how testing is still a relevant challenge for each type of mobile applications. A different perspective has been considered by Kochhar et al. (2015) and by Silva et al. (2016) that have studied how mobile application testers try to automate their work and what their needs.

Recently, secondary works addressing the problem of the comparison of the performance of techniques and tools for mobile testing automation have been published. Choudhary et al.

(2015) in 2015 have presented an empirical comparison of the performance of some of the most popular mobile testing automation tools available in literature, while Amalfitano et al. (2015b, 2017) and Jiang et al. (2017) have presented empirical comparisons focused on different testing techniques implemented in the context of the same testing tool.

Several works have addressed specific topics in the area of mobile testing automation. For example, Harrison et al. (2013) have proposed a literature review specifically focused on mobile usability testing, while Li et al. (2016a) have recently proposed a technical report with a systematic literature review of techniques and tools for static analysis of Android apps.

Several other works have addressed research fields that are wider than the one addressed in our study. Corral et al. (2015) have presented a systematic mapping updated to 2012, focusing their attention on development practices helping testing and quality assurance of mobile applications. More recently, Holl and Elberzhager (2016) have presented a systematic mapping regarding quality assurance of mobile applications. They have provided answers to seven research questions regarding the approaches found in literature supporting both testing and quality assurance of mobile applications, including automatic and manual approaches, static and dynamic analysis, and functional and non-functional testing. The study have selected 230 papers from Scopus, ScienceDirect, IEEE, and ACM, published up to 2015.

Another recent systematic mapping study is the one presented by Zein et al. (2016). This work is focused on a wide spectrum of testing techniques, including studies on security, privacy, usability, and context awareness testing by including 79 relevant papers after a very selective process. In particular, they have selected a subset of 29 papers related to mobile testing automation. Another similar study is the one of Sahinoglu et al. (2015) that in 2015 have presented a systematic mapping including both functional and non-functional mobile testing approaches published up to 2014. They have included 123 papers and their research questions have studied at a coarse level of detail the typologies of testing activities focused on by the selected papers.

With respect to the studies (Corral et al. 2015; Holl and Elberzhager 2016; Sahinoglu et al. 2015), we have restricted the research field to mobile functional testing automation. In this more focused context, we have designed questions addressing specific aspects related to mobile functional testing automation. With respect to the study of Zein et al. (2016) that considers functional testing automation besides other different types of mobile testing and quality assurance activities, we did not select only papers providing an empirical evaluation of the proposed techniques and tools. We included also papers providing a demonstration of the feasibility of the proposed techniques.

The unique systematic mapping study related to the specific field of automation of functional testing for mobile applications is the one of Mendez-Porras et al. (2015b). This work addresses several general questions on the basis of a selected subset including 83 papers up to 2014, such as bibliometric analyses of authors, journals, and venues, the challenges of automated testing of mobile applications, the proposed techniques and approaches, and the adopted evaluation methods. This work has represented a good starting point for our research. In particular, with respect to this work, we have improved the investigation protocol by performing an objective validation of the proposed queries, by widening the search to other search engines such as ACM and Google Scholar, and by performing a more detailed analysis of the existing works, on the basis of a richer set of research questions.

3 Research methodology

The research methodology followed in this study is based on the guidelines provided by Petersen et al. (2008, 2015) and Kitchenham and Charters (2007).

The process followed in this paper is composed of six sequential steps. Each step includes a list of tasks that have been executed sequentially and each task has produced some outputs. Figure 1 shows the steps, the tasks, and the main outputs of the process.

The first step consists of the definition of the research questions that will guide the process: to this aim we have adopted the Goal/Questions/Metrics paradigm. At the end of this step, a candidate classification scheme has been designed.

In the second step, the strategy for searching relevant papers in literature has been defined. Firstly, a set of sources of evidence has been selected, then a set of queries on these sources has been defined, executed, and validated. The results of the execution of these queries represent the initial set of candidate papers.

In the third step, two sets of inclusion and exclusion criteria have been defined to filter the studies that are relevant to this systematic mapping.

The fourth step is devoted to the screening of papers and to the selection of the ones relevant for the topic addressed by this study. It consists of a preliminary elimination of duplicated studies followed by the Keywording of Abstracts task. In the execution of this task, the title, keywords, and abstract of each paper have been analyzed in order to evaluate which papers can be excluded since they do not satisfy all the inclusion criteria or they satisfy some of the exclusion criteria.

In the Data Extraction and Mapping step, the filtered set of papers has been analyzed in detail by reading each of them. The papers that satisfy the above-defined criteria have been included in the study and the data needed to fill the classification scheme have been collected. The final systematic map has been obtained at the end of this step.

Finally, analyses and discussions of the metrics evaluated on the systematic map have been carried out and reported in this paper that represent the output of this last step.

Details about the execution of each step of the systematic mapping process will be reported in the next subsections.

3.1 Definition of the research questions

This study builds a classification scheme of the works in literature related to the proposal and the evaluation of techniques and tools supporting the automation of functional testing activities in the context of mobile applications.

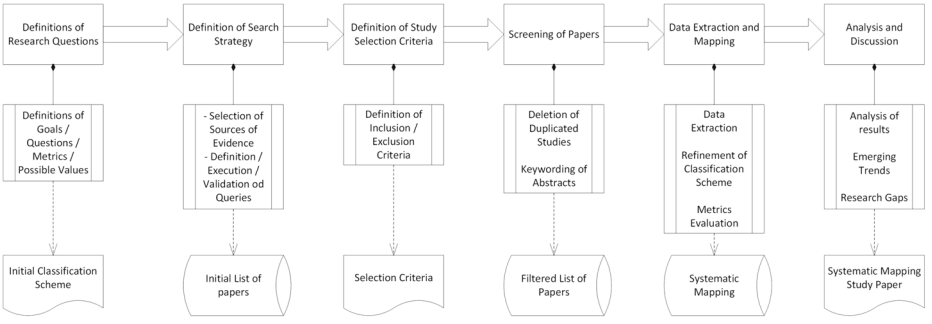

Fig. 1 The systematic mapping process

In order to express this objective in terms of research questions and to link them to metrics the Goals/Questions/Metrics (GQM) paradigm originally proposed by Basili et al. (1994) has been applied. This methodology has been used both to formulate the research questions and to define the metrics needed to map the studies. A detailed description of the proposed goals, questions, and metrics follows.

3.1.1 Goals

The four goals of this study are:

G1 To classify the articles in the area of mobile application functional testing automation on the basis of the offered support to testing automation and of the addressed testing levels.
G2 To study in detail the characteristics of the proposed techniques and tools, such as the enabling inputs, their support to the generation of test cases, the generated test outputs and the supported mobile frameworks.
G3 To study how the proposed techniques and tools have been evaluated in terms of the characteristics of the experiments that have possibly been carried out, the involved applications under test, and the comparisons with other techniques and tools.
G4 To identify the most active researchers in this area and their affiliations, the most attractive venues and journals for papers in this field, and the most influential papers.

3.1.2 Research questions

For each of the four proposed goals, a set of specific questions have been formulated. With respect to the first goal (G1), the following questions have been posed:

RQ 1.1 What testing activities are automated?
RQ 1.2 What testing levels are addressed?

The first question is aimed at classifying the selected papers with respect to the degree of automation they provide to the testing process. In particular, we have evaluated if the techniques and tools proposed in the considered articles support the automation of test case design and implementation, test case execution, and oracle definition and evaluation. The second question aims at the classification of the papers according to the addressed testing level (i.e., unit testing, integration testing, or system testing).

The second goal (G2) has been addressed by the following seven research questions:

RQ 2.1 What inputs are used by the proposed testing techniques to derive test artifacts?
RQ 2.2 What kinds of techniques are proposed for test case generation?
RQ 2.3 What kinds of test oracles are considered?
RQ 2.4 What kinds of test artifacts are generated?
RQ 2.5 What are the characteristics of the proposed testing tools?
RQ 2.6 Which mobile frameworks are the targets of the proposed techniques and tools?
RQ 2.7 Are the proposed techniques and tools usable on emulators or real devices?

This set of questions aims at the detailed analysis of the main characteristics of the proposed techniques and tools. In particular, RQ 2.1 focuses on the inputs needed to enable the proposed techniques and tools such as source code, executable code, high-level models, existing test cases, or user sessions, while RQ 2.2 focuses on the proposed test case

generation techniques, if any. The questions RQ 2.3 and RQ 2.4 respectively examine the considered test oracles, if any, and the outputs generated by the application of the proposed testing techniques. RQ 2.5 has been posed to collect information about the characteristics of the proposed testing tools (if any), such as their names, their dependencies on other external tools or resources, their availability (e.g., open source, free downloadable, commercial or not available), the type of technique adopted for test case generation (i.e., static analysis, dynamic analysis or hybrid, i.e., that combines static and dynamic analyses), the languages used for the tool implementation, the targeted execution framework, and the date of their last update. The questions RQ 2.6 and RQ 2.7 are directed to the technological characteristics of the proposed techniques and tools, since they express the technological scope (i.e., Android, iOS, Windows Phone, or others) and the possibility to apply them to real devices, emulated devices or both.

The third goal (G3) has been pursued by analyzing in detail the evaluation experiments carried out to validate the proposed techniques and tools and to compare their results with the ones obtained by using other similar tools. In particular, the following three questions have been posed:

RQ 3.1 What are the characteristics of the performed evaluation studies?
RQ 3.2 What are the characteristics of the sets of applications objects of the evaluation experiments?
RQ 3.3 What are the characteristics of the performed comparative studies?

Question RQ 3.1 is focused on the evaluation studies performed to validate the proposed techniques and tools. We have distinguished between papers providing experimental studies aiming at the evaluation of the performance of the proposed techniques or tools and papers providing only a demonstration of the feasibility of the proposed technique or tool. The question RQ 3.2 investigates the type and the quantity of the applications considered by the evaluation experiments reported in the selected papers. Finally, question RQ 3.3 regards the characteristics of the comparative studies that possibly have been carried out to compare the performance of the proposed techniques and tools with the ones of other tools considered as benchmarks.

Finally, the goal G4 has been investigated by posing some questions related to the demographics and bibliometrics of the selected articles and authors. The following questions have been formulated:

RQ 4.1 What is the number of published articles per year?
RQ 4.2 Which are the venues having the higher article counts?
RQ 4.3 Which are the more influential articles in terms of citation counts?
RQ 4.4 Who are the authors with the higher number of articles?
RQ 4.5 Which countries have produced more articles?
RQ 4.6 Which are the author affiliations?

The first question aims at the investigation of the trend of interest of the scientific community over the years with respect to the studied topic, while the second question summarizes the venues and journals that more frequently host contributions in this field. The third question aims at the evaluation of the papers that have had more influence on the literature by taking into account the number of works citing them. The fourth question has been posed to individuate the more prolific authors. Finally, the last two questions classify the papers in terms of the country of work of the authors and of their affiliation (academia or industry).

3.1.3 Metrics

In order to support the evaluation of the research questions, for each of them an attribute or a list of sub-attributes has been formulated. For each attribute and sub-attribute, a set of possible values has been defined.

These lists of attributes and possible values have been just sketched in the first step of the process leaving the possibility to refine them during the next steps. In particular, these lists have been modified during the Data Extraction and Mapping step and finalized at the end of that step. Table 1 reports, for each research question, the set of attributes and sub-attributes that have to be evaluated for each selected paper and the list of possible values for each of these attributes. For the sake of brevity, we have reported only the final version of this list. In Table 1, the sub-attributes have been written in italic.

The attributes have been designed in order to admit zero or more possible values for each paper. In fact, some attributes are not applicable to each paper (e.g., the tool characteristics are applicable only on papers presenting a tool).

In order to provide answers to the proposed research questions, for most of the considered attributes, we have considered and automatically evaluated the metric consisting in counting the occurrences of each value assigned to each attribute.

3.2 Search strategy definition

The search strategy adopted in this paper is inspired by the one proposed by Kitchenham and Charters (2007) and recently adopted by Zein et al. (2016).

In detail, (1) we have selected a set of sources of evidence (i.e., online available search engines), (2) we have selected a set of keywords able to drive the search, (3) we have formulated a set of tentative search strings including the selected keywords, (4) we have executed the proposed queries on the considered search engines, and (5) we have validated the results of these queries. In order to perform this validation, we have preventively built a list of papers that we consider relevant for the proposed topic and we have evaluated the ability of the formulated queries in retrieving the papers belonging to this list. The last three operations have been repeated until the tentative queries have been able to retrieve all the papers of this list.

The set of *sources of evidence* has been built by considering the online databases that index Computer Science literature and that have been often considered in other systematic mapping studies, too. In particular, the search engines that have been considered are as follows: Scopus[4], IEEExplore[5], ACM[6], SpringerLink[7], ISI Web of Knowledge[8], ScienceDirect[9], and Google Scholar[10].

In general, it is expected that the most part of the papers retrievable by using IEEExplore, ACM, SpringerLink, ISI, and ScienceDirect are retrievable by using Scopus, too, because Scopus has been designed to be an aggregator of all the contributions available

[4] https://www.scopus.com/
[5] http://ieeexplore.ieee.org/Xplore/home.jsp
[6] http://dl.acm.org/advsearch.cfm
[7] http://link.springer.com/
[8] https://webofknowledge.com/
[9] http://www.sciencedirect.com/
[10] http://scholar.google.com

Table 1 Final classification scheme reporting attributes and sub-attributes that have been evaluated on each selected paper and possible values for the attributes

RQ	Attribute	Sub-attributes and possible values
1.1	Support to Testing Automation	Test Case Generation - Test Execution - Oracle Definition / Evaluation
1.2	Testing Level	Unit Testing - Integration Testing - System Testing
2.1	Testing Input	Source Code - Bytecode / Executable - User Sessions - Manually Inferred Models - Automatically Inferred Models - Existing Test Cases - Bug Repository
2.2	Test Case Generation Technique	Model Based - Model Learning / Active Learning - User Sessions Based Capture and Replay - Random - Search Based - Mutation - Manual
2.3	Oracle	Crashes / Exception Occurrence - Expected Bitmap - Expected GUI State Invariant Condition - Manually Designed Assertions
2.4	Generated Test Artifact	Test Input Values - Executable Test Cases - Oracles - Application Models - Test Reports
2.5	Attributes of the Testing Tool	*Tool Name - Names of 3rd Party Included Components Tool Availability* (Open Source - Free Download - Demo Version - Commercial Version - not available)
		Test Generation Technique (Static Analysis - Dynamic Analysis - Hybrid)
		Implementation Language - Targeted Framework - Time of Last Update
2.6	Supported Mobile Framework	Android - iOS - Windows Phone - Symbian / J2ME
2.7	Testing Platform	Emulator - Real Device
3.1	Evaluation	*Evaluation Method* (Experimental - Demonstration of Feasibility) *Evaluation Metric* (Code coverage - Injected Faults Found - Real Failures Found)
3.2	Attributes of the Set of AUTs Involved in the Evaluation	*Number of AUTs - AUTs Size* (Toy Examples - Small Apps - Large Apps) *AUTs Availability* (Open Source Apps - Real Commercial Apps)
3.3	Attributes of the Performed Comparative Studies	*Names of the Compared Tools-Comparison Attributes*(Failure Detection Capability - Code Coverage Capability - Offered Features)
4.1	Publication Year	
4.2	Publication Journal or Venue	
4.3	Number of Citations of the Paper	
4.4	Publication Authors	
4.5	Publication Countries of Authors	
4.6	Affiliation of Authors	*Name of the Institution - Type* (Industry - Academia)

from the most relevant publishers. Google Scholar, instead, has a larger scope, since it has been designed to index any content available on the web. For this reason, a larger number of contributions is retrievable by Google Scholar but the relevance of these contributions should be accurately evaluated.

After the selection of the sources of evidence, a set of tentative search strings has been built by selecting popular keywords used in this field. We have selected these keywords both on the basis of our specific knowledge of the research field and on the keywords used by

similar systematic mapping studies in literature (i.e., Mendez-Porras et al. 2015b; Zein et al. 2016; Sahinoglu et al. 2015; Holl and Elberzhager 2016). Synonyms and other alternative keywords have been considered, too. The boolean operator OR has been used to consider different synonyms while the boolean operator AND has been used to link the different keywords. Since each search engine supports a different syntax for queries and provides different options to filter the search, different search strings have been formulated taking into account the peculiarities of the search engines.

Four main keywords have been considered: "mobile applications" (having as popular hyponyms the keywords "Android applications," "apps," "iOS applications," "Symbian applications," "Windows Phone applications"), "testing," "technique" (for which different synonyms and hyponyms have been considered, such as "approach," "method," "tool," and "framework"), and "automation" (and its synonyms "automated" and "automatic").

The proposed search strings are conceptually equivalent between them. For all the engines, the searches have been restricted to title, abstract, and keywords of each indexed paper. In addition, the search on Scopus has taken into account, too, the titles of the papers referenced in the bibliography section of each paper. On Scopus and ScienceDirect, the search has been restricted to the Computer Science field to reduce the set of results. The possibility of extending the search to the entire text of the papers is available only in some search engines such as IEEExplore and ACM, but has been discarded in order to limit the wideness of the set of results (when the search has been extended to the full text of the papers, the same search strings returned 8,931 results for IEEExplore and 397,267 results for ACM). For Google Scholar, instead, the search regards always the full text of the papers. Google Scholar estimated that the number of returned results was higher than 56,000 but we have limited the analysis to the first 1,000 results that are the more relevant ones according to the Google Scholar ranking algorithm.

In order to validate the proposed search strings, we have selected a set of papers that have been judged relevant for this systematic mapping study. This set includes 55 papers recently cited by recent secondary studies in literature (Zein et al. 2016; Choudhary et al. 2015), in the related work sections of some recent and influential papers (Moran K et al. 2016; Mao et al. 2016) or in the Ph.D. thesis of one of the authors (Amatucci 2016). The other authors have preventively read these papers and have confirmed that they should be included in this study for their relevance.

Table 2 shows the final set of search strings that have been formulated for the seven considered search engines. In particular, the third column of this table reports the number of papers from this set of 55 relevant papers that have been retrieved, too, by each of the considered search engines, while the total number of papers retrieved by each query is shown in the last column. The last row of the table shows that all the 55 relevant papers have been found by at least one search engine and that the total number of retrieved papers (including duplicates) is 4509.

The queries to the search engines have been carried out on September 1, 2017, and report only the papers indexed at that date. The results reported in Table 2 show that the Scopus search engine is able to find 54 out of the 55 relevant papers (the paper of Mirzaei et al. (2012) is only retrievable on ACM and Google Scholar), while Google Scholar is able to find 46 out of 55 relevant papers. The remaining search engines have retrieved smaller subsets of relevant papers because they index only papers from a subset of publishers. Since the selected search strings have demonstrated their ability to retrieve all the 55 selected papers in at least one of the considered search engines, they have been considered valid for this systematic mapping study.

Table 2 The search strings that have been executed on the different search engines and the obtained results (the queries have all been executed on September 1, 2017)

Search engine	Search string	No. of retrieved relevant papers	No. of retrieved papers
Scopus	ALL (((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))) AND (LIMIT-TO (SUBJAREA,”COMP”))	54	2409
IEEExplore	((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))	22	192
ACM	((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))	24	212
SpringerLink	((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))	0	460
ISI	TOPIC: (”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND TOPIC:(testing) AND TOPIC: (technique OR approach OR method OR tool OR framework) AND TOPIC: (automation OR automatic OR automated)	22	167
ScienceDirect	tak(((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))){[} All Sources(Computer Science){]}.	1	69
Google Scholar	((”mobile applications” OR ”Android applications” OR apps OR ”Windows Phone applications” OR ”Symbian applications” OR ”iOS applications”) AND testing AND (technique OR approach OR method OR tool OR framework) AND (automation OR automated OR automatic))	46	1000
Total		55	4509

The search strings reported in Table 2 have been selected after several trials characterized by worst performance in terms of retrieval of the set of 55 relevant papers. For example, the same search on Scopus limited only to title, abstract, and keywords fields returned 209 results including only 26 out of 55 expected papers.

Differently from our search strings, the more general search strings proposed by Zein et al. (2016) have been able to retrieve only 49 of the 55 relevant papers on Scopus, but with a total number of retrieved papers that is more than 10,000. The search strings proposed by the similar work of Mendez-Porras et al. (2015b) are able to find only 50 out of 55 relevant papers on Scopus and are not executable on some search engines such as IEEExplore, due to the current limitation of 15 keywords per query.

3.3 Study selection criteria definition

The purpose of the inclusion and exclusion criteria is to limit the study selection to papers that fit in the proposed topic, i.e., techniques and tools for the automation of functional testing of mobile applications and that are available in scientific literature. To this aim, a set of inclusion criteria useful to identify studies that could be considered in this mapping has been designed. In addition, another set of exclusion criteria has been formulated in order to exclude studies related to other fields of interest or not specifically focused on the selected topic.

In details, the following list of *inclusion criteria* has been considered:

1. Studies must be directly related to automated software testing techniques for native mobile applications.
2. Studies must be focused on functional testing of mobile applications, including, too, system testing, unit testing, integration testing, or any other testing activity aiming at the verification of the functional correctness of the application.
3. Studies must provide a qualitative or a quantitative evaluation of the proposed contributions.

In order to filter the set of papers from off topic ones, the following list of *exclusion criteria* have been defined:

1. Studies focused on testing embedded systems in general, and not directly related to mobile devices.
2. Studies focused on mobile communication infrastructure, mobile hardware, or robotics.
3. Studies focused on testing mobile applications different from native applications, such as mobile web applications.
4. Studies focused on other testing or quality assurance techniques, such as security testing, performance testing, energy consumption evaluation, usability testing, and compatibility testing.
5. Studies focused on static analysis without support to testing automation.
6. Studies related to other software development phases such as analysis, design, or implementation and not focused on testing.
7. Studies that merely present opinions or ideas without any proposed testing technique and any implemented testing tool.
8. Studies written in languages other than English or not available on the Internet in full-text form.
9. Studies that did not appear in the published proceedings of a peer reviewed conference, symposium, or workshop, or did not appear in a journal or magazine (i.e., thesis, technical reports, patents, blogs, or personal web pages).
10. Studies that are duplicates of other studies.
11. Surveys, reviews, mapping studies, and any other secondary study.

3.4 Screening of papers

During the execution of the Screening of Papers step of the process, the set of papers retrieved by the search engines has been progressively reduced by filtering duplicated and irrelevant papers.

A first filtering of the 4509 returned results consisting of the automatic elimination of duplicates, i.e., the elimination of all but the first entry of each paper retrieved in more than one search engine.

After this task, 3810 papers remained, as shown by the third column of Table 3. A majority of this papers (2409) are indexed by Scopus that is the first considered search engine. For example, the 86 papers in the IEEExplore row are articles that have been retrieved by IEEExplore but that have not been retrieved by Scholar.

The filtering of the remaining papers has been executed by means of the Keywording of Abstracts task based on the inclusion and exclusion criteria presented in the previous subsection. In detail, the authors have analyzed title, abstract, and keywords (where available) of the 3810 considered papers and have filtered out all the articles that do not satisfy all the inclusion criteria or that satisfy one or more exclusion criteria. All the borderline papers for which this analysis has not been sufficient to include or exclude them have not been excluded in this step. After the execution of this screening activity, 351 papers remained. The fourth column of Table 3 reports the number of papers remaining, grouped for search engine.

The papers retrieved by IEEExplore, ACM, ISI, and ScienceDirect are generally very recent papers (not yet indexed by Scopus) or papers published in venues not covered by Scopus. The 105 papers retrieved only on Google Scholar correspond to papers not indexed by the other search engines or to papers retrievable only with a full-text research.

3.5 Data extraction and mapping of studies

The Data Extraction and Mapping step has been performed by the authors by completely reading the full text of the 351 selected papers.

The papers to be analyzed have been divided between the authors and each author evaluated if the paper should be included or excluded from the systematic mapping on the basis of the study selection criteria. In addition, for all the papers resulting from this filtering,

Table 3 Number of selected papers after the different steps of the systematic mapping process

Search Engine	Number of papers retrieved from search engines	Number of papers remaining after elimination of duplicates	Number of papers selected after Keywording of Abstract	Number of selected papers
Scopus	2409	2409	218	108
IEEExplore	192	86	12	0
ACM	212	84	11	6
SpringerLink	460	436	0	0
ISI	167	27	5	1
ScienceDirect	69	61	0	0
Google Scholar	1000	707	105	16
Total	4509	3810	351	131

the authors have assigned values to each of the attributes in Table 1, when applicable. The authors have had some joint meetings in order to discuss about the inclusion or exclusion of all the borderline papers and to review the values assigned to the attributes. Only the demographic and bibliometric data requested by the fourth goal of the study have been automatically extracted from the data exported from the search engines without any need for reviews or joint discussions.

At the end of this step, a set of 131 papers has been selected. The last column of Table 3 reports the final number of selected papers for each considered search engine.

3.6 Data availability

The Systematic Mapping reporting the complete list of the selected papers and all the values assigned to each attribute of each paper is not reported here for reasons of space, but it is available online at https://goo.gl/678T5P.

4 Analysis of results

The data extracted and collected during the previous steps have been aggregated in order to provide answers to the proposed research questions. In the following, the results obtained from the study and the answers that can be given to each research question will be presented and discussed.

4.1 RQ 1.1 What testing activities are automated?

Only 23 out of 131 papers provide fully automated testing processes including automatic test case generation and execution and automatic generation and evaluation of test oracles (Adamsen et al. 2015; Amalfitano et al. 2012a, 2015d; Costa et al. 2014; Hao et al. 2014; Hu et al. 2016; Imparato 2015; Liang et al. 2014; Liu et al. 2016; Maji et al. 2012; Mao et al. 2016; Mendez-Porras et al. 2015a; Mirzaei and Heydarnoori 2015; Moran K et al. 2015; Packevicius et al. 2015; Takala et al. 2011, White et al. 2015; Liu et al. 2014a; Zaeem et al. 2014; Zhu et al. 2015; Li et al. 2016b, 2017; Fazzini et al. 2016b).

Of the remaining papers, 63 present techniques and tools that are able to automatically generate and execute test cases but that do not provide any support to the automatic definition or evaluation of oracles.

On the other hand, we have found 32 papers that support the automatic test case execution and the oracle definition and evaluation (in most cases, the occurrence of crashes and exceptions has been evaluated).

In addition, we have found 13 papers that provide automation only for a single activity of the testing process. Five of them provide support only to the automatic generation of test cases that are not directly executable. In particular, in Puspika et al. (2015), Zheng et al. (2017), and Shabaan et al. (2017), model-based techniques are proposed, while Yu and Takada (2016) describe an approach based on the generation of external events and Liu et al. (2017) propose an approach exploiting machine learning techniques.

Seven other papers are focused on the automatic test execution. For example, in the paper of Griebe et al. (2016), the problem of executing test cases on speech-based applications is faced. Other studies are related to the automatic test execution in the context of Windows mobile applications (Mayan et al. 2015), Symbian applications (She et al. 2009; Jiang et al. 2007), or cloud infrastructures (Prathibhan et al. 2014). Finally, the works of Sadeh (2011)

and Liu et al. (2014b) propose techniques for the automatic execution of unit test cases in the context of Android applications.

Finally, there is only one paper completely devoted to oracle evaluation. In the approach presented in Hsiao et al. (2014), the Android framework has been instrumented so that logs of the executions of the Android applications under test are generated while it is exercised by real users. These logs are automatically analyzed in order to detect concurrency races.

Table 4 reports the total list of the references of the 131 papers retrieved by the study, classified on the bases of the support offered to the testing activities.

Table 4 List of selected papers, grouped by their support to test automation

	Retrieved papers
Fully automation	Adamsen et al. (2015), Amalfitano et al. (2012a, 2015d), Costa et al. (2014), Hao et al. (2014), Hu et al. (2016), Imparato (2015), Liang et al. (2014), Liu et al. (2010a), Maji et al. (2012), Mao et al. (2016), Mendez-Porras et al. (2015a), Mirzaei and Heydarnoori (2015), Moran K et al. (2016), Packevicius et al. (2015), Takala et al. (2011), White et al. (2015), Liu et al. (2014a), Zaeem et al. (2014), Zhu et al. (2015), Li et al. (2016b, 2017), Fazzini et al. (2017)
Test case generation and execution	Anbunathan and Basu (2015), Baek and Bae (2016), Bielik et al. (2015), Do et al. (2016), Gomez (2015), Hu and Neamtiu et al. (2011a, b, 2014, 2015, 2016), Jaaskelainen et al. (2012), Joorabchi et al. (2016), Lin et al. (2014), Liu et al. (2010b, 2013), Ma et al. (2016), Machado et al. (2013), Machiry et al. (2013), Maiya et al. (2014), van der Merwe et al. (2012), Morgado and Paiva (2016a, b), Raut and Tomar (2014), Ravindranath et al. (2014), Salva and Laurencot (2015), Shan et al. (2016), Song et al. (2015), Tao and Gao (2016), Tang et al. (2016), Ye et al. (2013), Yeh et al. (2014), Zhauniarovich et al. (2015), Salihu and Ibrahim (2016), Moran et al. (2017), Yoo and Lee (2017), Paulovsky et al. (2017), Wu et al. (2016), Arnatovich et al. (2016), Zhang et al. (2016)
Test case execution and oracle definition and evaluation	Amalfitano et al. (2011, 2012a, 2013a, 2015a, c), Anand et al. (2012), Anbunathan and Basu (2016a, b), Azim and Neamtiu (2013), Choi et al. (2013), Coelho et al. (2016), Delamaro et al. (2006), De Cleva Farto and Endo (2015), Dev et al. (2012), Dutia et al. (2015), Gomez et al. (2013, 2016), Gudmundsson et al. (2016), Griebe and Gruhn (2014), Griebe et al. (2015), Hesenius et al. (2014), Jamrozik and Zeller (2016), Jensen et al. (2013), Jha et al. (2015), Jiang et al. (2016), Kaasila et al. (2012), Li et al. (2014a, b), Lin et al. (2014), Linares-Vasquez (2015a), Linares-Vasquez et al. (2015b), Liu et al. (2015, 2014a), Lu et al. (2012), Akanksha Ashok Magare (2016), Mahmood et al. (2014), Majeed and Ryu (2016), Meng et al. (2015), Mirzaei et al. (2012, 2016a, b), Nagowah and Sowamber (2012), Nguyen et al. (2012), Qin et al. (2016a, b), Reddy et al. (2016), San Miguel and Takada (2016), Su (2016), Sun et al. (2016), Wang et al. (2014), Wen et al. (2015), Yang et al. (2013), Yeh et al. (2014), Zeng et al. (2016), Zhang and Pi (2015a), and Zun et al. (2016)
Only test case generation	Puspika et al. (2015), Yu and Takada (2016), Zheng et al. (2017), Liu et al. (2017), and Shabaan et al. (2017)
Only test case execution	Griebe et al. (2016), Jiang et al. (2007), Liu et al. (2014b), Mayan et al. (2015), Prathibhan et al. (2014), Sadeh (2011), and She et al. (2009)
Only oracle definition and evaluation	Hsiao et al. (2014)

Figure 2 shows the distribution of papers according to the support offered to the testing activities.

4.2 RQ 1.2 What testing levels are addressed?

Most of the proposed papers (122 out of 131) present system testing approaches, in which the whole application is tested in its execution environment (real devices or emulators). In particular, there is a large number of GUI-based approaches, in which test cases are defined as sequences of events or interactions acting on the GUI of the application under test.

Only two approaches regard integration testing. The works of Maji et al. (2012) and Jha et al. (2015) are focused on the testing of the interactions between the main components of the application under test (e.g., activities, services, broadcast receivers) by means of specific intent calls.

Seven papers propose techniques and tools supporting unit testing (Delamaro et al. 2006; Sadeh 2011; Mirzaei et al. 2012; van der Merwe et al. 2012; Liu et al., 2014b, 2014c). In the works of Mirzaei et al. (2012) and van der Merwe et al. (2012) and Liu et al. (2014c), specific components of the Android applications are tested in isolation from the target execution environment. In fact, in these three approaches, the application components are tested on the traditional Java Virtual Machine (JVM) instead of on the specific Dalvik Virtual Machine that was mounted on almost all the Android devices at the time of the writing of these papers. The other four papers propose techniques and tools directly supporting unit testing of classes and methods included in Android applications (Delamaro et al. 2006; Sadeh 2011; Liu et al. 2014b; De Cleva Farto and Endo 2015).

A possible reason for which so few approaches to unit and integration testing are available in literature may be that many of the techniques supporting unit and integration testing designed for desktop applications can be reused on mobile applications, too. For example, the JUnit framework can be used to test any component of an Android application that is not dependent on any specific functionality of the targeted mobile device. Thus, there is no need of mobile specific unit testing techniques.

4.3 RQ 2.1 What inputs are used by the proposed testing techniques to derive test artifacts?

Most of the contributions found in literature exploits one or more of the following five sources of information: the source code of the application under test, its executable code (usually bytecode), high-level models, existing test cases, and user sessions.

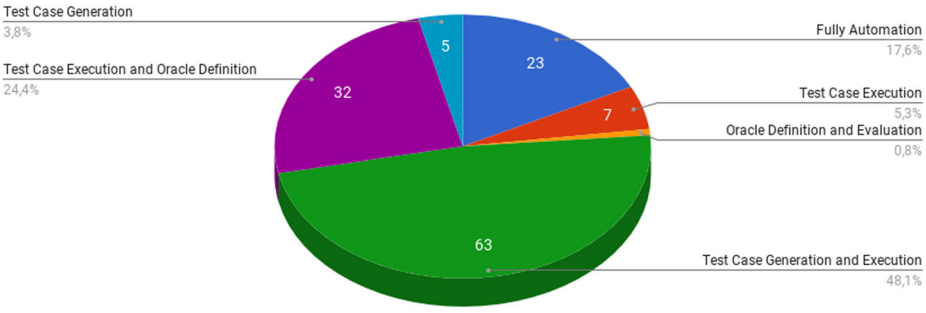

Fig. 2 Classification of papers according to the support to testing automation

There is a substantial equivalence between the number of white box approaches based on the analysis of the source code of the applications under test (53 out of 131) and the number of black box approaches needing only the executable code (57 out of 131).

Black box approaches have been often evaluated on very large sets of free applications available on public markets such as Google Play for Android, while white box approaches have often been evaluated on sets of applications found on public repositories of open-source mobile applications (for example, http://f-droid.org for Android applications).

Forty-seven approaches are based on the analysis of high-level models of the application under test. In particular, 20 papers are based on manually designed models of the application under test (including, for example, finite state machines (Nguyen et al. 2012; Majeed and Ryu 2016; Su 2016), sequence diagrams (Anbunathan and Basu 2016b), activity diagrams Griebe et al. 2015; Li et al. 2014a). Tweny-seven other approaches are instead based on models that are automatically generated by reverse engineering processes, such as GUI trees (Wang et al. 2014; Wen et al. 2015).

The 23 approaches based on existing user sessions include the 12 ones proposing capture and replay techniques able to collect and re-execute user sessions. The other ones are generally able, too, to transform existing user sessions in executable test cases. Usually, basic user events are considered by these approaches, but there are some papers specializing in the identification and generation of complex gesture events typical of mobile devices, such as the one of Hesenius et al. (2014).

Other 23 contributions are based on the transformation of existing test cases, that are often in the form of executable JUnit test cases for Android applications. Finally, there is only a preliminary contribution based on information obtained from bug repositories (Mendez-Porras et al. 2015a) in which only the requirements of a test generation system based on the analysis of a bug repository have been proposed.

The histogram in Fig. 3 reports the distribution of the selected papers with respect to the types of needed input sources.

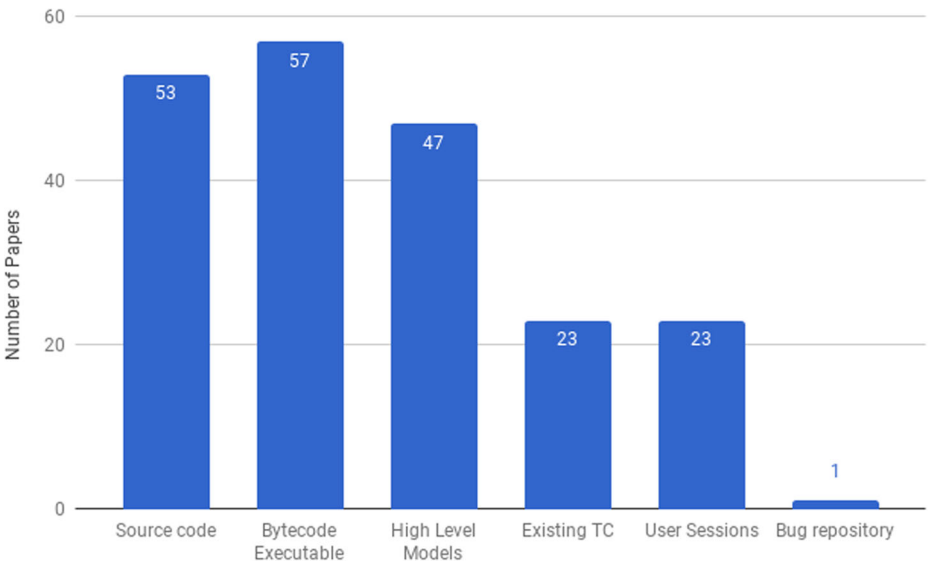

Fig. 3 Types of input sources used by the techniques and tools proposed in the selected papers

4.4 RQ 2.2 What kinds of technique are proposed for test case generation?

The automatic generation of test cases is a fundamental feature for most of the approaches found in literature. Only 14 out of 131 contributions are based on manually written test cases: in these cases, the automation is limited to test case execution or evaluation.

In the approaches presented in 69 of the 131 considered papers, the test case generation is obtained with model-based techniques. These techniques are based on high-level models (i.e., behavioral models such as sequence diagrams, activity diagrams, GUI trees, event flow graphs, finite state machines) or low-level models (i.e., models directly related to the code of the applications under test, such as control flow graphs or call graphs). In addition, in 29 papers, models are automatically generated during the testing process itself (they are usually called active learning techniques; Hao et al. 2015).

In 21 other contributions, test cases have been generated by transforming existing user sessions in executable test cases as shown by the answer to the previous research question.

In only 2 articles, test cases have been obtained by mutating existing test cases (Adamsen et al. 2015; Amalfitano et al. 2013a). Both Adamsen et al. (2015) and Amalfitano et al. (2013a) have injected specific sequences of events in existing test cases reproducing, for example, the closing and restart of the application or the loss of the Internet connection and to test the robustness of the application under test with respect to these events.

In 22 cases, random techniques support mobile application testing. Both uniform random techniques (for example in Choi et al. (2013), Machiry et al. (2013), Hu et al. (2014), and Amalfitano et al. (2015c)) and smarter random techniques (for example in Liu et al. (2010a), Hu and Neamtiu (2011a), Machiry et al. (2013), and Wen et al. (2015)) have been considered for test case generation.

A recent trend appears, the proposal of search-based testing techniques, usually based on genetic algorithms: in this study, we have found five contributions since 2014 (Mahmood et al. 2014; Zhu et al. 2015; Amalfitano et al. 2015a; Mao et al. 2016; Su 2016).

A summary of the techniques used for test case generation is shown in the histogram reported in Fig. 4.

4.5 RQ 2.3 What kinds of test oracles are considered?

As regards the oracle definition, it is generally considered the most difficult phase of the testing process to be automated (Barr et al. 2015). In 66 of the considered papers, no approaches at all for automatic oracle definition and evaluation have been proposed. In addition, in 48 of the remaining approaches, the detection of crashes or exceptions represents the unique implicit way to evaluate the result of the executed test cases.

More specific test oracles have been proposed in few papers. In eight papers, models of the behavior of the application are available from application design or via reverse engineering, and an abstraction of the state of the GUI of the application under test has been proposed. In these cases, it is possible to define the expected GUI state at the end of the execution of each test case. The result of the test case is given by the comparison between the expected GUI state and the state of the GUI that has been obtained (Costa et al. 2014; Hao et al. 2014; Hu et al. 2014; Hu et al. 2015; Salva and Laurencot 2015; Joorabchi et al. 2016; Baek and Bae 2016; Hu et al. 2016). In particular, this approach has been used for the evaluation of the fidelity of the replayed traces in capture and replay techniques (Hu et al. 2014, 2015).

In five contributions, the result of the test is obtained by automatically comparing the screenshot of the current GUI with the expected one that usually has been obtained by

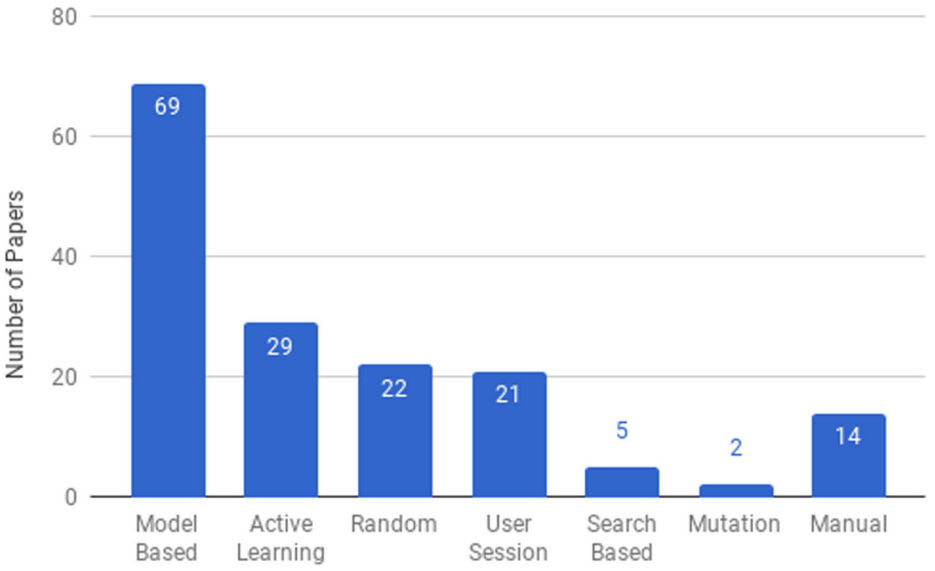

Fig. 4 Types of techniques for test case generation proposed in the selected papers

previous executions of the same application (in Liu et al. (2010b), Lin et al. (2014), Mendez-Porras et al. (2015a), Packevicius et al. (2015), and Tang et al. (2016)), whereas seven other articles propose the evaluation of invariants, such as race conditions (Hsiao et al. 2014; Maiya et al. 2014; Bielik et al. 2015) or other specific invariants (Hao et al. 2014; Shan et al. 2016; Li et al. 2017). In particular, in Zaeem et al. (2014), invariants retrieved from the analysis of common bugs of the applications have been considered as oracles.

Finally, in five papers, manually written assertions have been used to evaluate the proposed testing tool (Fazzini et al. 2017; Liu et al. 2014a; She et al. 2009; Jiang et al. 2016; Wu et al. 2016).

Figure 5 shows the distribution of papers with respect to the types of oracles.

4.6 RQ 2.4 What kinds of test artifacts are generated?

As regards the explicit outputs of the proposed testing techniques and tools, 75 papers propose techniques that generate executable test cases. In many cases, the tests can be executed only by means of the same tool able to generate them. In some cases, the proposed tools are able to export the generated test cases so that they can be executed outside the context of the test generation process (for example, in the form of JUnit test cases). For example, Android Ripper (Amalfitano et al. 2012b) is able to generate executable JUnit test cases, while RERAN (Gomez et al. 2013) is able to reproduce the same user sessions that have been captured, and Sapienz (Mao et al. 2016) is able to generate sequence of events by means of which it is possible to derive test cases.

In most of the approaches (112 out of 131 papers), a test execution report is provided as output, providing information about crashes, code coverage, or just an execution log. In addition, as shown by RQ 2.2, in 27 papers, models are automatically built during the testing activities and can be considered as an additional output that can be useful to comprehend

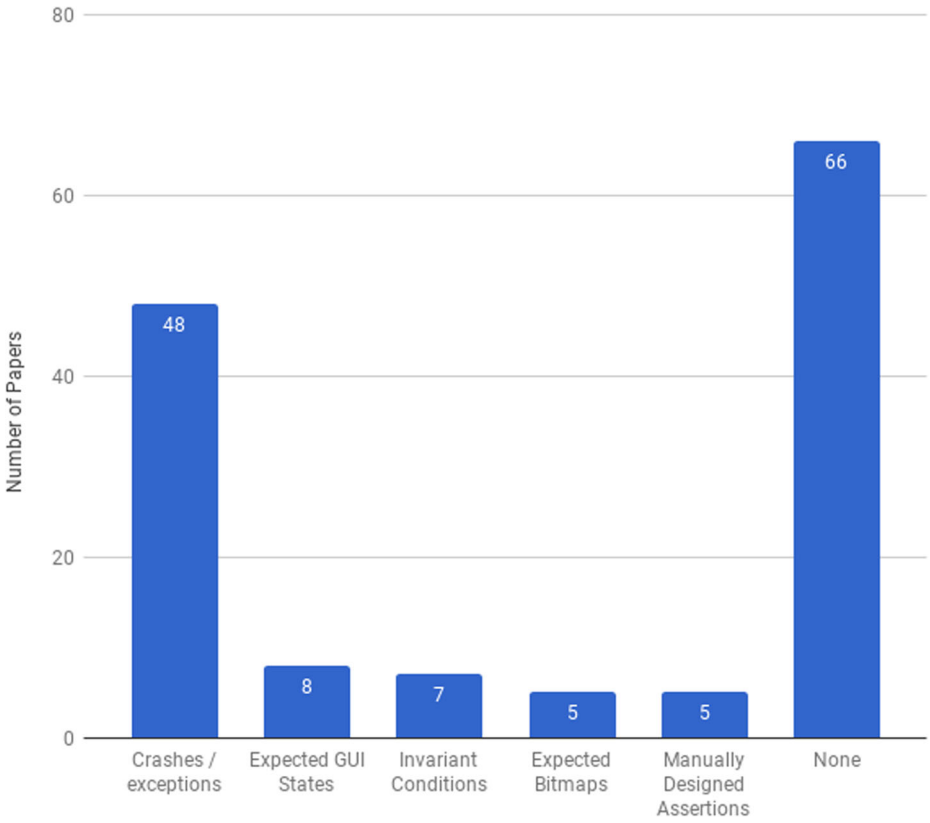

Fig. 5 Types of oracles considered in the selected papers

the behavior of the tested application. Finally, in 15 cases, the proposed techniques are able to produce input values that can be used to automatically generate test cases.

4.7 RQ 2.5 What are the characteristics of the proposed testing tools?

Most of the selected contributions are not strictly theoretical or methodological, but they include the presentation of tools implementing the proposed testing techniques. In fact, 107 different tools have been presented in the selected papers, but only some of these tools are freely available. The source code of 22 of these tools is available, usually in the context of github projects, whereas six of these tools are available only in executable form or in demo version (i.e., Agrippin Amalfitano et al. (2015a), the executable only versions of Android Ripper used in Amalfitano et al. 2012a, b, EventRacer (Bielik et al. 2015), TrimDroid (Mirzaei et al. 2016b), and BARISTA (Fazzini et al. 2017)). In addition, three tools are commercially available (i.e., Testdroid (Kaasila et al. 2012), Caiipa (Liang et al. 2014), and MZoltar (Machado et al. 2013) for which a limited demo version is also available): they have borne as academic prototypes and they have evolved in commercial tools.

It is interesting to see that all the open-source tools presented in the selected papers have been developed for the Android platform, whereas Caiipa is the unique example of

a commercial tool presented in the selected papers and developed for the Windows Phone framework.

The test generation techniques adopted by the 22 open-source tools have been classified by distinguishing between static analysis techniques (where tests are generated on the basis of information such as source code or high-level models of the application under test), dynamic analysis techniques (where tests are generated and executed on-the-fly by analyzing the application during its execution), and hybrid techniques (that combine static and dynamic analyses). We found six tools based on static analysis techniques: ICCMATT (Jha et al. 2015) and DroidFuzzer (Ye et al. 2013), that analyze the source code of the application under test including the Android manifest, BBoxTester (Zhauniarovich et al. 2015), and CRAXDroid (Yeh et al. 2014) that analyze the bytecode of the application, Magi[c] (Nguyen et al. 2012) that analyzes a statically designed FSM model of the application, and THOR (Adamsen et al. 2015), that mutates the source code of existing test cases.

Thirteen other tools are based on dynamic analysis techniques. Most of them exploit systematic model learning and/or random techniques for the automatic exploration of the GUI of the application under test (e.g., Android Ripper (Amalfitano et al. 2012b), A3E (Azim and Neamtiu 2013), SwiftHand (Choi et al. 2013), Dynodroid (Machiry et al. 2013), SlumDroid (Imparato 2015), DroidMate (Jamrozik and Zeller 2016), MCrawlT (Salva and Laurencot 2015), PUMA (Hao et al. 2014), SmartMonkey (Sun et al. 2016), DroidBot (Li et al. 2017), and DroidRacer (Maiya et al. 2014)). Finally, RERAN (Gomez et al. 2013) and VALERA (Hu et al. 2015) are capture and replay tools able to automatically generate and execute test cases corresponding to the observed executions of the application under test.

The remaining three tools combine static and dynamic analysis techniques. In fact, Sapienz (Mao et al. 2016) combines information obtained by static analysis and dynamic exploration techniques to implement a search-based exploration strategy. KREFinder and KREReproducer (Shan et al. 2016) respectively exploits a static analysis technique to find the resume and restart event handlers, and a dynamic exploration technique to test the behavior of the application under test with respect to the execution of these events. Finally, JPF-Android (van der Merwe et al. 2012) exploits static analysis to model an Android application in order to dynamically test it by adopting a Java PathFinder extension in the context of a Java Virtual Machine.

Not all these tools are currently maintained: 14 out of 22 have not been updated since 2016. The strict dependence between the tools and the rapidly evolving Android environment makes very hard and time consuming the maintenance of these projects. Choudhary et al. (2015) have performed a comparative experiment of the performance of several of these tools by executing them in the same execution environment. They have reported about their difficulties in adapting the tools to a common target execution environment. Most of the tools have been implemented partially or totally in Java in order to interact with the source code of the application under test and/or with the Android JUnit test environment. Some tools (in particular the ones interacting at low level with the Android framework) have been implemented by partially or totally using other languages such as Python (Ye et al. 2013; Zhauniarovich et al. 2015; Li et al. 2017; Maiya et al. 2014; Mao et al. 2016; Sun et al. 2016), Scala (Choi et al. 2013), Ruby (Azim and Neamtiu 2013), Kotlin (Jamrozik and Zeller 2016), C (Gomez et al. 2013), C++ (Hu et al. 2015), and Javascript (Adamsen et al. 2015).

Table 5 reports a summary of the characteristics of the 22 open-source tools, the type of the adopted test case generation technique, the URLs at which they are currently available, and the date of their last update.

Table 5 Characteristics of open-source testing tools

Tool name	Tool description
A3E (Azim and Neamtiu 2013)	It is a tool able to dynamically execute user events on the application under test by performing depth first or targeted exploration strategies on dynamically reconstructed activity transition graphs. It is able to measure the achieved coverage in terms of executed activities and methods. It is able to generate executable sequence of events.
Technique Type:	Dynamic analysis
URL:	https://github.com/tanzirul/a3e
Last updated:	September 15, 2016
Android Ripper (Amalfitano et al. 2012b)	It is an active learning tool able to generate test cases based on the exploration of a dynamically built GUI model of the application under test. It supports different exploration strategies including random and systematic strategies and is able to generate reusable test cases. It has been evaluated in terms of achieved code coverage and number of found crashes and has been used, too, to compare the performance of different testing strategies (Amalfitano et al. 2017).
Technique Type:	Dynamic analysis
URL:	https://github.com/reverse-unina/AndroidRipper
Last updated:	December 28, 2017
BBox Tester (Zhauniarovich et al. 2015)	It is a tool able to automatically generate test cases by statically analyzing the bytecode of the application under test. The effectiveness of the generated test cases can be measured in terms of code coverage by means of Emma.
Technique Type:	Static analysis
URL:	https://github.com/zyrikby/BBoxTester
Last updated:	June 11, 2016
CRAX Droid (Yeh et al. 2014)	It is a tool able to generate test cases on the basis of the symbolic execution of the bytecode of the Android applications under test. Its effectiveness has been evaluated in terms of crashes found.
Technique Type:	Static analysis
URL:	https://github.com/Lance0312/craxdroid
Last updated:	May 1, 2014
Droid Fuzzer (Ye et al. 2013)	It is a tool able to statically generate fuzz test cases on the Android application under test on the basis of information retrieved from the Android Manifest file. The effectiveness of the test cases is evaluated by detecting framework and application crashes.
Technique Type:	Static analysis
URL:	https://github.com/manfiS/droidfuzzer
Last updated:	February 13, 2016
Droid Mate (Jamrozik and Zeller 2016)	It is a tool able to dynamically generate and execute user events on the application under test on the basis of the set of executable events retrieved by the UIAutomator library.
Technique Type:	Dynamic analysis
URL:	https://github.com/konrad-jamrozik/droidmate
Last updated:	August 6, 2017

Table 5 (continued)

DroidBot (Li et al. 2017)	It is a tool able to automatically generate and execute test cases on the basis of a state-transition model generated on the fly and on a depth-first exploration strategy.
Technique Type:	Dynamic analysis
URL:	https://github.com/honeynet/droidbot
Last updated:	March 22, 2018
Droid Racer (Maiya et al. 2014)	It is a tool able to dynamically generate and execute test cases aimed at revealing failures due to data races in the application under test.
Technique Type:	Dynamic analysis
URL:	http://www.iisc-seal.net/droidracer
Last updated:	May 13, 2017
Dynodroid (Machiry et al. 2013)	It is a tool able to dynamically explore the application under test by means of uniform random, smart random, or active learning strategies. It is able to dynamically execute both user and system events. It has been evaluated in terms of achieved code coverage and crashes found.
Technique Type:	Dynamic analysis
URL:	https://bitbucket.org/pag-lab/dynodroid
Last updated:	November 14, 2017
ICCMATT (Jha et al. 2015)	It is a tool able to automatically generate test cases on the basis of Intercomponent Communication models built by statically analyzing the source code and the manifest of the Android application under test.
Technique Type:	Static analysis
URL:	https://github.com/HiFromAjay/ICCMATT
Last updated:	September 28th, 2016
JPF-Android (van der Merwe et al. 2012)	It is a tool able to dynamically test an Android application by executing it in the context of a Java Virtual Machine by exploiting the Java Path Finder tool.
Technique Type:	Static and dynamic Analyses
URL:	http://heila.bitbucket.org/jpf-android/
Last updated:	November 16, 2017
KREFinder (Shan et al. 2016)	It is a tool able to analyze the bytecode of the application under test in order to find the code of the resume and restart event handlers and the data involved in the execution of these methods. The tool is able to dynamically test an Android application against failures subsequent to sequences of these events.
Technique Type:	Static and dynamic Analyses
URL:	https://github.com/krefinder/krefinder-source-code
Last updated:	July 19, 2016
Magi[c] (Nguyen et al. 2012)	It is a tool able to generate test cases on the basis of a statically defined FSM model representing the behavior of the Android application under test, by combining model-based and combinatorial testing techniques. It is able to evaluate the achieved code coverage and the corresponding ability in finding model violations.
Technique Type:	Static analysis
URL:	http://selab.fbk.eu/magic/
Last updated:	February 16, 2012

Table 5 (continued)

MCrawlT (Salva and Laurencot 2015)	It is a tool able to perform different dynamic exploration strategies on the Android application under test by applying a technique based on the ant colony optimization algorithm. It is able to evaluate the achieved code coverage and the failure finding ability.
Technique Type:	Dynamic analysis
URL:	https://github.com/statops/MCrawlerT
Last updated:	November 27, 2014
PUMA (Hao et al. 2014)	It is a framework by means of which many different dynamic testing strategies expressed with the Pumascript language can be executed. In particular, it supports random testing, accessibility testing, security testing, and dynamic analysis of the Android application under test.
Technique Type:	Dynamic analysis
URL:	https://github.com/USC-NSL/PUMA
Last updated:	September 2, 2014
RERAN (Gomez et al. 2013)	It is a capture and replay tool able to automatically generate replayable sequence of events on the basis of the observation of the execution of the Android application under test on a real device.
Technique Type:	Dynamic analysis
URL:	https://github.com/lorenzogomez/RERAN
Last updated:	August 7, 2013
Sapienz (Mao et al. 2016)	It is a tool able to generate a set of executable test cases on an Android application under test, guided by a multi-objective search-based testing strategy aimed at optimizing code coverage, failures finding capability and test sequences length. It combines random and systematic strategies for test case generation.
Technique Type:	Static and dynamic analyses
URL:	http://github.com/Rhapsod/sapienz
Last updated:	August 27, 2017
Slum Droid (Imparato 2015)	It is a tool able to extend the Android Ripper tool by considering different strategies of setting of the input field values during the exploration of the application under test.
Technique Type:	Dynamic analyses[-.5pt]
URL:	https://github.com/slumdroid
Last updated:	October 10, 2016
Smart Monkey (Sun et al. 2016)	It is a tool able to extend and improve Monkey by applying techniques of visual saliency detection to the screenshots of the Android application under test, in order to identify operable parts of the GUI and to execute randomly generated events on them.
Technique Type:	Dynamic analysis
URL:	https://github.com/sunchenglong/smartmonkey
Last updated:	March 17, 2016
Swift Hand (Choi et al. 2013)	It is a tool able to efficiently explore the GUI of the Android application under test by means of an optimized active learning L* strategy aiming at the minimization of application restarts. The effectiveness of the executed exploration has been evaluated in terms of the achieved branch coverage.
Technique Type:	Dynamic analysis
URL:	https://github.com/wtchoi/SwiftHand
Last updated:	January 13, 2015

Table 5 (continued)

THOR (Adamsen et al. 2015)	It is a tool able to inject in existing JUnit test cases sequences of neutral events (i.e., events that should not cause variations in the applications' behavior). By executing the generated test cases, it is possible to evaluate the robustness of the Android application under test by observing if the GUIs remain unchanged after the execution of each sequence of neutral events.
Technique Type:	Static analysis
URL:	https://github.com/cs-au-dk/thor
Last updated:	May 17, 2017
VALERA (Hu et al. 2015)	It is a tool supporting capture and replay of user and system events that improves the RERAN tool in terms of fidelity of representation and replication of complex events.
Technique Type:	Dynamic analysis
URL:	http://spruce.cs.ucr.edu/valera/tutorial.html
Last updated:	September 29, 2016

These tools are usually based on existing libraries and other tools. The most commonly used resource is the Robotium[11] library supporting the writing of JUnit test cases, that is used by 21 tools. Other similar libraries are UIAutomator[12] (used in 11 contributions), HierarchyViewer[13] (used in two contributions), and Espresso[14] (used just in Tang et al. 2016). Espresso is the library recommended by Google for the development of test cases but it is not yet considered by other academic studies, probably for its recent diffusion (it is available and integrated with Android Studio just since 2014). The Emma library, that is available in all the Android framework versions, has been used in 14 different contributions to measure code coverage. Other supporting tools often used and that are provided by the Android framework are Monkey[15] (used in nine contributions) and chimpchat, that is included in MonkeyRunner[16] (used in five contributions) that are both able to generate and send low-level random events to an Android application. Java Path Finder[17], that is a framework designed to analyze Java applications, has been used in five contributions to perform symbolic analysis of the Java source code of Android applications.

4.8 RQ 2.6 Which mobile frameworks are the targets of the proposed techniques and tools?

On the basis of the collected data, the mobile framework object of most of the existing studies in literature is the Android framework. In fact, 121 papers out of 131 propose techniques suitable for some Android framework versions.

[11] https://github.com/RobotiumTech/robotium
[12] https://google.github.io/android-testing-support-library/docs/uiautomator/
[13] https://developer.android.com/studio/profile/hierarchy-viewer.html
[14] https://google.github.io/android-testing-support-library/docs/espresso/
[15] https://developer.android.com/studio/test/monkey.html
[16] https://developer.android.com/studio/test/monkeyrunner/index.html
[17] http://javapathfinder.sourceforge.net/

The iOS-based systems are rarely objects of specific studies in literature, probably due to their proprietary nature, that makes it difficult for their diffusion in the academic community. In fact, we found in this study only a single paper proposing a testing technique applied to iOS applications (the recent one of Liu et al. 2017) and other four papers that propose techniques and tools that are both applicable to iOS and Android applications (Li et al. 2014a; Joorabchi et al. 2016; Gudmundsson et al. 2016; Zun et al. 2016). In addition, just three papers are focused on the less diffused Windows Phone framework (Liang et al. 2014; Ravindranath et al. 2014; Mayan et al. 2015). Six other papers are based on Symbian or J2ME (Delamaro et al. 2006; Jiang et al. 2007; She et al. 2009; Liu et al. 2010b; Nagowah and Sowamber 2012; Dev et al. 2012), but they have all been published before 2013 since this framework is clearly in a declining phase. Figure 6 shows the prevalence of Android-based contributions with respect to the ones based on the other platforms.

4.9 RQ 2.7 Are the proposed techniques and tools usable on emulators or real devices?

Unfortunately, the details given in the considered papers are not always sufficient to provide an answer to this question. In some cases, in fact, no information at all has been provided, while in some other cases, this information can only be deduced by observing the description of the experiments reported in the evaluation section. On the basis of the available information, it appears that in 65 contributions the testing approach can be executed on emulators, while in 40 cases it can be executed on real devices and in the remaining 26 cases it can be executed on both real devices and emulators. In particular, the use of real devices enables some specific analysis involving different devices (e.g., Android TVs in Jiang et al. 2016), the verification of timing problems in capture and replay approaches (Gomez et al. 2013; Gomez et al. 2016; Ravindranath et al. 2014), and the detection of critical races (Hsiao et al. 2014; Maiya et al. 2014).

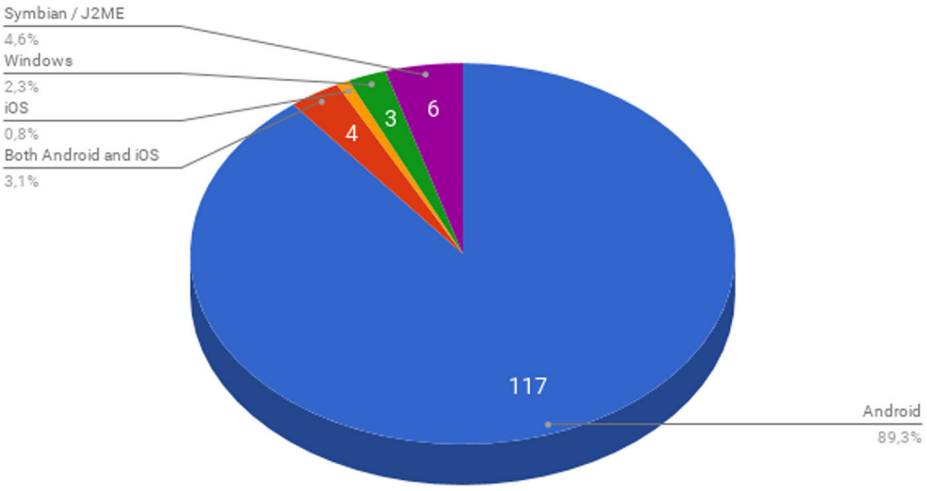

Fig. 6 Distribution of papers according to the targeted mobile frameworks

4.10 RQ 3.1 What are the characteristics of the performed evaluation studies?

In 45 papers out of 131, the validity of the proposed techniques and tools have been shown only on the basis of a demonstration of their feasibility obtained by showing examples of their use.

In the remaining 86 papers, experimental evaluations of the proposed techniques and tools have been carried out, and specific effectiveness metrics have been measured. The more frequently considered effectiveness metric is the number of failures that have been found (in particular crashes, exceptions or any other failure revealed by the considered oracles): these metrics have been evaluated in 54 different papers. Code coverage metrics (such as LOC coverage, method coverage, branch coverage, activity coverage) have been measured, instead, in 44 different papers. In 18 of these papers, both these evaluations have been carried out. More rarely, techniques and tools have been evaluated in terms of their ability in finding injected faults (in six papers).

Figure 7 shows the distribution of papers according to the different considered evaluation methods.

4.11 RQ 3.2 What are the characteristics of the sets of applications objects of the evaluation experiments?

Most of the evaluation studies reported in the selected papers involve toy examples or real mobile applications. In only nine cases, the papers provide only a qualitative evaluation of the proposed techniques in terms of a description of the offered features, without any example or case study (Hesenius et al. 2014; Kaasila et al. 2012; Mendez-Porras et al. 2015a; Prathibhan et al. 2014; Reddy et al. 2016; Mirzaei et al. 2012; van der Merwe et al. 2012; Akanksha Ashok Magare 2016; Dutia et al. 2015).

In 30 of the remaining papers, the evaluation of the proposed techniques or tools is based only on toy examples, i.e., applications realized and proposed by the authors of the paper, while in all the other articles, case studies involving real mobile applications have been presented. In 13 papers, just a case study involving a simple application is presented, while in

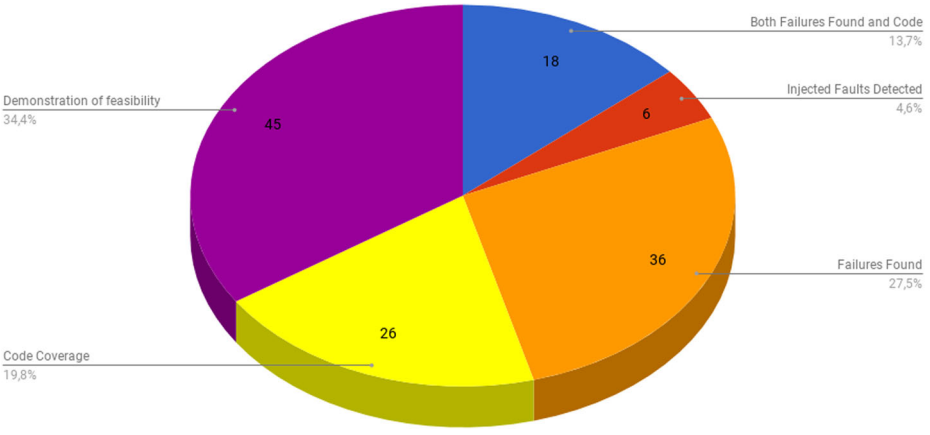

Fig. 7 Distribution of papers according to the considered evaluation methods

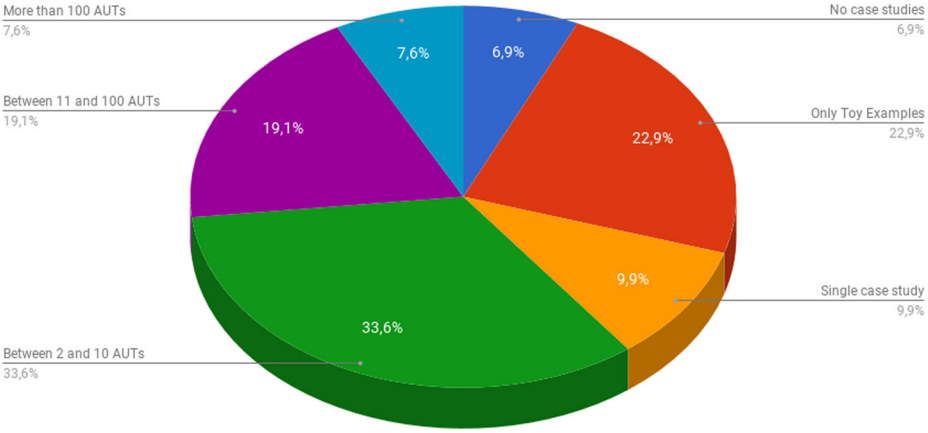

Fig. 8 Distribution of papers according to the number of object applications

other 44 papers, several applications (from 2 to 10) have been used to assess the effectiveness of the proposed contribution. In 25 of the remaining cases, a more significant evaluation is presented, based on a number of applications between 11 and 100. Finally, in 10 cases, massive experiments involving more than 100 applications have been presented.

Figure 8 shows the distribution of the papers with respect to the number of application object of the evaluation of the proposed techniques and tools.

The tools experimented on more than 100 applications are BBoxTester (Zhauniarovich et al. 2015), EventRacer (Bielik et al. 2015), PUMA (Hao et al. 2014), DroidMate (Jamrozik and Zeller 2016), Caiipa (Liang et al. 2014), JarJarBinks (Maji et al. 2012), Sapienz (Mao et al. 2016), SIG-Droid (Mirzaei and Heydarnoori 2015), Vanarsena (Ravindranath et al. 2014), and KREfinder (Shan et al. 2016). The experiments involving the largest quantity of applications are all related to black box testing techniques searching for crashes or other invariant oracles, since they can be carried out in a fully automated testing process.

Many of the experiments involving a number of applications between 10 and 100 are aimed at the evaluation of white box testing techniques. For example, the model learning techniques provided by Dynodroid (Machiry et al. 2013), MCrawlT (Salva and Laurencot 2015), and Crashscope (Moran K et al. 2016) have been tested, respectively, on 50, 32, and 20 applications.

The applications under test selected for the experiments are usually open-source real applications (in 49 cases), often downloaded from F-Droid[18]. In 32 cases, real applications published on an official market (usually Google Play[19] market for Android apps) have been considered. Open-source applications from F-Droid have generally been used for experiments involving source code coverage measures, while applications downloaded from Google Play have generally been used for black box testing approaches.

In some cases (in particular, in the experiments involving open-source applications), it is possible to estimate the complexity of the tested applications since the authors have reported some size metrics (usually the number of LOCs of the applications under test). We have found this information on 28 papers and we can observe that in only two papers very small

[18] https://f-droid.org/
[19] https://play.google.com/store

applications (less than 1 kLOC in average) have been considered, while in 17 cases the experiments involved relatively small applications (between 1 and 10 kLOC in average). Only eight experiments involved medium-sized applications having more than 10 kLOC in average.

4.12 RQ 3.3 What are the characteristics of the performed comparative studies?

Experiments involving the comparison between the effectiveness of the proposed approaches with the one of other existing tools represent a convincing way to evaluate the improvements of the considered approach with respect to the state of the art. Our study shows that they are not very common in this field.

In fact, only in 39 papers out of 131 is there at least a comparative study of the performance of the proposed testing approach.

The tool that is mostly used as baseline for comparisons is the Monkey tool (available in the Android Framework) that can be executed in a completely automatic manner, almost without any configuration effort. It has been considered as a term of comparison in 26 different articles. Other tools considered for comparative evaluations are Dynodroid (used as term of comparison in 10 papers), Android Ripper (in seven papers), and A3E and Swift-hand (in three papers). In only four papers, comparisons between the effectiveness of the proposed testing tool and the one of test cases designed by human testers (students or the authors of the paper) have been presented.

Different attributes have been considered to compare the performance of the proposed approach with other existing ones: in particular, failure detection capability has been considered in 15 papers while code coverage capability has been used in 23 other papers. Simple comparisons in terms of offered features have been performed in 14 papers. In particular, in eight papers (Zhauniarovich et al. 2015; Zhu et al. 2015; Machiry et al. 2013; Mao et al. 2016; Moran K et al. 2016; Nguyen et al. 2012; Qin et al. 2016a; Wu et al. 2016), comparative experiments have been carried out, where failure detection capability and achieved code coverage have been both considered.

The papers presenting the comparative experiments involving the largest number of different tools are the ones of Salva and Laurencot (2015) and of Moran K et al. (2016). In the paper of Salva and Laurencot, the performance of the proposed MCrawlT tool is compared to the ones of Monkey, Orbit, Guitar, AndroidRipper, SwiftHand, and Dynodroid both in terms of offered features, achieved code coverage, and failure detection capability. In the paper of Moran et al., instead, there is a comparative study of the features offered by 20 different tools and an experiment for the comparison of the failure detection capability of the proposed CrashScope tool with the ones of five other different tools (A3E, Android Ripper, Dynodroid, PUMA, Monkey). However, other interesting comparative studies can be found in secondary works, such as in the recent works of Choudhary et al. (2015) and Amalfitano et al. (2015b, 2017).

4.13 RQ 4.1 What is the number of published articles per year?

The evaluation of the article count per year has confirmed the relative youth of the mobile testing automation field and a growing interest in its findings. In fact, as shown in Fig. 9, there are very few papers before 2011. These papers (Delamaro et al. 2006; Jiang et al. 2007; She et al. 2009; Liu et al. 2010b) are all about J2ME or Symbian, while the first paper based on Android applications is from the same authors of one of the papers on Symbian (Liu et al. 2010a). Since 2011, probably due to the great success and diffusion of the Android devices,

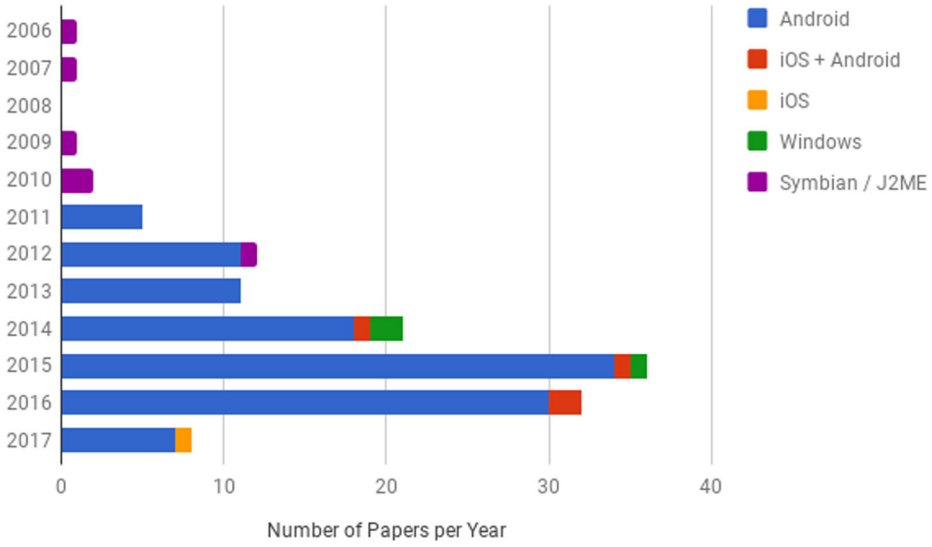

Fig. 9 Distribution of the selected papers per year and per used frameworks

there has been a rapid growth of the number of articles based on Android application testing (and in few cases on testing of Windows mobile phone applications or iOS applications), that has reached a peak in 2015 with 36 different papers. In 2017, eight papers have already been published at the time of this study.

4.14 RQ 4.2 Which are the venues having the higher article counts?

The growing interest of the scientific community for this field is also witnessed by the number of papers published and presented on general purpose, eminent conferences, such as the International Conference on Software Engineering (ICSE), hosting nine mobile testing automation papers in the last years six and the ACM Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) that has hosted five articles. Several other articles have been presented in other general software engineering conferences: four at the Symposium on the Foundations of Software Engineering (FSE), four at the Asia-Pacific Software Engineering Conference (APSEC), three at the Automated Software Engineering Conference (ASE), three at the International Computer Software and Applications Conference (COMPSAC), three at the ACM Symposium on Applied Computing (SAC), two at the Conference on Programming Language Design and Implementation (PLDI), and two at the Conference on Software Engineering and Knowledge Engineering (SEKE).

The two specific communities hosting mobile testing automation papers are the testing community and the mobile computing community. In the testing community, we have found six papers presented at the Workshop on Automation of Software Test (AST), five papers at the International Symposium on Software Testing and Analysis (ISSTA), five at the International Conference on Software Testing, Verification and Validation (ICST), three at the International Workshop on TESTing Techniques and Experimentation Benchmarks for Event-Driven Software (TESTBEDS), and two at the International Symposium on Software Reliability Engineering (ISSRE). In venues related to the mobile computing community,

instead, there are, among the others, six papers presented at the ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), two papers at the International Conference on Mobile Systems, Applications, and Services (Mobisys), two papers at the International Workshop on Mobile Development (Mobile!), and two papers at the International Workshop on Software Development Lifecycle for Mobile (DeMobile). Figure 10 shows in the form of histogram the venues hosting at least two of the papers included in this study.

Finally, in the last years, some of the more influential journals have published researches on mobile testing automation (such as two papers in ACM Software Engineering Notes (van der Merwe et al. 2012; Mirzaei et al. 2012): one paper in the Journal of Systems and Software (Qin et al. 2016a), in IEEE Software (Amalfitano et al. 2015d), in the IEEE Transactions on Software Engineering (Lin et al. 2014), and in the IEEE Transactions on Reliability (Jiang et al. 2016). In addition, two secondary papers related to mobile testing automation have been published in the Journal of Systems and Software (Zein et al. 2016; Amalfitano et al. 2017).

4.15 RQ 4.3 Which are the more influential articles in terms of citation counts?

An approximate analysis of the influence of the publications can be carried out by counting the overall number of articles citing the ones considered in this systematic mapping study. With reference to the count values provided by the Scopus search engine (that appears to be the more exhaustive search engine considered in this study), it is possible to observe that the most cited work is the one of Amalfitano et al. (2012b) presented at the IEEE/ACM International Conference on Automated Software Engineering (ASE) in 2012, having 156 citations up to September 2017. Other contributions of some of these authors are in the set of the most cited papers of Amalfitano et al. (2011, 2015d). Other works with more than 100 citations are the ones of Machiry et al. (141 citations), Anand et al. (2012) (112), Hu and

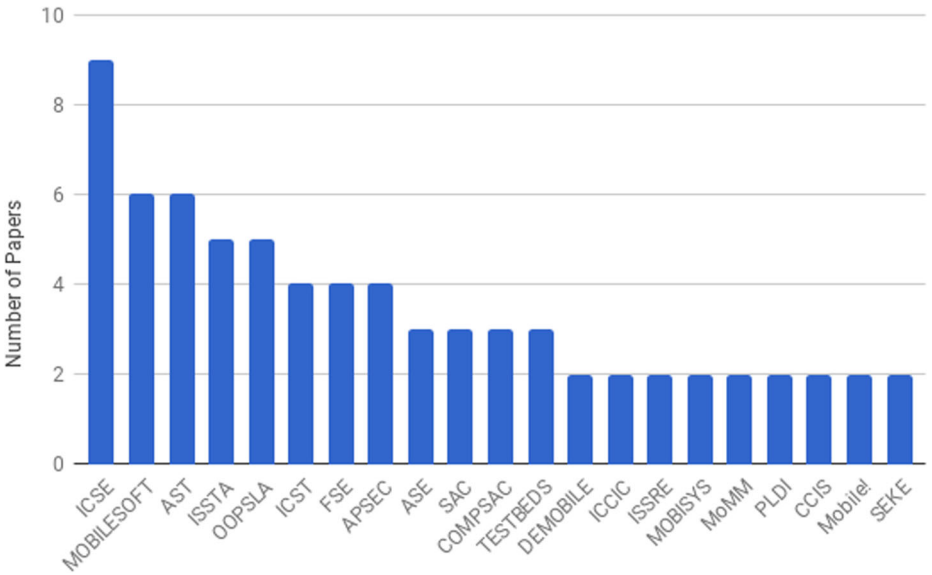

Fig. 10 Venues hosting two or more of the selected papers

Neamtiu (104), and Gomez et al. (101). Figure 11 shows the most cited papers considered in this study.

It is interesting to note that almost all the most cited contributions have a practical relevance, since they present open-source testing tools, that sometimes have been used as benchmarks in citing papers.

4.16 RQ 4.4 Who are the authors with the higher number of articles?

The authors having the highest number of papers are currently Amalfitano, Fasolino, and Tramontana, who work in the same group at the University of Naples and are authors of eight different papers; Neamtiu (seven papers); and Linares-Vazquez (5 papers). It is interesting to note that only a very limited number of interchanges of authors between different groups have been observed.

4.17 RQ 4.5 Which countries have produced more articles?

The analysis of the countries of affiliation of the authors of the selected papers shows how this field of study is diffused worldwide, with contributions from all the continents. The countries with the higher number of contributions are the USA (38), China (26), Italy (10), and India (9). Figure 12 shows a world map where countries having more publications are filled in with darker colors. It is possible to observe, too, that only 17 works have been written by authors of at least two different countries of affiliation.

4.18 RQ 4.6 Which are the author affiliations?

A detailed analysis of the affiliations of the authors of the selected papers has revealed that only few articles result from collaborations between authors from industry and authors from academia (17 papers), whereas the most part of the articles (115 papers) have been written

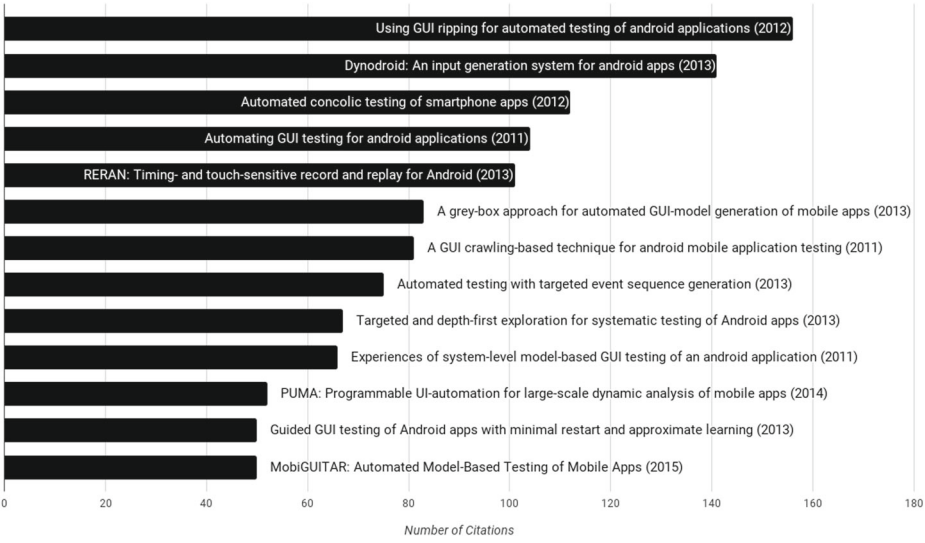

Fig. 11 The most cited papers (up to September 2017)

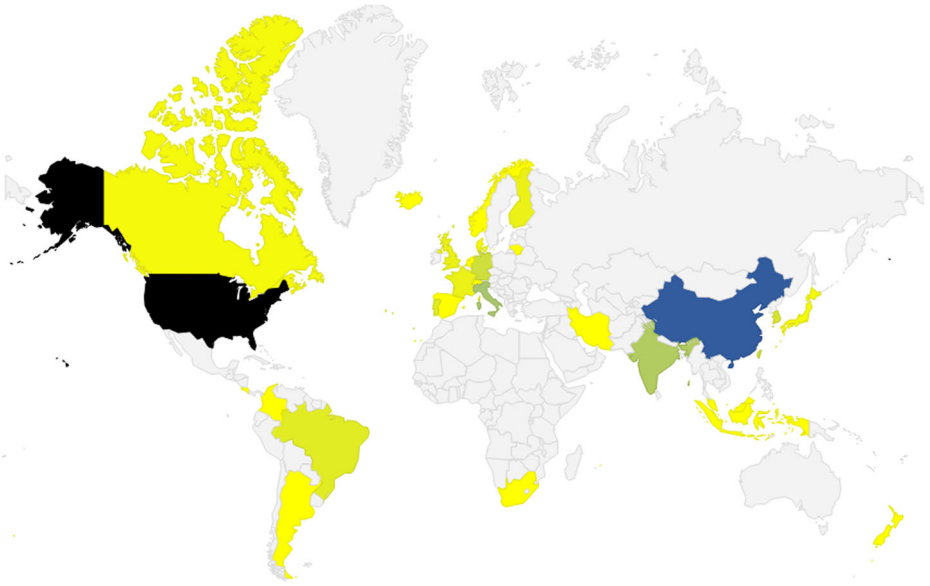

Fig. 12 Map of the country of affiliation of the authors of the selected papers

only by authors from academia. No papers have been written only by authors from industry. The most active groups in this fields are the ones of the University of Naples Federico II (8 papers), of the College of William and Mary in the USA (6 papers), of the Nanjing University in China (6 papers), of the University of California (5 papers), and of the George Mason University (4 papers). As regards the industry, 3 papers are in collaboration with Microsoft Research and other 3 in collaboration with Fujitsu laboratories. Figure 13 provides a geographical view of the distribution of the affiliation of the authors of the selected papers where larger circles indicate the institutions from which larger numbers of papers come from.

5 Threats to validity

Threats to the validity of a systematic mapping study are due to several possible aspects such as the suitability of the categorization scheme, the recall and precision of study selection, the accuracy of data extraction, and the correctness of the conclusions. In this section, we will discuss the main threats to the validity of this study and the actions that have been taken to mitigate them in coherence with the classification of threats adopted by Vahid Garousi in some of its systematic mapping studies (e.g., in Garousi et al. 2013; Garousi and Mantyla 2016).

5.1 Threats to Internal validity

Threats to internal validity can be caused by the process adopted to select the articles considered in this systematic mapping.

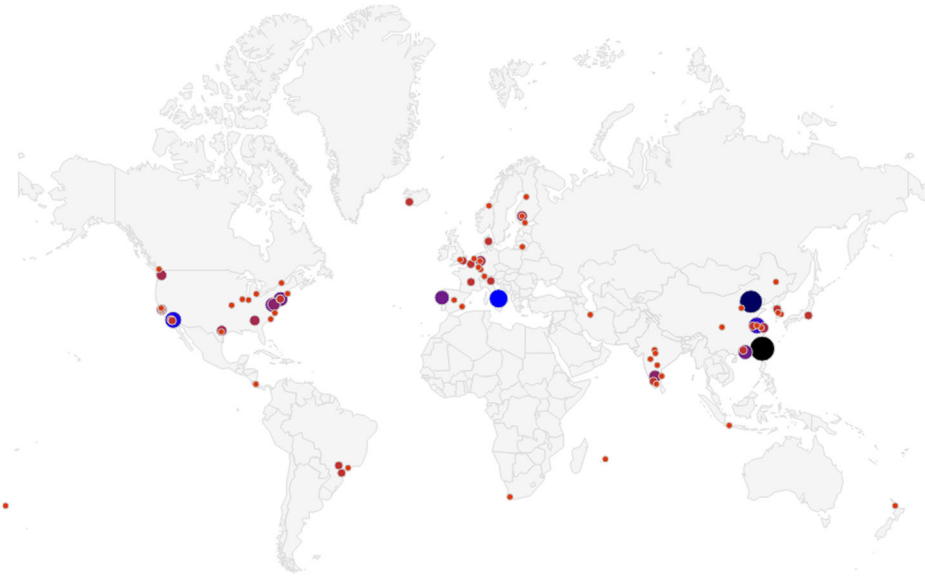

Fig. 13 Geographical map of the affiliations of the author of the selected papers

A first threat is related to the capability of the designed queries in finding all the works in literature describing techniques and tools supporting automation of functional testing of mobile applications. To this purpose, the set of keywords considered in the queries has been initially designed on the basis of the domain knowledge of the authors and of the similar keywords used in the most similar studies in literature (in particular, the ones of Holl Holl and Elberzhager (2016), Zein et al. (2016), Mendez-Porras et al. (2015b), and Sahinoglu et al. (2015)). Queries resulting from several combinations of keywords have been formulated taking into account the characteristics and the limitations imposed by the search engines, too. In order to validate the proposed queries, they have been preliminary verified with respect to a set of 55 publications that should surely be included in the systematic mapping for their coherence with the topic of the study, as described in Section 3.2.

A second threat is related to the evaluation of the inclusion and exclusion criteria. To this aim, inclusion and exclusion criteria have been expressed in order to be objectively evaluated and to mitigate the risk of arbitrary judgements of the authors of the study. In addition, each author labeled as "borderline" each paper for which he has some doubts about its inclusion. The inclusion and the exclusion of these papers have been jointly discussed by all the authors in order to have a shared decision.

5.2 Threats to construct validity

Threats to construct validity are related to the suitability of the proposed RQs and of the attributes characterizing the categorization scheme. In order to limit this threat, the GQM approach has been used to preserve the traceability between research goals, questions, and metrics. The categorization scheme has been obtained in several steps. In the first step, a set of attributes and possible values has been jointly designed by all the authors. During the data extraction step, the authors have classified the papers with respect to the proposed categorization scheme but feeling free to add other possible attributes and values that in

their opinion better fit the characteristics of the analyzed papers with respect to the proposed research questions. Finally, the authors have restructured the categorization scheme taking into account the added attributes and values. With this process, we think that the risk that relevant details of the analyzed papers have been neglected has been mitigated.

Another threat is related to possible inaccuracies of the extracted data. To mitigate this threat, for each considered paper the authors have labeled as borderline all the attributes for which they have some doubts about their values. The values assigned to these attributes have been discussed and fixed after a final joint review involving all the authors.

In more detail, we have observed that some questions are more prone to be answered inaccurately. As regards RQ 2.5, in order to know if each proposed tool is currently available on the web, we have considered the URLs declared in the papers and we have verified if the tool is actually available at that address. In addition, for the tools for which no URLs have been reported in the paper, we have searched online, on the basis of the name of the tool, if it is currently available. In this way, we have found that some tools that were not available at the time of the publication of the paper are now online, whereas some other tools are no more available at the indicated URLs. It is possible that some tools have changed their names or have been merged in other tools or frameworks, so we have not been able to find them. Of course, the list of available tools should be periodically updated in the online map. As regards RQ 2.7, many papers do not explicitly indicate whether the proposed techniques and tools are executable on real devices and/or emulators. In these cases, it was assumed that they can be executed on both. In some other papers, the answer to this question has been based on the experimental configuration declared in the evaluation section of the article. The number of citations needed to answer RQ 4.3 has been measured considering only Scopus, since it is the search engine providing the most complete view of the academic literature, as confirmed by the preliminary validation of the search queries. To this aim, we have discarded the measures provided by less inclusive search engines such as IEEExplore and ACM and the ones provided by Google Scholar that take into account many sources such as web sites, personal blogs, and other sources that are outside the scope of this study.

5.3 Threats to conclusion validity

In order to mitigate the possibility to give incorrect answers to the proposed research questions, we have formulated our conclusions only on the basis of the analysis of the extracted data. The online availability of the extracted data makes it possible for other researchers to independently validate the correctness of the conclusions. The spreadsheet with all the extracted data can be freely downloaded and commented on by readers and the authors will correct and update it periodically.

5.4 Threats to external validity

A first threat to external validity is related to its replicability. To avoid this threat, in this paper all the details needed to make possible an independent replication of the study have been reported.

Another threat is related to the generalization of the results of the study. The scope of this study is limited to the academic community, whereas its validity in the different contexts of software industry has not been evaluated. As regards the academic community, the completeness of the study is guaranteed by the set of considered search engines, which includes all those commonly used by software engineering researchers and by similar systematic mapping studies. On the other hand, the proposed strategy for study selection cannot be

extended to the industrial context, where contributions may be found as blog posts, pages on commercial web sites, presentations, videos, and other forms. As recently shown by Garousi and Mantyla (2016), a multivocal literature review (MLR) could be the way to extend the scope of an academic systematic mapping study to the industrial world.

6 Discussion

This section presents a possible approach for selecting papers from the systematic mapping using a ranking metric and reports a discussion about emerging trends and current research gaps in the field of automated functional testing of mobile apps.

6.1 Extracting relevant papers from the systematic mapping

As we already have reported in Section 1, Petersen et al. (2008, 2015) specify that a systematic mapping provides a classification scheme and structures a software engineering field of interest. In order to show how different facets of this scheme can be combined to answer more specific research questions, we now present a possible approach for article selection that expresses the readers' specific research interests.

To reach his specific objective, a reader will have to select a number of indicators from the ones measured in the systematic mapping study that he considers as relevant for his specific research interests. Therefore, the reader will have to design a scoring function to assign each indicator a score. Each paper will be ranked by means of a ranking metric that aggregates the single indicator scores. Finally, the reader will be able to select the papers that reach specific score values.

For example, a reader may be interested in selecting only articles that are characterized by a strong academic relevance in the literature and presenting techniques/tools having a strong practical relevance. To this aim, he may select the six indicators reported below and design the corresponding scoring functions for each of them.

A simple scoring function has been considered, that assigns each paper with a score of 0 or 1 for each indicator. An exception is represented by the indicator C4 for which the scoring function assigns one of the three possible scores of 0, 0.5, and 1.

The six considered indicators and the corresponding scoring functions are the following:

C1 The editorial relevance of the journal in which the paper has been included or of the conference at which the paper has been presented. Well-known ranking systems have been considered, i.e., Scimago[20] for journals and the Core Rankings Portal[21] for conferences. One point has been assigned to papers published on journals belonging to the quartiles Q1 and Q2 according to Scimago and to papers presented at rank A or A* conferences according to Core Ranking, zero points otherwise.

C2 The number of citations to the paper, already presented discussing RQ 4.3. One point has been assigned to papers reaching at least the threshold of 50 citations, zero points otherwise.

C3 The availability of the tool. One point has been assigned to papers describing publicly available testing tools, zero points otherwise.

[20] http://www.scimagojr.com/index.php

[21] http://portal.core.edu.au/conf-ranks/

C4 The approach used to evaluate the effectiveness of the proposed testing techniques/tools. It has been checked if in the selected papers the effectiveness has been evaluated by means of coverage metrics (e.g., code coverage or model coverage) or by counting the number of failures/faults detected. 0.5 points have been assigned to papers in which the effectiveness has been assessed only by means of coverage measures, 0.5 points for papers reporting only the number of detected failures or faults. One point has been assigned to papers presenting both these approaches, and zero points to papers that do not provide any empirical validation of the proposed technique/tool.

C5 The size of the application sample involved in the possible empirical study reported in the paper. One point has been assigned to papers reporting the results of studies involving at least 10 applications, zero points otherwise.

C6 The existence of empirical comparisons between the techniques/tools proposed in the paper and the state of the art. One point has been assigned to papers reporting techniques that have been empirically compared against the state of art (in the same paper or in other selected papers), zero points otherwise.

The total score of each paper can range between 0 and 6: the greater the value, greater the relevance of the paper.

Of course, we are aware that the selection of the indicators and the corresponding scoring functions can influence the results of the evaluation and that the corresponding scoring functions could produce different rankings of the analyzed papers. In particular, with respect to the proposed ranking metric, recent papers are hindered by the number of citations while older tools could be not involved in comparative studies due to the evolution of the mobile execution environments. The scoring functions and the ranking metric have been directly evaluated on the latter columns of the spreadsheet available at https://goo.gl/678T5P and can be easily modified by the readers by modifying the formulas reported in that spreadsheet.

The 12 papers that have reached a score of at least 4 points are reported in Table 6, while the complete ranking can be seen in the last column of the online spreadsheet.

According to the proposed ranking, the more relevant paper is the one of Machiry et al. (2013) presenting the publicly available tool Dynodroid that have demonstrated its ability in automatically testing Android applications with both model learning and random-based techniques, reaching a good code coverage level and founding real failures. Dynodroid has demonstrated its testing adequacy in an empirical study involving 50 applications, comparing it with the one of Monkey. It has been often considered as a benchmark for the other tools developed since 2013.

The second most relevant paper is the one of Azim and Neamtiu (2013) that presents A3E, another tool able to automatically explore and test Android applications. A3E has demonstrated its testing adequacy in terms of code coverage in an experiment involving 25 applications, and has been used as a benchmark by some other studies, that have often obtained better performance. A3E has been publicly available since 2013.

Another very relevant paper is the one of Mao et al. (2016) that presents the Sapienz tool, that is able to automatically explore and test Android applications outperforming Dynodroid and some other tools available in 2016, both in terms of code coverage and capability of finding real software failures. The improvements introduced by Sapienz are essentially related to its effective combination of random, systematic, and search-based exploration techniques. Sapienz is publicly available but it is no longer maintained (as stated on the tool web site).

Table 6 Most relevant papers according to the proposed ranking metric

Paper title	Total score
Dynodroid: An input generation system for android apps (Machiry et al. 2013)	6
Targeted and depth-first exploration for systematic testing of Android apps (Azim and Neamtiu 2013)	5,5
Sapienz: Multi-objective automated testing for android applications (Ma et al. 2016)	5
Using GUI ripping for automated testing of android applications (Amalfitano et al. 2012b)	5
RERAN: Timing- and touch-sensitive record and replay for Android (Gomez et al. 2013)	4,5
Towards black box testing of android apps (Zhauniarovich et al. 2015)	4
PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps (Hao et al. 2014)	4
Combining Model-Based and Combinatorial Testing for Effective Test Case Generation (Nguyen et al. 2012)	4
Automatic Text Input Generation for Mobile Testing (Zheng et al. 2017)	4
Automatically Discovering, Reporting and Reproducing Android Application Crashes (Moran K et al. 2016)	4
Testing android apps via guided gesture event generation (Wu et al. 2016)	4
A Context-Aware Approach for Dynamic GUI Testing of Android Applications (Zhu et al. 2015)	4

Another tool able to explore Android applications in a completely automatic way, with different possible random and systematic techniques, is Android Ripper, presented by Amalfitano et al. in a series of papers since 2011 (Amalfitano et al. 2011, 2012a, b, 2015d). The effectiveness of the tool in terms of achieved code coverage has been assessed by different experiments involving Android applications, and this tool has been used as a benchmark in many other different studies that have sometimes overtaken its performance. The tool is publicly available and maintained at the time of this research.

The paper of Choi et al. (2013) presents Swifthand, another tool for the automatic exploration and testing of Android applications that is publicly available and maintained up to 2015. It has been used, too, as benchmark by other studies (Mao et al. 2016).

Other very relevant papers present contributions related to the automation of more specific testing activities. The papers of Azim et al. have presented the tools RERAN and VALERA (Gomez et al. 2013; Hu et al. 2015) that are record and replay tools able to extract and analyze the sequences of events corresponding to user interactions with an Android application and to generate executable test cases able to reproduce these interactions with a high fidelity. The RERAN tool is currently publicly available and represents a very useful tool to automatically generate test cases by user sessions.

The paper of Maiya et al. (2014), instead, presents DroidRacer, that is devoted to the automatic research of concurrency bugs on Android applications. The DroidRacer tool has been capable to find many real bugs and is currently available and maintained. The recent paper of Shan et al. (2016) presents KREFinder, a tool able to find bugs due to incorrect management of the resume and restart of Android applications on the basis of information extracted via static analysis. KREFinder is publicly available and has been maintained up to 2016.

Finally, the paper of Hao et al. (2014) has presented a tool called PUMA and a language to configure it (PUMAScript) allowing the implementation of many different testing and quality assessment tasks on Android applications. The tool is publicly available and currently maintained. It represents the most flexible presented tool since it has been applied to several different testing activities.

In our knowledge, we can confirm the actual relevance of all the selected papers, so we are confident about the usefulness of the proposed metric.

6.2 Focus on GUI-based testing approaches for Android applications

Almost all the works found by this study concern techniques and tools applicable to the Android framework (about 92% of the total, as shown by the answer to RQ 2.6), with very few works that focus on other popular frameworks like iOS. We think that the reason of this polarization is related to the open-source nature of most of the Android tools, that makes it possible for researchers the realization of free testing tools and their free sharing for academic purposes.

In addition, most of the proposed approaches tackle the problem of testing by executing events on the GUI of the application under test. The reason for which GUI-based testing techniques are so popular may be due to the availability of libraries supporting GUI testing of Android applications via JUnit test cases since its earlier versions (the fundamental InstrumentationTestCase library has been released with the first version of Android in 2008).

The Android framework also provides tools and libraries allowing low-level interactions with the applications under test, such as MonkeyRunner or other basic system tools such as *sendEvent* and *getEvent* through which you can have access to the event stream of the application under test. They have been rarely exploited by the tools found in literature: a unique contribution in the literature is based on MonkeyRunner (Dutia et al. 2015), while *sendEvent* and *getEvent* are the basis for the capture and replay tools RERAN (Gomez et al. 2013) and Valera (Hu et al. 2015).

The low-level testing tool that is more often used by the tools retrieved in this study is Monkey that automatically generates random events and sends them to the device by means of the *sendEvent* tool. No contributions at all have been found regarding testing of native code components developed in C++ language using the Android NDK development framework. Moreover, no specific testing approaches focused on the automatic testing of components of Android applications such as services, broadcast receivers, and content providers have been found in literature.

Other testing issues for which few contributions have been found in literature are context-aware testing and concurrency testing.

Context-aware testing is an important issue for mobile applications due to the large availability of sources of contextual events on mobile devices (i.e., sensors, Internet connection, background services, etc.). Nevertheless, of this evidence, only 12 of the considered papers take into account these events (i.e., Amalfitano et al. 2013a; Gomez et al. 2013; Hu et al. 2015; Liang et al. 2014; Griebe and Gruhn et al. 2014, 2015; Adamsen et al. 2015; Song et al. 2015; Hu and Neamtiu 2016; Qin et al. 2016a; Yu and Takada 2016; Arnatovich et al. 2016). On the other hand, the interest on this field appears to be growing since most of these publications have been published in the last 2 years. The recent studies of de Sousa Santos et al. (2017) and Matalonga et al. (2017) have further highlighted this lack in the current literature.

Concurrency races represent the cause of many failures in mobile applications. In particular, although in the first Android versions the support to concurrency was quite limited,

a large support to develop concurrent Android applications is now available to developers, including the support for threads and asynchronous tasks. On the other hand, very few approaches devoted to the search of concurrency races from the tester point of view have been found in this study (in particular, only 5 papers have been found (Hsiao et al. 2014; Maiya et al. 2014; Tang et al. 2016; Hu et al. 2016; Li et al. 2016b)).

6.3 Scarce attention to fault modeling and finding

As observed by formulating an answer for the question RQ 3.1, the two more frequent testing targets of the papers included in this study are code coverage and failure detection, whereas bug finding and fixing received a very limited attention. In fact, the objective of more than 50 approaches is to find failures of mobile applications, including crashes, unhandled exceptions, concurrency races, and context-aware issues. On the other hand, only few papers attempted to find them by characterizing application faults. Only a single approach is focused on bug localization (the one of Machado et al. 2013) and only one other uses historical bug information from bug repositories to identify new bugs (Mendez-Porras et al. 2015a).

6.4 Distance between industry and academia

In our systematic mapping, we have found a very limited number of contributions from the industry or from collaborations between industry and academia. This can be due to the strategies followed by many companies that do not publish freely available testing tools. For example, the first three tools from academia that have been the basis for commercial projects (i.e., Testdroid (Kaasila et al. 2012), Caiipa (Liang et al. 2014), and MZoltar (Machado et al. 2013)) are not more freely available and no other academic publications regarding their evolution can be found in literature. Another example is related to Google, that has released several testing services for Android applications in the last years (e.g., the Espresso library, the Firebase Test Lab cloud environment, the Android Robo Test tool for automatic testing) but no academic publications demonstrating their effectiveness. On the other hand, information about these tools can be found in other forms, such as tutorials on the Android Developer web site[22] or videos from the Google I/O events[23].

In addition, in the academic papers, we have not found any validation experiment involving testing of real industrial applications during their development, but only experiments involving black box testing of already published industrial applications (Zeng et al. 2016). This observation confirms the conclusion highlighted in the systematic mapping of Zein et al. (2016) about the absence of case studies on large commercial applications during their life cycle. On the other hand, differently from other observations reported in Zein et al. (2016), the answer to the research question RQ 3.2 shows how a relevant number of testing techniques and tools have now been evaluated with respect to large sets of real applications available on public markets.

6.5 Comparative studies and testing benchmarks

The analysis of both primary studies considered in the systematic mapping and secondary studies described in Section 2 has shown that there is a relative lack of papers

[22] https://developer.android.com/guide/index.html
[23] https://events.google.com/io2016/

addressing comparison experiments involving different techniques and tools for mobile testing automation, but that this is an emerging topic.

Comparative experiments are included in few papers and the experiments that have been carried out present many limitations in terms of replicability and in terms of generalization of the conclusions. We have investigated about the existing issues making difficult a fair comparison between the available testing tools. A first issue is the rapid obsolescence of the academic tools available in literature that is primarily due to the very rapid evolution of the Android framework and of its supporting development environment that make continuous evolutive maintenance tasks on the testing tools necessary.

For this reason, it is difficult to design a testing harness able to compare different testing tools in the same environment and it is difficult, too, to select a set of applications on which all the tools under comparison can be executed.

In addition, most of the available testing tools are released in the form of prototypes without the possibility to customize all the characteristics of the tools. For example, in many tools, it is not possible to set preconditions on the applications under test.

Several recent works have tried to face these issues. The recent contribution of Choudhary et al. (2015) represents the better tentative to perform a fair comparison between the available tools. They have realized a test harness able to test different tools in the context of the same machine and in the same execution conditions. To this aim, they have contacted the authors of the tools published in literature up to 2014 and with their collaboration they have modified almost all the available open-source tools in order to execute them on a common Linux-based environment. They reported that they have been able to include only 7 tools on their experimentation: Monkey, Dynodroid (Machiry et al. 2013), Android Ripper (Amalfitano et al. 2012b), A3E (Azim and Neamtiu 2013), SwiftHand (Choi et al. 2013), PUMA (Hao et al. 2014), and ACTeve (Anand et al. 2012). Due to the rapid evolution of the Android framework, it is also very difficult to establish a common benchmark of applications. For example, Choudhary et al. (2015) attempted to form a testing benchmark by considering a set of applications that was previously tested in the papers presenting the tools under comparison. They collected 68 applications, but they admitted that only 51 of them resulted as executable on each of the seven considered testing tools.

A different approach has been followed by Amalfitano et al. (2017), that have focused their attention on testing techniques, by comparing the performance of several active learning and random techniques in the context of the same testing environment.

The problem of the comparison between the performance of academic tools and commercial tools is still open, since only the open-source tool Monkey has been involved in comparative experiments. For example, there are no experiments comparing the performance of commercial tools such as Android Robo Test[24] from Google or capture and replay tools such as Robotium Recorder[25], Espresso Test Recorder[26], and Ranorex Android Test Automation[27]. In the same way, there are no papers presenting comparisons with testing frameworks for testing iOS applications, such as Earl Grey[28] and Frank[29].

[24] https://firebase.google.com/docs/test-lab/robo-ux-test
[25] https://robotium.com/products/robotium-recorder
[26] https://developer.android.com/studio/test/espresso-test-recorder.html
[27] http://www.ranorex.com/mobile-automation-testing/android-test-automation.html
[28] https://opensource.google.com/projects/earlgrey
[29] https://github.com/TestingWithFrank/Frank

6.6 Absence of specific venues and journals focused on mobile testing automation

A relevant number of papers focused on mobile testing automation has been found in literature, in particular in the last years, as shown in Fig. 9. Sometimes, in the recent past, specific sessions of scientific conferences have been centered on some aspects of mobile testing automation (for example, the sessions called "UI Automation" at MobiSys 2014, "Quality Assurance" at Mobilesoft 2015, "Mobile GUI" at ISSRE 2015, "Android" at ISSTA 2016, "Testing Smartphone Applications" at AST 2016). In addition, sessions related to testing mobile applications have been often hosted by industrial events such as the Google Testing Automation Conference (GTAC)[30] that do not publish papers indexed by the considered search engines. Although this is a good level of interest of the scientific community, no specific venues have been dedicated so far to mobile testing automation or, more in general, to mobile testing. Analogously, no special issues on this topic have been already published in international journals. These considerations appear as an evidence of the temporary absence of a cohesive community of scientists involved in this topic.

7 Conclusions

The systematic mapping study presented in this paper provides a panorama of the state of the art of the scientific literature in the specific field of the automation of functional testing of mobile applications. This study presents a classification of the contributions provided by a set of 131 papers in literature, selected by applying an accurate strategy based on validated search queries and on the application of a set of inclusion and exclusion criteria. The systematic mapping study has been guided by 4 main goals and a total of 18 different research questions.

This study represents an upgrade with respect to the existing secondary studies in literature focused on the specific field of automation of functional testing of mobile applications. It can represent a useful tool for researchers, students, and practitioners in order to have an overall and detailed view of the state of the scientific literature related to the considered topic. For validation purposes, all the information necessary for replication of the study have been made available in this paper, as well as all the extracted data are available online at https://goo.gl/678T5P in the form of a Google spreadsheet. The online spreadsheet can be freely downloaded or commented on by readers that can propose edit operations to the authors, too.

The analysis of the systematic map has allowed the individuation of some research trends and some gaps in this research area. In particular, this study has found that there are few contributions from the industry and that there is a lack of contributions regarding specific topics such as techniques and tools for testing iOS applications, testing tools based on Android Espresso, and testing techniques aiming at testing of C++-based components of mobile applications. In addition, a limited attention on some topics including context-aware testing, concurrency testing, and fault detection has been revealed. Finally, the analysis of bibliography has demonstrated the absence of specific venues and journal focused on mobile testing automation. We are confident that these gaps can be filled by researchers in the next years.

[30] https://developers.google.com/google-test-automation-conference/

References

Adamsen, C.Q., Mezzetti, G., Moller, A. (2015). Systematic execution of android test suites in adverse conditions. In *Proceedings of the 2015 International Symposium on Software Testing and , ISSTA 2015* (pp. 83–93). New York: ACM. https://doi.org/10.1145/2771783.2771786.

Ahmad, A., Li, K., Feng, C., Asim, S.M., Yousif, A., Ge, S. (2018). An empirical study of investigating mobile applications development challenges. *IEEE Access*, *6*, 17:711–17:728. https://doi.org/10.1109/ACCESS.2018.2818724.

Akanksha Ashok Magare, M.D.L. (2016). Automated gui testing for android application. *Imperial Journal of Interdisciplinary Research*, *2*(8), 884–888.

Amalfitano, D., Fasolino, A.R., Tramontana, P. (2011). A gui crawling-based technique for android mobile application testing. In *ICST Workshops* (pp. 252–261).

Amalfitano, D., Fasolino, A., Tramontana, P., De Carmine, S., Imparato, G. (2012a). *A toolset for gui testing of android applications* (pp. 650–653). https://doi.org/10.1109/ICSM.2012.6405345, cited By 16.

Amalfitano, D., Fasolino, A.R., Carmine, S.D., Memon, A., Tramontana, P. (2012b). Using gui ripping for automated testing of android applications. In *ASE '12: Proceedings of the 27th IEEE international conference on Automated software engineering*. DC: IEEE Computer Society Washington.

Amalfitano, D., Fasolino, A., Tramontana, P., Amatucci, N. (2013a). Considering context events in event-based testing of mobile applications. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 126–133). https://doi.org/10.1109/ICSTW.2013.22.

Amalfitano, D., Fasolino, A., Tramontana, P., Robbins, B. (2013b). Testing android mobile applications: Challenges, strategies, and approaches. *Advances in Computers*, *89*, 1–52. https://doi.org/10.1016/B978-0-12-408094-2.00001-1.

Amalfitano, D., Amatucci, N., Fasolino, A., Tramontana, P. (2015a). *Agrippin: A novel search based testing technique for android applications* (pp. 5–12), https://doi.org/10.1145/2804345.2804348, cited By 1.

Amalfitano, D., Amatucci, N., Fasolino, A.R., Tramontana, P. (2015b). A conceptual framework for the comparison of fully automated gui testing techniques. In *Sixth International Workshop on Testing Techniques for Event BasED Software (TESTBEDS)* (pp. 2015).

Amalfitano, D., Amatucci, N., Fasolino, A.R., Tramontana, P., Kowalczyk, E., Memon, A. (2015c). Exploiting the saturation effect in automatic random testing of android applications. In *The Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (pp. 2015).

Amalfitano, D., Fasolino, A., Tramontana, P., Ta, B., Memon, A. (2015d). Mobiguitar: Automated model-based testing of mobile apps. *IEEE Software*, *32*(5), 53–59. https://doi.org/10.1109/MS.2014.55, cited By 29.

Amalfitano, D., Amatucci, N., Memon, A., Tramontana, P., Fasolino, A. (2017). A general framework for comparing automatic testing techniques of android mobile apps. *Journal of Systems and Software*, *125*, 322–343. https://doi.org/10.1016/j.jss.2016.12.017, cited By 0.

Amatucci, N. (2016). *Automated gui testing techniques for android applications*. PhD thesis, Ph.D. course in Information Technology and Electrical Engineering.

Anand, S., Naik, M., Yang, H., Harrold, M. (2012). Automated concolic testing of smartphone apps. No GIT-CERCS-12-02.

Anbunathan, R., & Basu, A. (2015). *A recursive crawler algorithm to detect crash in android application* (pp. 256–267), https://doi.org/10.1109/ICCIC.2014.7238518, cited By 0.

Anbunathan, R., & Basu, A. (2016a). Automatic test generation from uml sequence diagrams for android mobiles. *International Journal of Applied Engineering Research*, *11*(7), 4961–4970. cited By 0.

Anbunathan, R., & Basu, A. (2016b). Data driven architecture based automated test generation for android mobile. https://doi.org/10.1109/ICCIC.2015.7435772, cited By 0.

Arnatovich, Y.L., Ngo, M.N., Kuan, T.H.B., Soh, C. (2016). Achieving high code coverage in android ui testing via automated widget exercising. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 193–200), https://doi.org/10.1109/APSEC.2016.036.

Arzensek, B., & Hericko, M. (2014). *Criteria for selecting mobile application testing tools* (vol. 1266, pp. 1–8), cited By 0.

Azim, T., & Neamtiu, I. (2013). Targeted and depth-first exploration for systematic testing of android apps. *SIGPLAN Not*, *48*(10), 641–660. https://doi.org/10.1145/2544173.2509549.

Baek, Y.M., & Bae, D.H. (2016). Automated model-based android gui testing using multi-level gui comparison criteria (pp. 238–249), https://doi.org/10.1145/2970276.2970313, cited By 0.

Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S. (2015). The oracle problem in software testing: a survey. *IEEE Transactions on Software Engineering*, *41*(5), 507–525. https://doi.org/10.1109/TSE.2014.2372785.

Basili, V.R., Caldiera, G., Rombach, H.D. (1994). The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley.

Behrouz, R.J., Sadeghi, A., Garcia, J., Malek, S., Ammann, P. (2015). Ecodroid: An approach for energy-based ranking of android apps. In *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software* (pp. 8–14). https://doi.org/10.1109/GREENS.2015.9.

Bielik, P., Raychev, V., Vechev, M. (2015). Scalable race detection for android applications (vol. 25-30-Oct-2015, pp. 332–348). https://doi.org/10.1145/2814270.2814303, cited By 5.

Canfora, G., Mercaldo, F., Visaggio, C.A., DAngelo, M., Furno, A., Manganelli, C. (2013). A case study of automating user experience-oriented performance testing on smartphones. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation* (pp. 66–69). https://doi.org/10.1109/ICST.2013.16.

Choi, W., Necula, G., Sen, K. (2013). Guided gui testing of android apps with minimal restart and approximate learning. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages and applications* (pp. 623-640). ACM.

Choudhary, S.R., Gorla, A., Orso, A. (2015). Automated test input generation for android: Are we there yet? (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 429–440). https://doi.org/10.1109/ASE.2015.89.

Coelho, T., Lima, B., Faria, J.P. (2016). Mt4a: a no-programming test automation framework for android applications. In Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation (pp. 59–65), A-TEST 2016. New York: ACM. https://doi.org/10.1145/2994291.2994300.

Corral, L., Sillitti, A., Succi, G. (2015). Software assurance practices for mobile applications. *Computing*, *97*(10), 1001–1022. https://doi.org/10.1007/s00607-014-0395-8.

Costa, P., Paiva, A., Nabuco, M. (2014). Pattern based gui testing for mobile applications (pp 66–74). https://doi.org/10.1109/QUATIC.2014.16, cited By 6.

Crispin, L., & Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*, 1st edn. Addison-Wesley Professional.

de Sousa Santos, I., de Castro Andrade, R.M., Rocha, L.S., Matalonga, S., de Oliveira, K.M., Travassos, G.H. (2017). Test case design for context-aware applications: Are we there yet? Information and Software Technology (pp. –). https://doi.org/10.1016/j.infsof.2017.03.008.

De Cleva Farto, G., & Endo, A. (2015). Evaluating the model-based testing approach in the context of mobile applications. *Electronic Notes in Theoretical Computer Science*, *314*, 3–21. https://doi.org/10.1016/j.entcs.2015.05.002, cited By 2.

Delamaro, M., Vincenzi, A., Maldonado, J. (2006). A strategy to perform coverage testing of mobile applications. In *Proceedings of the 2006 international workshop on Automation of software test* (pp. 118–124). ACM,.

Dev, R., Jaaskelainen, A., Katara, M. (2012). Model-based gui testing. case smartphone camera and messaging development. *Advances in Computers*, *85*, 65–122. https://doi.org/10.1016/B978-0-12-396526-4.00002-3, cited By 2.

Do, Q., Yang, G., Che, M., Hui, D., Ridgeway, J. (2016). Regression test selection for android applications (pp. 27–28). https://doi.org/10.1145/2897073.2897127, cited By 0.

Dubinsky, Y., & Abadi, A. (2013). Challenges and research questions for testing in mobile development: Report on a mobile testing activity. In *Proceedings of the 2013 ACM Workshop on Mobile Development Lifecycle, MobileDeLi '13* (pp. 37–38). New York: ACM. https://doi.org/10.1145/2542128.2542140.

Dutia, S.N., Oh, T.H., Oh, Y.H. (2015). Developing automated input generator for android mobile device to evaluate malware behavior. In *Proceedings of the 4th Annual ACM Conference on Research in Information Technology, RIIT '15* (pp. 43–43). New York: ACM. https://doi.org/10.1145/2808062.2808065.

Fazzini, M., Freitas, E.N.D.A., Choudhary, S.R., Orso, A. (2017). Barista: A technique for recording, encoding, and running platform independent android tests. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)* (pp. 149–160). https://doi.org/10.1109/ICST.2017.21.

Gao, J., Bai, X., Tsai, W.T., Uehara, T. (2014). Mobile application testing: a tutorial. *Computer*, *47*(2), 46–55. https://doi.org/10.1109/MC.2013.445.

Garousi, V., Mesbah, A., Betin-Can, A., Mirshokraie, S. (2013). A systematic mapping study of web application testing. *Information and Software Technology*, *55*(8), 1374–1396. https://doi.org/10.1016/j.infsof.2013.02.006.

Garousi, V., & Mantyla, M.V. (2016). When and what to automate in software testing? a multi-vocal literature review. *Information and Software Technology*, *76*, 92–117. https://doi.org/10.1016/j.infsof.2016.04.015.

Gomez, L., Neamtiu, I., Azim, T., Millstein, T. (2013). Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, Piscataway, NJ, USA, ICSE '13* (pp. 72–81).

Gomez, M. (2015). Debugging of Mobile Apps in the Wild Guided by the Wisdom of the Crowd. In *2nd International Conference on Mobile Software Engineering and Systems - ACM Student Reasearch Competition*. Florence.

Gomez, M., Rouvoy, R., Adams, B., Seinturier, L. (2016). Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring (pp. 88–99). https://doi.org/10.1145/2897073.2897088, cited By 1.

Griebe, T., & Gruhn, V. (2014). A model-based approach to test automation for context-aware mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC '14* (pp. 420–427). New York:ACM. https://doi.org/10.1145/2554850.2554942.

Griebe, T., Hesenius, M., Gruhn, V. (2015). Towards automated ui-tests for sensor-based mobile applications. *Communications in Computer and Information Science*, *532*, 3–17. https://doi.org/10.1007/978-3-319-22689-7_1, cited By 0.

Griebe, T., Hesenius, M., Gesthuesen, M., Gruhn, V. (2016). Test Automation for Speech-Based Applications. In Fujita, H., & Papadopoulos, G.A. (Eds.) *New Trends in Software Methodologies, Tools and Techniques, Frontiers in Artificial Intelligence and Applications*, (Vol. 286 pp. 85–100). https://doi.org/10.3233/978-1-61499-674-3-85, 15th International Conference on New Trends in Intelligent Software Methodology Tools, and Techniques (SoMeT), CYPRUS, SEP 13-15, 2016.

Gudmundsson, V., Lindvall, M., Aceto, L., Bergthorsson, J., Ganesan, D. (2016). Model-based testing of mobile systems - an empirical study on quizup android app (vol. 208, pp. 16–30). https://doi.org/10.4204/EPTCS.208.2, cited By 0.

Hao, S., Liu, B., Nath, S., Halfond, W.G., Govindan, R. (2014). Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '14* (pp. 204–217). New York: ACM. https://doi.org/10.1145/2594368.2594390.

Harrison, R., Flood, D., Duce, D. (2013). Usability of mobile applications: literature review and rationale for a new usability model. *Journal of Interaction Science*, *1*(1), 1. https://doi.org/10.1186/2194-0827-1-1.

Hesenius, M., Griebe, T., Gries, S., Gruhn, V. (2014). Automating ui tests for mobile applications with formal gesture descriptions (pp. 213–222). https://doi.org/10.1145/2628363.2628391, cited By 4.

Holl, K., & Elberzhager, F. (2016). Quality assurance of mobile applications: A systematic mapping study. In *Proceedings of the 15th International Conference on Mobile and Ubiquitous Multimedia, MUM '16* (pp 101–113). New York:ACM. https://doi.org/10.1145/3012709.3012718.

Hsiao, C.H., Pereira, C., Yu, J., Pokam, G., Narayanasamy, S., Chen, P., Kong, Z., Flinn, J. (2014). Race detection for event-driven mobile applications. *ACM SIGPLAN Notices*, *49*(6), 326–336. https://doi.org/10.1145/2594291.2594330, cited By 3.

Hu, C., & Neamtiu, I. (2011a). Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test* (pp. 77–83).

Hu, C., & Neamtiu, I. (2011b). A gui bug finding framework for android applications (pp. 1490–1491). https://doi.org/10.1145/1982185.1982504, cited By 6.

Hu, G., Yuan, X., Tang, Y., Yang, J. (2014). Efficiently, effectively detecting mobile app bugs with appdoctor. In *Proceedings of the Ninth European Conference on Computer Systems, EuroSys '14* (pp. 18:1–18:15). New York: ACM. https://doi.org/10.1145/2592798.2592813.

Hu, Y., & Neamtiu, I. (2016). Fuzzy and cross-app replay for smartphone apps (pp. 50–56), https://doi.org/10.1145/2896921.2896925, cited By 0.

Hu, Y., Azim, T., Neamtiu, I. (2015). Versatile yet lightweight record-and-replay for android (vol. 25-30-Oct-2015, pp. 349–366), https://doi.org/10.1145/2814270.2814320, cited By 4.

Hu, Y., Neamtiu, I., Alavi, A. (2016). Automatically verifying and reproducing event-based races in android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016* (pp. 377–388). New York: ACM. https://doi.org/10.1145/2931037.2931069.

Imparato, G. (2015). A combined technique of gui ripping and input perturbation testing for android apps (vol. 2, pp. 760–762). https://doi.org/10.1109/ICSE.2015.241, cited By 0.

ISO 29119 Software Testing Standard (2013). Software and systems engineering software testing part 1: concepts and definitions. *ISO/IEC/IEEE*, *29119-1*(E), 1–64. https://doi.org/10.1109/IEEESTD.2013.6588537.

Jaaskelainen, A., Takala, T., Katara, M. (2012). *Model-Based Gui Testing of Android Applications, Addison-Wesley Professional (Pearson Education)* (pp. 253–275).

Jamrozik, K., & Zeller, A. (2016). Droid mate: A robust and extensible test generator for android (pp. 293–294). https://doi.org/10.1145/2897073.2897716, cited By 0.

Jensen, C.S., Prasad, M.R., Moller, A. (2013). Automated testing with targeted event sequence generation. In Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA (pp. 67–77). New York: ACM. https://doi.org/10.1145/2483760.2483777.

Jha, A., Lee, S., Lee, W. (2015). Modeling and test case generation of inter-component communication in android (pp. 113–116). https://doi.org/10.1109/MobileSoft.2015.24, cited By 2.

Jiang, B., Long, X., Gao, X. (2007). Mobiletest: A tool supporting automatic black box test for software on smart mobile devices. https://doi.org/10.1109/AST.2007.9, cited By 1.

Jiang, B., Chen, P., Chan, W.K., Zhang, X. (2016). To what extent is stress testing of android tv applications automated in industrial environments? *IEEE Transactions on Reliability*, *65*(3), 1223–1239. https://doi.org/10.1109/TR.2015.2481601.

Jiang, B., Zhang, Y., Chan, W.K., Zhang, Z. (2017). Which factor impacts gui traversal-based test case generation technique most? a controlled experiment on android applications. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 21–31). https://doi.org/10.1109/QRS.2017.12.

Joorabchi, M., Ali, M., Mesbah, A. (2016). Detecting inconsistencies in multi-platform mobile apps (pp. 450–460). https://doi.org/10.1109/ISSRE.2015.7381838, cited By 0.

Kaasila, J., Ferreira, D., Kostakos, V., Ojala, T. (2012). Testdroid: Automated remote ui testing on android. https://doi.org/10.1145/2406367.2406402, cited By 5.

Kirubakaran, B., & Karthikeyani, V. (2013). Mobile application testing 2014; challenges and solution approach through automation. In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME)* (pp. 79–84). https://doi.org/10.1109/ICPRIME.2013.6496451.

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Technical Report EBSE Technical Report EBSE-2007-01, Software Engineering Group School of Computer Science and Mathematics. Keele University and Department of Computer Science, University of Durham.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S. (2009). Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, *51*(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009.

Kochhar, P.S., Thung, F., Nagappan, N., Zimmermann, T., Lo, D. (2015). Understanding the test automation culture of app developers. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)* (pp. 1–10). https://doi.org/10.1109/ICST.2015.7102609.

Li, A., Qin, Z., Chen, M., Liu, J. (2014a). Adautomation: An activity diagram based automated gui testing framework for smartphone applications (pp. 68–77). https://doi.org/10.1109/SERE.2014.20, cited By 1.

Li, L., Bissyande, T., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., Klein, J., Le Traon, Y. (2016a). *Static analysis of android apps: A systematic literature review. Technical report, Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg, Fraunhofer SIT Technische Universitat Darmstadt*. Germany: University of Wisconsin and Pennsylvania State University.

Li, Q., Jiang, Y., Gu, T., Xu, C., Ma, J., Ma, X., Lu, J. (2016b). Effectively manifesting concurrency bugs in android apps. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 209–216). https://doi.org/10.1109/APSEC.2016.038.

Li, X., Jiang, Y., Liu, Y., Xu, C., Ma, X., Lu, J. (2014b). User guided automation for testing mobile apps. In Software engineering conference (APSEC), 2014 21st asia-pacific (Vol. 1, pp. 27–34). https://doi.org/10.1109/APSEC.2014.13.

Li, Y., Yang, Z., Guo, Y., Chen, X. (2017). Droidbot: A lightweight ui-guided test input generator for android. In *Proceedings of the 39th International Conference on Software Engineering Companion, ICSE-C '17* (pp. 23–26). Piscataway: IEEE Press. https://doi.org/10.1109/ICSE-C.2017.8.

Liang, C. J. M., Lane, N.D., Brouwers, N., Zhang, L., Karlsson, B.F., Liu, H., Liu, Y., Tang, J., Shan, X., Chandra, R., Zhao, F. (2014). Caiipa: Automated large-scale mobile app testing through contextual fuzzing. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking* (pp. 519–530). New York: ACM. https://doi.org/10.1145/2639108.2639131.

Lin, Y.D., Rojas, J., Chu, E.H., Lai, Y.C. (2014). On the accuracy, efficiency, and reusability of automated test oracles for android devices. Software Engineering. *IEEE Transactions on*, *40*(10), 957–970. https://doi.org/10.1109/TSE.2014.2331982.

Linares-Vasquez, M. (2015a). Enabling testing of android apps (Vol. 2, pp. 763–765). https://doi.org/10.1109/ICSE.2015.242, cited By 1.

Linares-Vasquez, M., White, M., Bernal-Cardenas, C., Moran, K., Poshyvanyk, D. (2015b). Mining android app usages for generating actionable gui-based execution scenarios (vol. 2015-August, pp. 111–122). https://doi.org/10.1109/MSR.2015.18, cited By 2.

Liu, Z., Gao, X., Long, X. (2010a). Adaptive random testing of mobile application. In *2010 2nd international conference on Computer engineering and technology (ICCET)* (Vol. 2, pp. V2–297). IEEE.

Liu, Z., Liu, B., Gao, X. (2010b). Test automation on mobile device (pp. 1–7). https://doi.org/10.1145/1808266.1808267, cited By 0.

Liu, Y., & Xu, C. (2013). Veridroid: Automating android application verification. https://doi.org/10.1145/2541534.2541594, cited By 0.

Liu, C.H., Lu, C.Y., Cheng, S.J., Chang, K.Y., Hsiao, Y.C., Chu, W.M. (2014a). Capture-replay testing for android applications. In *2014 International Symposium on Computer, Consumer and Control (IS3C)* (pp. 1129–1132). https://doi.org/10.1109/IS3C.2014.293.

Liu, Y., Lu, Y., Li, Y. (2014b). An android-based approach for automatic unit test (Vol. 2014), https://doi.org/10.1049/cp.2014.1290, cited By 0.

Liu, Y., Xu, C., Cheung, S.C., Yang, W. (2014c). Checkerdroid : Automated quality assurance for smartphone applications. *Int J Software and Informatics*, *8*, 21–41.

Liu, C.H., Chen, S.L., Chen, H.K. (2015). Robotdroid-a keyword-driven testing tool for android applications. *Frontiers in Artificial Intelligence and Applications*, *274*, 1865–1874. https://doi.org/10.3233/978-1-61499-484-8-1865, cited By 0.

Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., Zeng, L. (2017). Automatic text input generation for mobile testing. In *Proceedings of the 39th International Conference on Software Engineering, ICSE '17* (pp 643–653). Piscataway: IEEE Press. https://doi.org/10.1109/ICSE.2017.65.

Lu, L., Hong, Y., Huang, Y., Su, K., Yan, Y. (2012). Activity page based functional test automation for android application (pp. 37–40). https://doi.org/10.1109/WCSE.2012.15, cited By 3.

Ma, X., Wang, N., Xie, P., Zhou, J., Zhang, X., Fang, C. (2016). An automated testing platform for mobile applications (pp. 159–162). https://doi.org/10.1109/QRS-C.2016.25, cited By 0.

Machado, P., Campos, J., Abreu, R. (2013). Mzoltar: Automatic debugging of android applications (pp 9–16). https://doi.org/10.1145/2501553.2501556, cited By 3.

Machiry, A., Tahiliani, R., Naik, M. (2013). Dynodroid: an input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013* (pp. 224–234). New York: ACM. https://doi.org/10.1145/2491411.2491450.

Mahmood, R., Mirzaei, N., Malek, S. (2014). Evodroid: Segmented evolutionary testing of android apps. *In Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014* (pp. 599–609). New York: ACM. https://doi.org/10.1145/2635868.2635896.

Maiya, P., Kanade, A., Majumdar, R. (2014). Race detection for android applications. *SIGPLAN Not*, *49*(6), 316–325. https://doi.org/10.1145/2666356.2594311.

Majeed, S., & Ryu, M. (2016). Model-based replay testing for event-driven software (vol. 04-08-April-2016, pp. 1527–1533). https://doi.org/10.1145/2851613.2851794, cited By 0.

Maji, A., Arshad, F., Bagchi, S., Rellermeyer, J. (2012). An empirical study of the robustness of inter-component communication in android. In *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 1–12). https://doi.org/10.1109/DSN.2012.6263963.

Mao, K., Harman, M., Jia, Y. (2016). Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016* (pp. 94–105), New York: ACM. https://doi.org/10.1145/2931037.2931054.

Matalonga, S., Rodrigues, F., Travassos, G.H. (2017). Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review. *Journal of Systems and Software*, *131*(Supplement C), 1–21. https://doi.org/10.1016/j.jss.2017.05.048.

Mayan, A.J., Menezes, J.R., Bruce, J. (2015). Developing a mobile based automated testing tool for windows phone 8. *Modern Applied Science*, *9*, 91–98.

Mendez-Porras, A., Nieto Hidalgo, M., Garcia-Chamizo, J., Jenkins, M., Porras, A. (2015a). A top-down design approach for an automated testing framework. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9454*, 37–49. https://doi.org/10.1007/978-3-319-26401-1_4, cited By 0.

Mendez-Porras, A., Quesada-Lopez, C., Jenkins, M. (2015b). Automated testing of mobile applications: a systematic map and review. In Araujo, J., Condori-Fernandez, N., Goulão, M., Matalonga, S., Bencomo, N., Oliveira, T., de la Vara, J.L., Brito, I.S., Antonelli, L., Pimentel, E., Miranda, J.J., Kalinowski, M., Pastor, Ó., Olsina, L., Guizzardi, R., España, S., Cuadros-Vargas, E. (Eds.) *XVIII Ibero-american conference on software engineering, URP,SPC,UCSP* (pp. 195–208). Lima-Peru: UCSP.

Meng, Z., Jiang, Y., Xu, C. (2015). Facilitating reusable and scalable automated testing and analysis for android apps. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware, ACM, New York, NY, USA, Internetware '15* (pp. 166–175). https://doi.org/10.1145/2875913.2875937.

Mirzaei, N., Malek, S., Pasareanu, C.S., Esfahani, N., Mahmood, R. (2012). Testing android apps through symbolic execution. *SIGSOFT Softw Eng Notes*, *37*(6), 1–5. https://doi.org/10.1145/2382756.2382798.

Mirzaei, H., & Heydarnoori, A. (2015). Exception fault localization in android applications (pp 156–157). https://doi.org/10.1109/MobileSoft.2015.42, cited By 0.

Mirzaei, N., Bagheri, H., Mahmood, R., Malek, S. (2016a). Sig-droid: Automated system input generation for android applications (pp. 461–471). https://doi.org/10.1109/ISSRE.2015.7381839, cited By 0.
Mirzaei, N., Garcia, J., Bagheri, H., Sadeghi, A., Malek, S. (2016b). Reducing combinatorics in gui testing of android applications (Vol. 14-22-May-2016, pp. 559–570). https://doi.org/10.1145/2884781.2884853, cited By 0.
Moran K, Linares-Vasquez, M., Bernal-Cardenas, C., Vendome, C., Poshyvanyk, D. (2016). Automatically discovering, reporting and reproducing android application crashes. In 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST) (pp. 33–44). https://doi.org/10.1109/ICST.2016.34.
Moran, K., Linares-Vasquez, M., Bernal-Cardenas, C., Vendome, C., Poshyvanyk, D. (2017). Crashscope: A practical tool for automated testing of android applications. In *Proceedings of the 39th International Conference on Software Engineering Companion, ICSE-C '17* (pp. 15–18). Piscataway: IEEE Press. https://doi.org/10.1109/ICSE-C.2017.16.
Morgado, I., & Paiva, A. (2016a). Impact of execution modes on finding android failures (Vol. 83, pp. 284–291). https://doi.org/10.1016/j.procs.2016.04.127, cited By 0.
Morgado, I., & Paiva, A. (2016b). Testing approach for mobile applications through reverse engineering of ui patterns (pp 42–49). https://doi.org/10.1109/ASEW.2015.11, cited By 0.
Muccini, H., Di Francesco, A., Esposito, P. (2012). Software testing of mobile applications: Challenges and future research directions. In *2012 7th International Workshop on Automation of Software Test (AST)* (pp. 29–35). https://doi.org/10.1109/IWAST.2012.6228987.
Nagowah, L., & Sowamber, G. (2012). A novel approach of automation testing on mobile devices (Vol. 2, pp. 924–930). https://doi.org/10.1109/ICCISci.2012.6297158, cited By 3.
Nguyen, C., Marchetto, A., Tonella, P. (2012). Combining model-based and combinatorial testing for effective test case generation (pp. 100–110). https://doi.org/10.1145/04000800.2336765, cited By 25.
Packevicius, S., Usaniov, A., Stanskis, S., Bareisa, E. (2015). *The Testing Method Based on Image Analysis for Automated Detection of UI Defects Intended for Mobile Applications*, (pp. 560–576). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-24770-0_48.
Paulovsky, F., Pavese, E., Garbervetsky, D. (2017). High-coverage testing of navigation models in android applications. In *Proceedings of the 12th International Workshop on Automation of Software Testing, AST '17* (pp. pp 52–58). Piscataway: IEEE Press. https://doi.org/10.1109/AST.2017.6.
Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08* (pp. 68–77). Swinton: British Computer Society.
Petersen, K., Vakkalanka, S., Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: an update. *Information and Software Technology*, *64*, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007.
Prathibhan, C.M., Malini, A., Venkatesh, N., Sundarakantham, K. (2014). An automated testing framework for testing android mobile applications in the cloud. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies* (pp. 1216–1219). https://doi.org/10.1109/ICACCCT.2014.7019292.
Puspika, B., Hendradjaya, B., Danar Sunindyo, W. (2015). Towards an automated test sequence generation for mobile application using colored petri net (pp. 445–449). https://doi.org/10.1109/ICEEI.2015.7352542, cited By 0.
Qin, Y., Xu, C., Yu, P., Lu, J. (2016a). Sit: Sampling-based interactive testing for self-adaptive apps. *Journal of Systems and Software*, *120*, 70–88. https://doi.org/10.1016/j.jss.2016.07.002, cited By 0.
Qin, Z., Tang, Y., Novak, E., Li, Q. (2016b). Mobiplay: A remote execution based record-and-replay tool for mobile applications (Vol. 14-22-May-2016, pp. 571–582). https://doi.org/10.1145/2884781.2884854, cited By 1.
Raut, P., & Tomar, S. (2014). Android mobile automation framework. International Journal of Multidisciplinary Approach and Studies. 1–12.
Ravindranath, L., Nath, S., Padhye, J., Balakrishnan, H. (2014). Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '14*, pp. 190–203. ACM. https://doi.org/10.1145/2594368.2594377.
Reddy, K., Babu Rajesh, V., Pareek, H., Patil, M. (2016). Dynaldroid: A system for automated dynamic analysis of android applications (pp. 124–129). https://doi.org/10.1109/RAECE.2015.7510239, cited By 0.
Saad, N., & Awang Abu Bakar, N. (2014). Automated testing tools for mobile applications. https://doi.org/10.1109/ICT4M.2014.7020665, cited By 0.

Sadeh, B. (2011). A study on the evaluation of unit testing for android systems. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 4(1).

Sahinoglu, M., Incki, K., Aktas, M.S. (2015). *Mobile Application Verification: A Systematic Mapping Study*, (pp. 147–163). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-21413-9_11.

Salihu, I.A., & Ibrahim, R. (2016). Systematic exploration of android apps' events for automated testing. In *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media, MoMM '16* (pp. 50–54). New York: ACM. https://doi.org/10.1145/3007120.3011072.

Salva, S., & Laurencot, P. (2015). Model inference and automatic testing of mobile applications. In *International Journal On Advances in Software*, Vol. 8. Iaria.

San Miguel, J.L., & Takada, S. (2016). Gui and usage model-based test case generation for android applications with change analysis. In *Proceedings of the 1st International Workshop on Mobile Development, Mobile! 2016* (pp. 43–44), New York: ACM. https://doi.org/10.1145/3001854.3001865.

Shabaan, M.M., Hamza, H.S., Omar, Y. M. K. (2017). Effects of fsm minimization techniques on number of test paths in mobile applications mbt. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)* (pp. 297–302). https://doi.org/10.1109/SERA.2017.7965741.

Shan, Z., Azim, T., Neamtiu, I. (2016). Finding resume and restart errors in android applications. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016* (pp. 864-880). New York: ACM. https://doi.org/10.1145/2983990.2984011.

She, S., Sivapalan, S., Warren, I. (2009). Hermes: A Tool for testing mobile device applications. In *Software engineering conference, 2009. ASWEC'09* (pp. 121–130). Australian: IEEE.

Silva, D.B., Endo, A.T., Eler, M.M., Durelli, V. H. S. (2016). An analysis of automated tests for mobile android applications. In *2016 XLII Latin American Computing Conference (CLEI)* (pp. 1–9). https://doi.org/10.1109/CLEI.2016.7833334.

Song, K., Han, A.R., Jeong, S., Cha, S. (2015). Generating various contexts from permissions for testing android applications (Vol. 2015-January, pp. 87–92). https://doi.org/10.18293/SEKE2015-118, cited By 0.

Su, T. (2016). Fsmdroid: Guided gui testing of android apps. In *Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16* (pp. 689–691). New York: ACM. https://doi.org/10.1145/2889160.2891043.

Sun, C., Zhang, Z., Jiang, B., Chan, W.K. (2016). Facilitating monkey test by detecting operable regions in rendered gui of mobile game apps. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 298–306). https://doi.org/10.1109/QRS.2016.41.

Takala, T., Katara, M., Harty, J. (2011). Experiences of system-level model-based gui testing of an android application. In Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST '11 (pp. 377–386). Washington: IEEE Computer Society. https://doi.org/10.1109/ICST.2011.11.

Tang, H., Wu, G., Wei, J., Zhong, H. (2016). Generating test cases to expose concurrency bugs in android applications (pp. 648–653). https://doi.org/10.1145/2970276.2970320, cited By 0.

Tao, C., & Gao, J. (2016). On building test automation system for mobile applications using gui ripping (vol 2016-January, pp. 480–485). https://doi.org/10.18293/SEKE2016-168, cited By 0.

van der Merwe, H., van der Merwe, B., Visser, W. (2012). Verifying android applications using java pathfinder. *SIGSOFT Softw Eng Notes*, *37*(6), 1–5. https://doi.org/10.1145/2382756.2382797.

Vilkomir, S., & Amstutz, B. (2014). Using combinatorial approaches for testing mobile applications. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops* (pp. 78–83). https://doi.org/10.1109/ICSTW.2014.9.

Vilkomir, S., Marszalkowski, K., Perry, C., Mahendrakar, S. (2015). Effectiveness of multi-device testing mobile applications. In *2015 2nd ACM International Conference on Mobile Software Engineering and Systems* (pp. 44–47). https://doi.org/10.1109/MobileSoft.2015.12.

Wang, P., Liang, B., You, W., Li, J., Shi, W. (2014). Automatic android gui traversal with high coverage. In *Proceedings of the 2014 Fourth International Conference on Communication Systems and Network Technologies, CSNT '14* (pp. 1161–1166). Washington: IEEE Computer Society. https://doi.org/10.1109/CSNT.2014.236.

Wen, H.L., Lin, C.H., Hsieh, T.H., Yang, C.Z. (2015). Pats: a parallel gui testing framework for android applications. In *2015 IEEE 39Th annual computer software and applications conference* (Vol. 2. pp. 210–215). https://doi.org/10.1109/COMPSAC.2015.80.

White, M., Linares-Vasquez, M., Johnson, P., Bernal-Cardenas, C., Poshyvanyk, D. (2015). Generating reproducible and replayable bug reports from android application crashes. In *2015 IEEE 23rd International Conference on Program Comprehension (ICPC)* (pp. 48–59). https://doi.org/10.1109/ICPC.2015.14.

Wu, X., Jiang, Y., Xu, C., Cao, C., Ma, X., Lu, J. (2016). Testing android apps via guided gesture event generation. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 201–208). https://doi.org/10.1109/APSEC.2016.037.

Yang, W., Prasad, M.R., Xie, T. (2013). A grey-box approach for automated gui-model generation of mobile applications. In *Fundamental Approaches to Software Engineering* (pp. 250–265). Springer.

Ye, H., Cheng, S., Zhang, L., Jiang, F. (2013). Droidfuzzer: Fuzzing the android apps with intent-filter tag. In Proceedings of International Conference on Advances in Mobile Computing and Multimedia, MoMM '13 (pp. 68:68–68:74). New York: ACM. https://doi.org/10.1145/2536853.2536881.

Yeh, C.C., Lu, H.L., Chen, C.Y., Khor, K.K., Huang, S.K. (2014). Craxdroid: Automatic android system testing by selective symbolic execution (pp. 140–148). https://doi.org/10.1109/SERE-C.2014.32, cited By 3.

Yoo, H., & Lee, Y. (2017). An automatic mobile app. testing method with user event scenario. In *2017 18th IEEE International Conference on Mobile Data Management (MDM)* (pp. 394–396). https://doi.org/10.1109/MDM.2017.71.

Yu, S., & Takada, S. (2016). Mobile application test case generation focusing on external events. In *Proceedings of the 1st International Workshop on Mobile Development, Mobile! 2016* (pp. 41-42). New York: ACM. https://doi.org/10.1145/3001854.3001864.

Zaeem, R.N., Prasad, M.R., Khurshid, S. (2014). Automated generation of oracles for testing user-interaction features of mobile apps. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation, ICST '14* (pp. 183–192). Washington: IEEE Computer Society. https://doi.org/10.1109/ICST.2014.31.

Zein, S., Salleh, N., Grundy, J. (2016). A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software*, *117*, 334–356. https://doi.org/10.1016/j.jss.2016.03.065.

Zeng, X., Li, D., Zheng, W., Xia, F., Deng, Y., Lam, W., Yang, W., Xie, T. (2016). Automated test input generation for android: Are we really there yet in an industrial case?. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016* (pp. 987–992). New York: ACM. https://doi.org/10.1145/2950290.2983958.

Zhang, S., & Pi, B. (2015a). Mobile functional test on taas environment. In *2015 IEEE Symposium on Service-Oriented System Engineering* (pp. 315–320). https://doi.org/10.1109/SOSE.2015.27.

Zhang, T., Gao, J., Cheng, J., Uehara, T. (2015a). Compatibility testing service for mobile applications. In *2015 IEEE Symposium on Service-Oriented System Engineering* (pp. 179–186). https://doi.org/10.1109/SOSE.2015.35.

Zhang, A., He, Y., Jiang, Y. (2016). Crashfuzzer: Detecting input processing related crash bugs in android applications. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)* (pp. 1–8). https://doi.org/10.1109/PCCC.2016.7820625.

Zhauniarovich, Y., Philippov, A., Gadyatskaya, O., Crispo, B., Massacci, F. (2015). Towards black box testing of android apps. In *2015 10th International Conference on Availability, Reliability and Security (ARES)* (pp. 501–510). https://doi.org/10.1109/ARES.2015.70.

Zheng, H., Li, D., Liang, B., Zeng, X., Zheng, W., Deng, Y., Lam, W., Yang, W., Xie, T. (2017). Automated test input generation for android: Towards getting there in an industrial case. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP '17* (pp. 253–262). Piscataway: IEEE Press. https://doi.org/10.1109/ICSE-SEIP.2017.32.

Zhu, H., Ye, X., Zhang, X., Shen, K. (2015). A context-aware approach for dynamic gui testing of android applications (vol 2, pp. 248–253). https://doi.org/10.1109/COMPSAC.2015.77, cited By 0.

Zun, D., Qi, T., Chen, L. (2016). Research on automated testing framework for multi-platform mobile applications. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)* (pp. 82–87). https://doi.org/10.1109/CCIS.2016.7790229.

Porfirio Tramontana is an Assistant Professor at the University of Naples Federico II. He graduated in Computer Engineering in 2001 and had a Ph.D. degree in 2005 at the same university. His research is focused on Software Engineering applied to Mobile and Web applications. His research fields include reverse engineering, testing, maintenance, comprehension, migration of legacy systems, and software quality.

Domenico Amalfitano is a postdoctoral researcher at the University of Naples Federico II. His research mainly concerns the reverse engineering, comprehension, migration, testing, and testing automation of event-driven software systems, mostly for web applications, mobile applications, and GUIs. Amalfitano received a PhD in Computer Engineering and Automation from the University of Naples Federico II.

Nicola Amatucci is a postdoctoral researcher at the University of Naples Federico II. His research interests are mainly related to event-based testing and mobile application testing. He received a PhD in Computer Engineering and Automation from the University of Naples Federico II in 2016.

Anna Rita Fasolino is an Associate Professor in the Department of Electrical Engineering and Information Technology of the University of Naples Federico II. Her research interests include software testing, software maintenance, reverse engineering, embedded software engineering, and mobile and web engineering. Prof. Fasolino has coordinated several research and technology transfer projects most of them conducted in cooperation with industrial partners. She has served on program committees for numerous events in the field of software engineering.

Affiliations

Porfirio Tramontana[1] · Domenico Amalfitano[1] · Nicola Amatucci[1] · Anna Rita Fasolino[1]

Domenico Amalfitano
domenico.amalfitano@unina.it

Nicola Amatucci
nicola.amatucci@unina.it

Anna Rita Fasolino
fasolino@unina.it

[1] Department of Electrical Engineering and Information Technology, University of Naples Federico II, Naples, Italy