GUI TESTING OF ANDROID APPLICATIONS: A SYSTEMATIC MAPPING

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF INFORMATICS
INSTITUTE OF MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUZAFFER AYDIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN THE DEPARTMENT OF INFORMATION SYSTEMS

DECEMBER 2014

Approval of the Graduate School of Informatics

GUI TESTING OF ANDROID APPLICATIONS: A SYSTEMATİC MAPPING

Submitted by **MUZAFFER AYDIN** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife BAYKAL
Director, **Informatics Institute, METU**

Prof. Dr. Yasemin YARDIMCI ÇETİN
Head of Department, **Information Systems, METU**

Assoc. Prof. Dr. Aysu BETİN CAN
Advisor, **Information Systems, METU**

Assoc. Prof. Dr. Vahid GAROUSI YUSİFOĞLU
Co-Advisor, **Software Eng., Atılım University**

Examining Committee Members:

Prof. Dr. Yasemin YARDIMCI ÇETİN
Head of Department, IS, METU

Assoc. Prof. Dr. Aysu BETİN CAN
IS, METU

Assist. Prof. Dr. Sadık EŞMELİOĞLU
CENG, Çankaya University

Assoc. Prof. Dr. Banu GÜNEL
IS, METU

Assoc. Prof. Dr. Pınar KARAGÖZ
CENG, METU

Date: 09.12.2014

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: MUZAFFER AYDIN

Signature

ABSTRACT

GUI TESTING OF ANDROID APPLICATIONS: A SYSTEMATIC MAPPING

AYDIN, MUZAFFER

M.Sc., Department of Information Systems

Co-Advisors: Assoc. Prof. Dr. AYSU BETİN CAN

Assoc. Prof. Dr. VAHID GAROUSI

December 2014, 78 Pages

Popularity of mobile devices is increasing rapidly all around the world. These devices can be used on various systems which are commonly used by the society. These systems are predicted to overtake desktop platform's popularity in the near future. Therefore the quality of mobile applications has vital importance. High quality applications can only be developed with good testing environments. Considering that multi-featured mobile applications have complex user interfaces, we decided to focus on graphical user interface (GUI) testing. We chose Android operating system (OS) as our platform which is the most popular one.

We have conducted a systematic mapping study that reviews the literature in area of GUI testing of Android applications. We have used goal-question-metric (GQM) paradigm. Through our goal, we have asked three main questions and their sub-questions as our research questions (RQs). Then, we have collected the articles related the domain since 2009 when first stable version of Android released until October 11th, 2014. We have applied them our inclusion/exclusion criteria to bring out our final article set which consist of 27 articles. We have prepared a classification scheme to extract data from given articles. Finally, the extracted data is used to gather the results to present a general idea of trends and maturity level of this particular area.

Keywords: Systematic mapping, Android Testing, GUI Testing

ÖZ

ANDROID UYGULAMALARINDA KULLANICI ARAYÜZÜ TESTİ: SİSTEMATİK ADRESLEME

AYDIN, MUZAFFER

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Eş Danışmanları: Doç. Dr. AYSU BETİN CAN

Doç. Dr. VAHID GAROUSİ

Aralık 2014, 78 Sayfa

Dünya çapında mobil cihazların popülerliği gün geçtikçe artmaktadır. Bu cihazlar, toplum tarafından sıkça kullanılan birçok sistemde kullanılabilmektedirler. Hatta yakın zamanda bu sistemlerin masaüstü platformunun yerini alacağı düşünülmektedir. Bu yüzden, mobil uygulamaların kalitesi ciddi bir öneme sahiptir ve yüksek kalite uygulamalar ancak yeterli test sistemlerinin mevcut olduğu ortamlarda mümkündür. Bu yüzden, yetenekli uygulamaların karmaşık kullanıcı arayüzlerine sahip olduğunu da göz önünde bulundurarak, kullanıcı arayüzü test etme konusuna odaklanmaya kara verdik. İşletim sistemi olarak da şuanda en popüler işletim sistemi olan Android'i seçmeye karar verdik.

Bu çalışma, Android uygulamalarında kullanıcı arayüzü testi konusunda literatürü tarayan bir sistematik adresleme çalışmasıdır. Amaç-soru-ölçüm tekniğini kullanılmıştır. Belirlediğimiz amaç doğrultusunda, araştırma sorularımız üç temel soru ve onların alt sorularından oluşmaktadır. Bu konu hakkında ki Android'in ilk sürümünün piyasaya sunulduğu yıl olan 2009'dan itibaren 11.11.2014 tarihine kadar yayınlamış tüm makaleleri topladıktan sonra eleme kriterlerine göre değerlendirerek, toplamda 27 makale den oluşan bir makale kümesi oluşturulmuştur. Bu makalelerden bilgi toplamak amacıyla bir sınıflama şeması oluşturulmuştur. Sonuç olarak, hem genel fikir bir sunmak hem de genel akımları ve bu alanın olgunluk seviyesini ölçmek için bu bilgiler ışığında ulaşılan sonuçlar paylaşılmıştır

Anahtar Kelimeler: Sistematik adresleme, Android test etme, Kullanıcı arayüzü test etme

To My Fiancée

ACKNOWLEDGEMENTS

I am deeply grateful to my co-advisors Assist. Prof. Dr. Aysu BETİN CAN and Assoc. Prof. Dr. Vahid GAROUSI who have helped during this process with patience. They gave me invaluable suggestions throughout this study.

I am also thankful to my fiancée Seda and my parents Meliha and Sedat for their patience and concern during this process.

My colleagues from Aselsan were also very tolerant and understanding during my study. I am also very grateful to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SM	Systematic Mapping
SLR	Systematic Literature Review
GUI	Graphical User Interface
OS	Operating System
CPU	Central Processing Unit
JVM	Java Virtual Machine
DVM	Dalvik Virtual Machine
XML	Extensible Markup Language
RQ	Research Question
URL	Uniform Resource Locator
SDK	Software Development Kit
AUT	Application Under Test
API	Application Programming Interface
CPA	Citation Normalization Per Article

CHAPTER 1

INTRODUCTION

Popularity of mobile devices is increasing rapidly all around the world. These devices can be used on various systems. While a cell phone is a mobile device, a smart-home system can be a more complex use of the mobile devices. The variation and adaptation capabilities of this concept makes it a better alternative to other possibilities and it seems that overtaking the desktop platform's popularity by these novel systems will not take too much time.

The principal explosive growth on mobile platform is caused by the evolution of smartphones. Actually smartphone concept is not a new concept. According to [1], in 1992 IBM has announced the first smartphone (called Simon). However neither CPU powers nor wireless network capabilities were adequate enough. With the improvements of these technologies, sales of smartphones are improved dramatically [3]. The main reason of this widespread usage of smartphones may be the application market concepts of the leading mobile operating systems like Android, IOS etc. In 2013, Google Inc. has announced that there are over 1 million applications in their market (Google Play) [9]. Apple side also declared a similar announcement after a few months later [10]. There are so many applications provided at these markets for various purposes. Application developers have to create multi-feature and comprehensive applications in order to come to fore among similar applications. On the other hand, most of these applications are typically developed by semi-professional developers or small companies that have not so much work power. So the major drawback of these applications is their quality. Testing processes are usually ignored or made with limited resources because of many reasons such as budget, time, and work power. Thus, aside from some good quality applications, the results are generally unsatisfactory.

Currently, Android is the most trendy mobile operating system in the smartphone market. According to Strategy Analytics's latest report [1], it takes 85 percentage of global smart phone shipment share and it is still growing. The Android operating system is built on Linux Kernel and is currently being developed by Google. There are four layers in Android. At the bottom, there is a hardware abstraction layer which is Linux kernel itself. Next layer consists of a native collection of C and C++ libraries whose features are used by upper Java libraries. On the top of this layer, there is a specialized implementation of

Java Virtual Machine (JVM), called Dalvik Virtual Machine (DVM). The last layer is the Application Framework Layer which contains the applications. Android applications are Java-based, event-driven applications supported by rich graphical user interfaces (GUIs). The applications consist of four main components; activities, services, broadcast receivers and content providers.

Screens of Android applications are XML-based and generally run on touch-screens. There are lots of events that trigger the GUI different from the traditional applications such as desktop GUI applications, web applications. This novelty makes the process of testing android applications more complicated. The main issue is that of assessing which testing approach-tool combination is applicable for verifying the application.

GUI testing, which is a practice of testing application with GUI events, is based on correctness of GUI states and behaviors. This testing process also verifies the data handling, control flows, states etc. Ease of GUI testing depends on several factors like tools, approach, criteria. Testers need to have experience about all of the factors in order to choose suitable one for their cases.

Since the earlier versions of Android to the latest one, researchers have proposed many techniques or tools about Android GUI testing. Therefore we have decided to make a systematic mapping (SM) study in order to get together the body of knowledge about this area in a public resource. Our motivation was inferring current trends about GUI testing approaches, tools, and techniques on Android OS. We have also intended to identify the points that are not explored enough and propose directions for future studies that are required in order to bridge the gap in the domain. We have aimed that our study will be a guideline for researchers, testers, and test tool developers.

In our SM, we have used goal-question-metric (GQM) paradigm [26]. Through our goal, we have asked three main questions and their sub-questions as our research questions (RQs). Then we have conducted a comprehensive research in order to collect all related articles published between 2009 when first stable version of Android released to October 11th, 2014. As a result of this research, we have gathered 59 studies. Then we have applied our inclusion/exclusion criteria to these studies. Finally we have obtained the final pool of article which consists of 27 studies. We have prepared a classification scheme to extract data from given articles. We have addressed our RQs using the data extracted from the papers. Finally, we have presented the results.

Outline of this thesis as follows. We present the background knowledge of our area and the works related to our study in section 2. In section 3, we give a detailed definition about our research methodology that contains our research questions, inclusion/exclusion criteria and the article set. We demonstrate the results of our study in Section 4. The discussion is located in Section 5. Finally the conclusion part is in Section 6.

CHAPTER 2

BACKGROUND & RELATED WORK

2.1. Systematic Mapping in Software Engineering

A systematic mapping study is an approach that is accepted by the software engineering communities which focuses on the literature research in order to make a general conclusion over primary studies in a specific domain [13, 21]. The result of this study is generally categorizes the primary studies according to a classification scheme.

We have adapted the process of systematic mapping study from [13]. First step of this study is specifying a goal and proper research questions (RQs). This step specifies the scope of the study. Then a comprehensive search is conducted to collect all studies in this scope. After that in order to eliminate the irrelevant papers, well-described inclusion and exclusion criteria are applied to these studies. As a result, the final set of primary studies is obtained. In order to extract data from the articles, a classification scheme is required. This scheme is created with the aim of taking all of the articles into account. This process may be iterative to the refinements on the scheme. When the scheme is ready, the data is extracted from the articles. Applying a peer review on the extracted data reduces the bias on them. The data collected from these studies is a repository that summarizes the whole set. The SM researcher can address the RQs using the repository. Finally, it is intended to find out unbiased, comprehensive result.

2.2. Structure of Android Applications

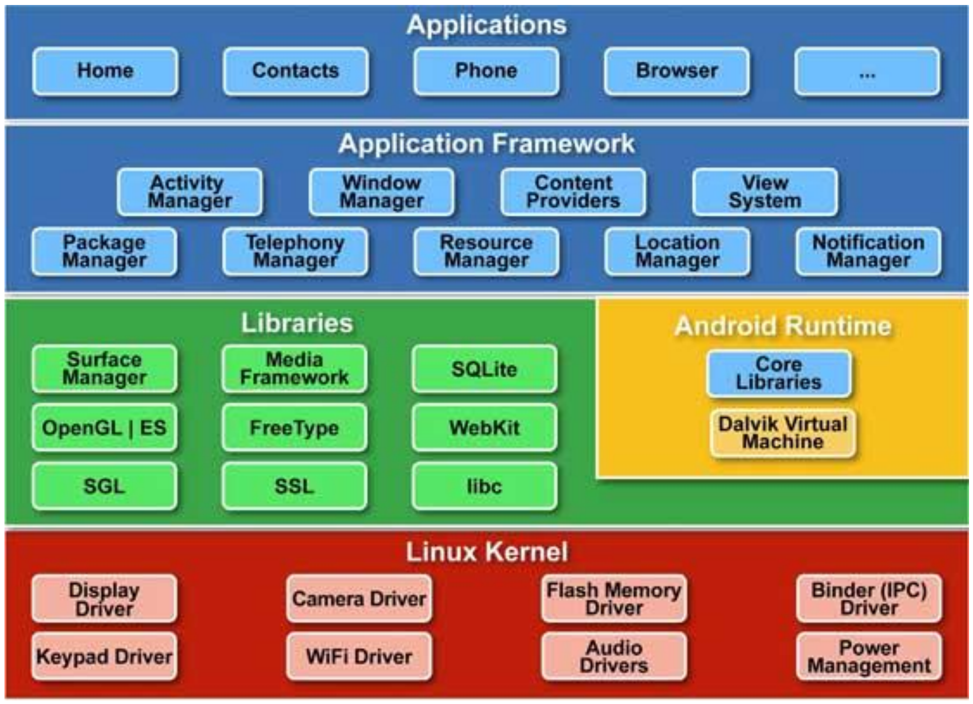


Figure 1 - Android Structure [4]

Android operating systems consists of four layers as depicted in Figure 1.Android applications are located at the top of last layer of Android architecture (Figure 1) [4]. They work on Dalvik Virtual Machine (DVM) which is a kind of Java Virtual Machine (JVM) especially optimized and designed for Android OS. DVM allows the use of some basic Linux features which are essential for Java language like memory management and multi-threading. Therefore the applications are developed using Java with extended specific libraries from Application Framework Layer.

The main components of an Android application are activity, service, content provider and broadcast receiver [5].

- **Activity:** An Activity is an application component that provides a GUI. Interaction between the user of the application and the GUI is handled by the activities. Activities called from the activity stack and only one of the activities can be upfront. Each activity has a lifecycle and many states (fig 2). If one of the activities is shut down, the activity manager pops another activity from the activity stack.

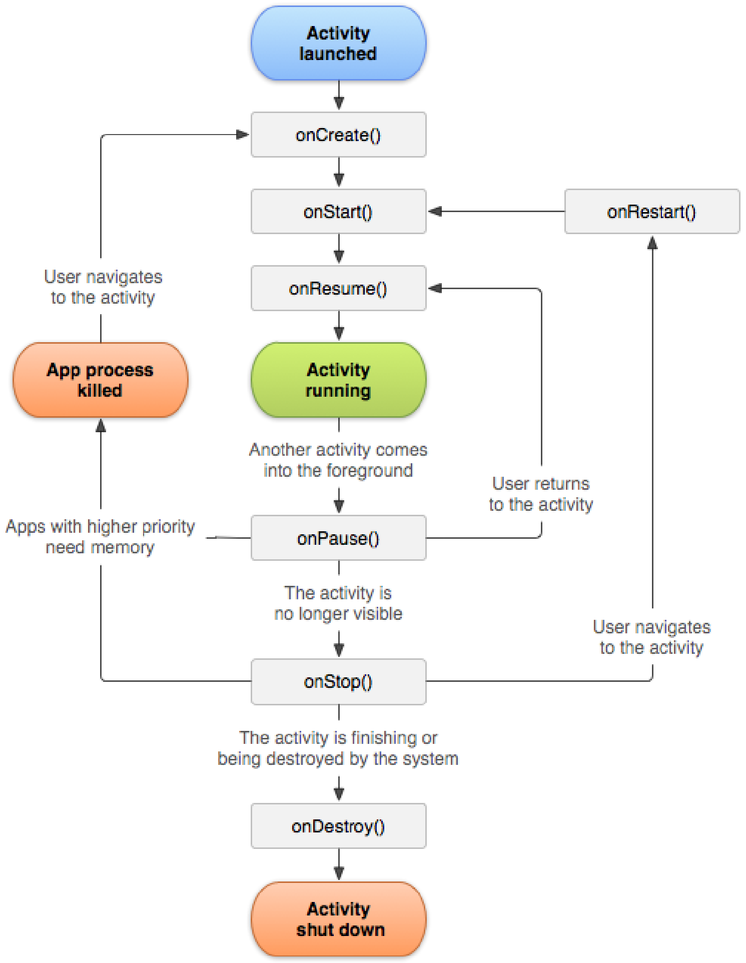

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2 - Activity Lifecycle [5]

- **Service:** A service is an application component that can execute long term operations in the background. It does not have a GUI.A service can be started by other application components and it keeps running in the background even if the application is stopped or switched to another application. A service is typically used to handle network operations, play music, or content provider interactions.
- **Content provider:** Content providers are the standard interfaces that controls flow of structured data between processes. Data encapsulation and data security are provided by this component.
- **Broadcast Receivers:** Broadcast Receivers are the components that catch broadcast messages from the system such as system start-up, SMS indication, or battery warnings. A custom broadcast massage can also be fired from an application.

2.3. GUI Testing in Android

As the Android is primarily used in devices with touch screens such as smartphones or tablets, the GUIs of applications are optimized for this use. In native android development, the view is based on XML files and it has predefined widgets. The developer can also create custom widgets using base classes.

There are three GUI testing tools in Android software development kit (SDK). The primary GUI testing tool in SDK is the Android instrumentation framework. This tool enables separate testing applications to run in same process with application under test (AUT) if all of them have an access to a common application context.

Another testing tool is the Monkey testing tool [6]. It is capable of sending pseudo-random user events such as tapping the screen, pressing a button or gestures to screen as well as creating system-level events. Therefore it is useful while testing of stability of the applications for crashes.

The last tool in the Android SDK is the Monkey Runner tool [7] which has an API in order to create applications to manage an Android system from outside of native code. This tools enables writing scripts e.g. python scripts which are capable to do main functions such as installing and running an application, sending user events and gestures taking screenshots.

All of these tools can be extendible and combinable with each other or other applications that are created for testing purposes. There are many examples for smarter applications which are successfully serving as comprehensive testing tools. They can be also improved for future needs which are not covered yet.

2.4. Related Work

To the best of our knowledge, there is no systematic mapping study on Android GUI testing domain other than this one. On the other hand, as the systematic mapping approach is been well-accepted and popular in software engineering as well as other secondary study approaches such as systematic literature review, or taxonomies can be found with a minor effort. We collected and discussed some related work below.

We present the related work under 3 topics: (1) published secondary studies about software engineering, (2) repositories, databases about software engineering, (3) secondary studies that focuses on GUI testing.

2.4.1. Secondary Studies in Software Testing

Doğan et al. [24] give a list of secondary studies in software engineering (SE) domain in their study which is a systematic literature review (SLR) about web application testing (WAT). Their research, which has no starting year, presents that the secondary studies like SMs or SLRs has an exponential-like growing curve in their counts. They have been able to find 24 secondary studies. There have categorized them as eight SMs, five SLRs, and remaining eleven studies for taxonomies, literature reviews, analysis and surveys etc. We have searched for secondary studies in SE published up to 2014. We were able to find 58 secondary studies on SE. 14 of these studies are SMs, 21 of them are SLSs, and 23 of them are regular surveys. We have listed and categorize them in Table 1.

(NER: Not explicitly reported, NOR: Number of references)

Table 1- Secondary studies in software testing

Type of Secondary Study	Secondary Study Area	Year of publication	Num. of Primary Studies	Ref.
SM (n=14)	Search-based testing for non-functional system properties	2008	35	[40]
	Product lines testing	2011	45	[44]
	Product lines testing	2011	64	[47]
	Alignment of requirements and testing	2011	35	[42]
	Testing in service-oriented architecture (SOA)	2011	33	[41]
	Automated tool support for unit testing	2012	136	[65]
	Static and dynamic quality assurance techniques	2012	51	[72]
	Testing web services	2012	150	[89]
	Tools for product lines testing	2012	33	[47]
	Reducing test effort	2012	144	[95]
	Web application testing	2013	79	[61]
	Graphical user interface (GUI) testing	2013	136	[62]
	Test-case prioritization	2013	120	[87]

	Testing of matlab simulink models	2013	44	[16]
	Functional Software Testing	2013	27	[20]
	Knowledge management initiatives in software testing	2014	13	[15]
	Software Development Documentation	2014	60	[33]
	Software test-code engineering	2014	60	[36]
SLR (n=21)	Model-based testing	2007	202	[70]
	Automated acceptance testing	2008	8	[75]
	Testing aspect-oriented programs	2008	43	[86]
	Search-based testing for non-functional system properties	2009	35	[45]
	Concurrent software testing	2009	109	[93]
	Unit testing for Business Process Execution Language (BPEL)	2009	27	[46]
	Integration testing of component-based software	2010	49	[71]
	Empirical investigation of search-based test-case generation	2010	68	[49]
	Human Factors in Software Development	2010	67	[35]
	Formal testing of web services	2010	37	[43]
	Regression test selection techniques	2010	27	[50]
	Concurrent software testing	2011	188	[80]
	Testing in adherence to the DO-178B Standard	2011	97	[83]

	Benefits and limitations of automated testing	2012	25	[76]
	Empirical evaluation of cloud-based testing	2012	38	[12]
	Regression test prioritization techniques	2012	65	[84]
	Automatic test-case generation from UML diagrams	2012	42	[85]
	State-based test tools	2013	12	[73]
	Ontologies in software testing	2013	18	[14]
	Software product line testing	2013	23	[82]
	Mutation testing for Aspect-J programs	2013	10	[74]
	Web application testing	2014	95	[24]
	Automated Testing	2014	58	[34]
	Testing scientific software	2014	49	[97]
Regular survey (n=23)	Testing finite state machines	1996	NER (NOR=153)	[79]
	Testing object-oriented (OO) software	1996	140	[51]
	Testing communication protocols	2002	NER (NOR=60)	[67]
	Empirical studies about testing technique	2004	36	[52]
	Search-based test data generation	2004	73	[53]
	Combinatorial testing	2005	30	[54]
	UML-based coverage criteria for software testing	2005	NER (NOR=31)	[69]
	Integration testing	2007	NER (NOR=84)	[90]

	Symbolic execution for software testing	2009	70	[56]
	SOA testing	2009	64	[55]
	Testing using model checkers	2009	NER (NOR=140)	[91]
	Model-based GUI testing	2010	NER (NOR=42)	[59]
	Test-driven development of user interfaces	2010	NER (NOR=36)	[60]
	Cost reduction of mutation testing	2010	NER (NOR=28)	[78]
	Testing web services	2010	NER (NOR=86)	[57]
	Combinatorial testing	2011	90	[66]
	Mutation testing	2011	390	[58]
	Search-based software testing	2011	NER (NOR=58)	[81]
	Software testing in the cloud	2012	NER (NOR=58)	[68]
	Regression testing minimization, selection and prioritization: a survey	2012	NER (NOR=189)	[96]
	Testing in SOA	2013	177	[88]
	Test-case generation from UML behavioral models	2013	NER (NOR=82)	[94]
	Test oracles	2014	611	[64]

2.4.2. Online repositories in Software Engineering

Some of secondary studies provide online repositories that contain the data extracted from the primary studies in their pool. These repositories are kept up to date with regular contributions of the authors. The repository approach has many advantages for new researchers such as providing them a reliable starting point, following latest trends, or opportunities to interpret the data in different perspectives.

The number of repositories is too few when it is compared to the number secondary studies. We have found seven online repositories. We have listed them in Table 2.

Table 2- Online repositories in software engineering

Topic	**Number of elements**	**Ref.**
Mutation Testing	424	[27]
Search based Software Engineering	1261	[28]
Software Test-Code Engineering	60	[19]
GUI Testing	143	[18]
Developing Scientific Software	141	[17]
Web Application Testing	95	[23]
Testing of Web Services[1]	150	[98]

2.4.3. Secondary Studies about GUI Testing

In [21], Banerjee et al. present a systematic mapping study on the area of graphical user interface (GUI) testing. In the study, they use goal-question-metric (GQM) paradigm [26] and they firstly prepared five goals. These goals are generally about three main points; (1)revealing the latest trends in GUI testing research and evaluation, (2) demographic and bibliometric information about articles and authors, and (3) the limitations, and directions for future studies in GUI testing area. They collect data from several articles in order to classify the studies towards these questions. They also publish an online repository about GUI Testing [18].

Memon et al. present a taxonomy study for GUI testing techniques [29]. This study focuses on the classification of model-based techniques whether they are manual or automatic. These techniques build up a model from the GUI and utilize it in order to produce test cases. Each technique is demonstrated on a small application to compare their strengths and weaknesses.

[1] Not currently available

CHAPTER 3

RESEARCH METHOD

We have applied the systematic mapping study to GUI testing of Android application area. We have adapted the steps defined in [13]. These steps describe the process of SM study in software engineering areas. As a result our research method is discussed as following:

- Overview
- Goal and research questions (RQs)
- Article sources and search keywords
- Inclusion\Exclusion criteria of Article set
- Final set of articles and the online repository
- Classification of the articles
- Data Extraction

3.1. Overview

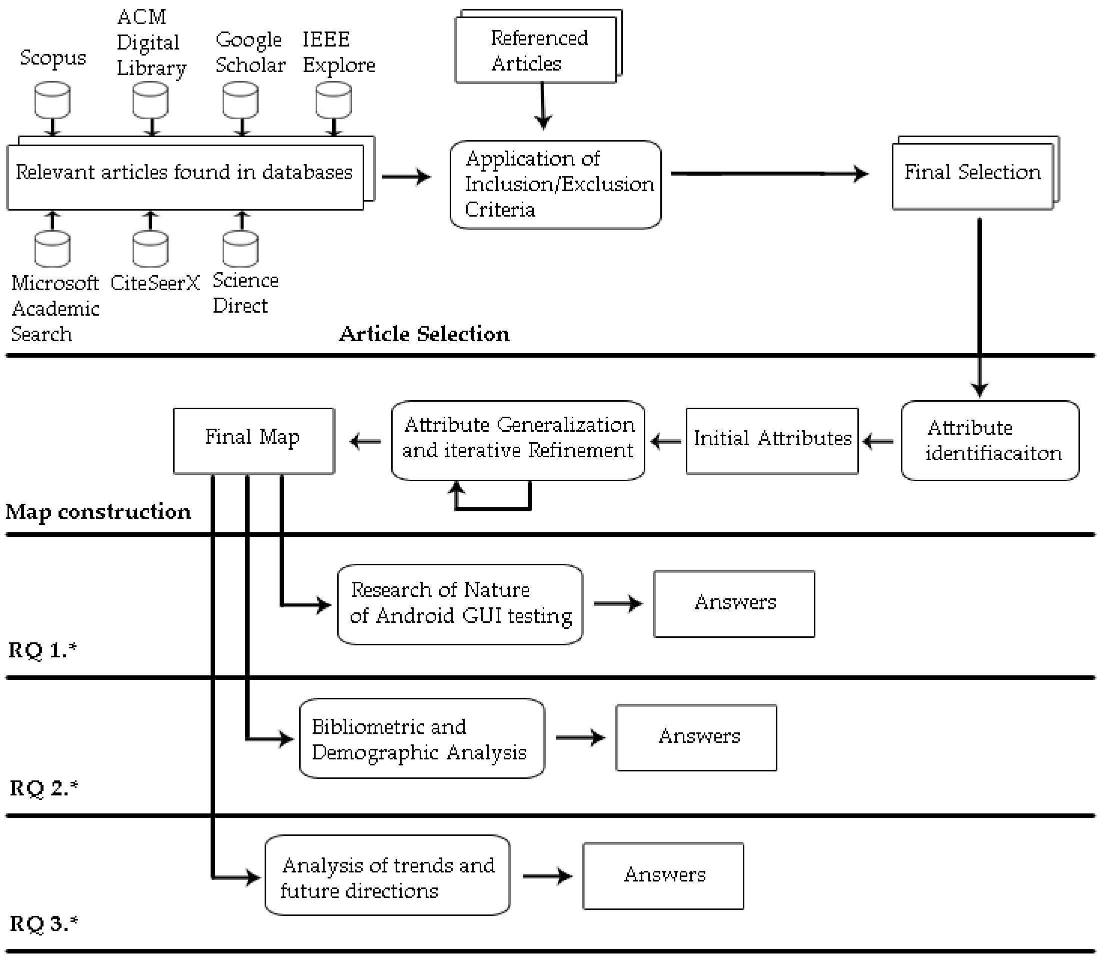
This SM is mainly carried out by following previous examples [15 - 16, 20] and the general study which focuses on using mapping studies in Software engineering proposed by Budgen et al. [13]. Although there is no similar instruction set as Kitchenham and Charters [11], which proposes a comprehensive guideline in order to SLR researchers, for SM studies, this guideline is also useful for us. Our method is a blend of these sources.

Our method can be summarized as follows:

- Defining the need for the study
- Determining the goal and research questions
- Searching for the primary studies that may be relevant to the subject.
- Applying the inclusion/exclusion criteria to the primary studies.
- Defining the attributes of the classification scheme
- Data extraction from primary studies in our scope.

- Performing a peer review on the extracted data in order to prevent bias.
- Classification of the primary studies in the final pool.

We have firstly determined the subject of the study. The subject "GUI testing of Android Applications" was new and unexplored area for a secondary study. The result would be very functional for followers. Then we have specified the goal and the RQs towards this goal. We have discussed it in section 3.2. Article selection and the elimination of the selected articles to create result article pool is another vital step. In Sections 3.2, 3.3 and 3.4, we have discussed about whole process and the result article set. In Section 3.6 we have defined the map construction process. The attributes of classification scheme are also described in subsections. Finally, through this scheme, we have extracted data from the article set in order to address our RQs. Whole process is abstracted in Figure 3.



Scopus
ACM Digital Library
Google Scholar
IEEE Explore
Referenced Articles
Relevant articles found in databases
Application of Inclusion/Exclusion Criteria
Final Selection
Microsoft Academic Search
CiteSeerX
Science Direct
Article Selection
Final Map
Attribute Generalization and iterative Refinement
Initial Attributes
Attribute identifiacaiton
Map construction
Research of Nature of Android GUI testing
Answers
RQ 1.*
Bibliometric and Demographic Analysis
Answers
RQ 2.*
Analysis of trends and future directions
Answers
RQ 3.*

Figure 3 - Protocol guide of the SM

3.2. Goal and Research Questions

According to Kitchenham and Charters [11] an SM study aims to classify and analyze the literature on software engineering domain. SM studies are also baseline studies for SLRs. These studies answer common questions about such as types of techniques that are commonly used, active researchers, trendy tools or density of studies over years.

We have used the paradigm that called Goal-Question-Metric (GQM) [26] in order to define our goal and research questions. We have mainly aimed to classify and analyze the literature on the subject "GUI testing of Android applications" starting from 2009 when first stable version of Android (1.5 - Cupcake) was released [8]. Based on our goal, we have defined three main RQ sets which cover all aspects of our research. Each of these questions is further partitioned into some sub-questions in order to advance details of the study, as described below:

- RQ 1-What are the natures of Android GUI testing?

  In Android GUI testing area, there are many components concerned by beginners or researchers who want to have more knowledge about the area. We aimed to aggregate all of the components in these questions. For more detailed result, we asked following sub-question:

  - RQ 1.1- Which are the testing activities applied in the primary studies?
  - RQ 1.2- What are the sources of information used to derive test artifacts?
  - RQ 1.3- Which are the test artifacts generated during testing process?
  - RQ 1.4- Which are the testing environments used to run tests? (Emulator, Real device, or Both)
  - RQ 1.5- How to simulate user interactions?
  - RQ 1.6- How to verify GUI behaviors (Oracles)?
  - RQ 1.7- What are the types of methods used to evaluation?
  - RQ 1.8- What are the attributes of the system under test (SUT)?

- RQ 2-What are the demographic and bibliometric aspects of the primary studies in Android GUI testing?

  The demographic and bibliometric aspects of the area can help to reveal an idea about tendency of researchers, institutions or countries on the subject. We expand our question as following:

  - RQ 2.1-What is the articles count per year?
  - RQ 2.2-What are the most popular articles? Which are the mostly considerable the venues and the authors are in terms of the article counts?
  - RQ 2.3-What is the article distribution over countries?

- RQ 3-What are the trends and future direction in GUI testing of Android applications?

  This RQ aims to expose current trends of researchers. In the scope of this question we also ask the tendency about future plans of researchers. The sub-questions are listed below:

  - RQ 3.1-What are the types of articles published?
  - RQ 3.2-What are the contributions provided by researchers?
  - RQ 3.3-Which are the most significant testing tools? Which are 3rd party components used by the tools?
  - RQ 3.4-What are the testing techniques/approaches used during test process?
  - RQ 3.5-What are the future directions of current researches?

The RQs are generally adapted from similar studies [12, 14-16, 20, 24] with accordance of our area GUI testing of Android applications. Each set of question examines a particular research area with its details by the sub-questions.

3.3. Article Sources and Search Keywords

We have aimed to collect all of the articles such as research papers, thesis, book chapters and technical reports related to our subject. In order to do that, we have decided to use seven major online academic article search engines listed below:

1. IEEE Xplore[2]
2. ACM Digital Library[3]
3. Google Scholar[4]
4. Microsoft Academic Search[5]
5. CiteSeerX[6]
6. Science Direct[7]
7. Scopus[8]

These search engines are capable to cover the literature with strong infrastructure for searching the sources. They are well-accepted in community and used for similar studies often, e.g., [12, 14-16, 20, 24].

Search keyword selection was also an important task in order to cover an area. With respect to this, we have used a search string consisting of three main parts. The first part

[2] *http://ieeexplore.ieee.org*
[3] *http://dl.acm.org*
[4] *http://scholar.google.com*
[5] *http://academic.research.microsoft.com*
[6] *http://citeseerx.ist.psu.edu*
[7] *http://www.sciencedirect.com*
[8] *http://www.scopus.com*

was related to platform definition. We have decided to select: (1) Android, and (2) Mobile. The second part have defined our testing domain: (3) Graphical User Interface, (4) GUI, (5) User Interface and (6) UI. The last part of our search string was about activity that is applied. Our selection for this part was: (7) Dynamic Analysis, (8) Model, (9) Ripping, (10) Static Analysis, (11) Testing and (12) Verification. In order to constitute the search string, we have put the Boolean operators between these words as conjunctions. We have used Boolean “AND” operator to combine three main parts of search string and Boolean “OR” operator to join the terms inside the parts. Final search string is shown in Figure 4.

("Android" OR "Mobile") AND ("GUI" OR "Graphical User Interface" OR "UI" OR "User Interface") AND ("Dynamic Analysis" OR "Model" OR "Ripping" OR "Static Analysis" OR "Testing" OR "Verification")

Figure 4- Search String

Our search was mainly based on Scopus. We have applied our search string to Scopus with additional choices which are "articles newer than 2008 (after first release date of Android [8])" and "articles related to Computer Science". The search engine has retrieved 798 results. We have examined titles and abstracts of these articles.

When we have applied the search string to the complementary search engines, we have discovered that some of them such as Google Scholar or CiteSeerX retrieve too many results. However most of these results are irrelevant to the topic. In addition, because we have listed the article by relevance, when we have get down to the articles retrieved by the search engines, remarkable article count was getting lower. Therefore we have decided to put a limit to the considerable maximum result number for such search engines. We have applied search string and we have determined a limit when we have experienced that their relevance was becoming ignorable. In Table 3, we have presented the limits for each search engine.

Table 3- Maximum considered result limits for search engines

Search Engine	**Limit of maximum considered result count**
IEEE Xplore	All results
ACM Digital Library	1 - 100 results
Google Scholar	1 – 200 results

Microsoft Academic Search	All results
CiteSeerX	1 - 200 results
Science Direct	All results
Scopus	All results

After the search was completed, we have eliminated the duplications and got the article pool that contains primary studies uniquely. In addition, we have examined references of the articles in order to find more relevant articles and provide full coverage of the domain. At the end, we have constructed a set of articles with 63studies. Before applying our inclusion/exclusion criteria, we have added them to our online repository. However we could not reach full texts of 4 articles. In the end, our pool consisted of 59 articles.

3.4. Inclusion/Exclusion Criteria

The inclusion/exclusion criteria are the conditions that have to be fulfilled by the article in order to improve reliability of our study. These criteria are determined by all participants of our study. In the paper selection process, the participants have systematically voted each of the articles in order to apply these criteria for deciding whether to include or exclude the article to final pool of articles. Each researcher has voted independently for each article in the pool. Such voting was performed to prevent the personal bias of the researchers.

We have asked two questions as our inclusion/exclusion criteria: (1) is the article relevant to the subject GUI testing of Android applications? (2) Does the article have a sound experiment or validation? These criteria were applied to all papers, including those presenting techniques, tools, or case studies/experiments. Voting mechanism was based on counting score of the articles. These questions have been answered by the researchers independently and individually. If the researcher have decided that the study addresses the criterion then he/she gave 1 point, otherwise 0 point. When all researchers have finished their voting, if article has more than three points, it was included otherwise we have left the article out of our scope. We have not encountered that two or more papers that describe same study (e.g. both a journal paper and a conference paper describes same study).

To facilitate the voting mechanism, all participants need to have a connection to the articles simultaneously. The best solution for this kind of cases is loading all the data on the web. There are several tools which meet this requirement. We have chosen the Google Drive System to work on. First we have uploaded all of our articles on to this drive. Then we have created a spread sheet document as our repository and put the references of the articles to there. Finally our repository was ready to vote the article to include or exclude for our scope.

3.5. Final Set of Articles and the Online Repository

When all participants have finished their voting, 32 of 59 articles were excluded from the final pool. Therefore our final list of articles has had 27 articles.

We have decided that publishing the repository in a public URL is useful to expose our work. It could be a continuous study and be easily followed by researchers using the web. Thus we have presented our online repository in a public URL [22].

3.6. Classification Scheme: The systematic map

After the final set of articles was determined, we have identified the initial attributes of our classification scheme by using the abstracts of primary studies. Then, we have determined the context of the research. Then we have carried out more comprehensive research on primary studies to recognize their contributions and nature. This research helped us to refine the attributes. The final attributes set is described in following sections.

3.6.1. *Type of Contribution*

We have adapted the classification of contributions by Petersen et al [37], which are method/technique, tool, model, metric and process. While adapting these facets to testing context, we added "other" category to map the articles that cannot be categories under these six groups. Finally, the type of contribution aspect has six categories:**(1) Test Method/Technique**, **(2) Test Tool**, **(3) Test Model**, **(4) Metric**, **(5) Process** and **(6) Other**. An article can fall in one or more categories in this aspect, e.g. an article may both present a test tool and a new test metric.

3.6.2. *Type of Research Article*

We have categorized research articles by using an existing schema designed by Wieringa et al [30]. This schema was originally created for requirement engineering paper classification but it can be easily adopted for our domain. It is also commonly used in similar studies. The classification can be summarized as follows:

1. **Solution Proposal:** These studies propose a novel solution to a problem or present a significant extension to an existing solution. The problem is defined clearly. The solution technique is evaluated by a small example or with a good line of argument
2. **Validation Research:** These studies present preliminary empirical evidence for a proposed solution. This kind of researches may include methods like experiments, simulations, prototyping, and mathematical investigation/proof.
3. **Evaluation Research:** These studies use formal experimental methods to evaluate novel techniques and tools implemented in practice. If the usage of the technique is reported, novelty is not a required criterion but soundness is. The evaluation of the technique/tool is more rigorous then a validation research.

4. **Experience Papers:** These papers are based on authors' personnel experience about usage of tools, application of techniques or other activities related to the domain. It is aimed to represent lesson learned from experiences.

In the original paper [30] which is used for this categorization there were two more categories for the papers: philosophical papers and opinion papers. However the papers in these categories were eliminated because of our inclusion/exclusion criteria.

During the mapping, an article can be placed only one of these categories.

3.6.3. *Type of Testing Activity*

To collect data about type of testing activities in this domain, we have used the categorization defined by J. Offutt et al. [31] who broke up testing into four general types of activities. These types were test design, test automation, test execution and test evaluation. Test design could be divided further into two types (1) Criteria Based, (2) Human knowledge based.

The categories under this aspect are summarized as follows:

1. **Test case design (Criteria Based):** Designing test cases to satisfy several testing criteria (such as coverage). This design can be automated or carried out manually.
2. **Test case design (Human Knowledge Based):** Designing test cases based on personnel knowledge of tester about testing and usage of SUT. This design can be automated or carried out manually.
3. **Test automation:** Creating executable test codes where test values are embedded. These test codes (scripts) are capable to run automatically.
4. **Test execution:** Executing tests on the SUT and gathering the results as an output. Test execution can be automated or carried out manually.
5. **Test evaluation:** Evaluating the test outcome in order to inform the developers.
6. **Other:** If the article cannot be placed into above categories.

3.6.4. *Source of Information to Derive Test Artifacts*

Another part of repository is source of information to derive test artifacts such as test cases. Finally we have decided on 7 parts as follows:

1. **Source code (white-box):** Making detailed analysis of internal code structure and logic to derive test artifacts. The source code is needed to be available. The tester needs to possess knowledge of the internal working of the source code.
2. **Requirements (Black-Box):** The test artifacts are derived without any information of the internal logic of an application. There is no access to the source code. Only the user interface is presented to make interactions by giving inputs and examining outputs.
3. **Requirements and Source code (gray-box):** Deriving test artifacts with partial information of the logic of internal mechanism of an application. Unlike black-

box approach where only user interface is presented the design documents or the database are available. This information may help to derive better and accurate test artifacts.

4. **Logs:** The logs that are automatically created while running the SUT are considered to generate test artifacts.
5. **Inferred Model (automatic):** Model of the SUT that is generated automatically is used to derive test artifacts.
6. **Inferred Model (manual):** Model of the SUT that is generated manually is used to derive test artifacts.
7. **Other**: if none of the above is applicable.

3.6.5. *Approach of GUI Behavior Verification*

Because our main goal is GUI testing, we have decided to make a section about GUI behavior verification (oracle) approaches. In this section we have classified mechanisms to check the GUI outputs, states etc for correctness after the test execution. We have observed that there are five main approaches described as follows:

1. **Bitmap Comparison:** This technique is used by the mechanism which is able reach a state of a visual object using screen-shots, images of widgets (buttons, text views etc.), and compare it with expected state of the object in order to verify the execution.
2. **Model-Based:** Models such as Finite State Machines (FSM) are able to keep states of user interfaces. These states consist of values of several GUI objects which can be used for test oracles.
3. **Checking Widgets via API:** Each programming language that supports visual objects has an API for management operations of widgets to change their values or states. Same API can used to verify expected value of these objects.
4. **Manual:** Manually comparing of states or values of GUI objects with the expected ones.
5. **Optical character recognition (OCR):** This technique is based on conversion of images of alphanumeric characters into machine-encoded text. It is usually applied on an image captured from user interface to get its values which are easily confirmable.
6. **Other**
7. **No Oracle:** Some articles are enablers of GUI testing and therefore they do not express their oracles.

3.6.6. *Approach of Simulating User Interaction*

Simulation of user interactions is a common approach in GUI testing especially. Therefore we have decided to collect information about how testers simulate these interactions on the user interface. We have detected two common approaches; coordinate based or via capturing the widget itself. Some of studies have used them both.

These categories are summarized as follows;

1. **Coordinate based:** In this approach, user actions are simulated on the screen according to a coordinates pair (x, y) that defines a single point on the interface. In order to support all the user actions (such as long press, drag-drop) in Android GUI, this technique may be represented in more comprehensive forms.
2. **Capturing Widget Object:** Information of a widget on the GUI is gathered in order to perform some user events on it. These events can be executed using widget own actions (via an API) or with external events (such as mouse or keyboard events).
3. **Both (Combined)**
4. **N/A:** If the user interactions are not simulated on the GUI objects in the testing process, this case will be in this category.

3.6.7. *Techniques Used*

In GUI testing domain, there are many well-accepted techniques which are utilized for various purposes. Tester may possibly use one or more of these techniques during whole testing process. Therefore we have decided to search for following techniques in studies analyzed.

1. **Symbolic execution**: In this technique, firstly the SUT is examined to detect inputs that execute each of the features of the SUT. Then symbolic values are used instead of actual data as input values, and software variables are represented as symbolic expressions. Thereby, the result of computation is demonstrated as a function of the symbolic inputs.
2. **Static program analysis**: This technique is applied without an executing the application. The analysis is generally realized on the source code of the SUT but for some cases it can be applied on object code the product of compilation.
3. **Concolic testing (Dynamic Symbolic Execution)**: This technique is a combination of two testing approaches; symbolic execution, and concrete execution [32].The concrete execution determines the particular inputs to be used through symbolic execution technique in order to generate test cases. This technique is mainly used to find bugs on the applications rather than verifying the application's correctness.
4. **GUI ripping:** GUI ripping is an approach that automatically traverses the GUI of the SUT by reaching each state of it and collecting the values of its elements [38]. Then this collected data is used to generate test cases.
5. **Model checking:** The software system is modeled as finite-state machine which controls states of the system to detect the undesirable states that can cause critical errors such as deadlocks or crashes. It is an automatic technique that verifies validity of system properties.
6. **Model-based testing:** In this approach, the model which extracted from the SUT is utilized to demonstrate the intended activities of the system or to demonstrate testing approaches. The model is generally used to extract test artifacts such as test cases automatically.
7. **Dynamic analysis**: Dynamic analysis is a technique that tests the SUT while system variables are dynamically changing in time. In this approach the source

code of the software is compiled and ready to run and test artifact may need to reach to the source code itself to embed values.

8. **Random testing:** This technique is based on random and distinct input generation to the SUT. Reactions and outputs are collected and compared with expected ones in order to verify the system. If the software consists of faults, a mismatch will be detected while the comparison or the inputs will cause a crash of the system. It is a black-box approach.
9. **Record-replay analysis:** This is a debugging technique that is based on repeating interactions that are recorded from the first usage of the application on the user interface of the SUT. During the execution given inputs, passed states and the outputs are listed as logs to define the causes of defects. It is also used for automating test executions.
10. **Other**

3.6.8. *Testing Environment*

Toward our research question (RQ 1.7), we have decided to get information about testing environment where the tests are run on. There are two main options to execute the test on: **(1) Real device** which refers to an android device (phone or tablet) and **(2) Emulator** that simulates Dalvik Virtual Machine (DVM) on popular computer operating systems such as Windows, Linux etc. We have decided that there maybe two other options **(3) Both** which stands for the cases that both real device and emulator is used and **(4) Not Reported** for the articles that does not specify the testing environment

3.6.9. *Type of Test Artifact Generated*

During the testing process various test artifact are could be generated. On the purpose of classifying these artifacts, we gathered these values from studies in our scope. The artifacts we have searched for in the articles are:

1. **Test Inputs:** The data used to execute test cases. It is gathered from an external such as hardware, software or human.
2. **Test Requirements:** The procedure of test conditions that identify which features of the SUT is required to be validated. It doesn't specifies input values for test cases
3. **Test Oracles:** The mechanism to compare the collected outputs of the SUT with expected ones.
4. **Test Driver:** A tool that replaces a component of the SUT which has a role during system execution in order to perform test procedures. Test scripts are the most common examples.
5. **Other**

3.6.10. *Type of Evaluation Method*

We have questioned the methods which are used to evaluate the proposed approaches in the studies. The evaluation scope is an important value to have an opinion about success and quality of the approach or tool presented. We categorize the evaluation methods as following;

1. **Coverage:** A criteria about calculating the percentage of the some items of the SUT such as code blocks, braches, logic, or classes that is reached during the test execution.
2. **Mutation Testing (Fault Injection):** The process of creating faulty versions of the SUT. It is indented to determine whether the injected defects could be found by test tool or technique.
3. **Time/Performance:** A criteria gather by calculating the duration of test process.
4. **Detecting Real Defects:** A method to evaluate applications to detect their faults.
5. **Other**

3.6.11. ***Attributes of Testing Tool Presented***

Lots of tools that aim to run tests for GUIs of applications in Android platform are presented in several studies. We thought that gathering this information may be helpful for researcher, testers, or developers in many ways. Firstly we have decided get **(1) Number of tools** that proposed in the study. Then we get **(2) Name of the tool**. Next part is about **(3) Third party applications/frameworks/components** that are used as building blocks while developing the tool. This data may be very helpful for especially developers who intended to develop a testing tool. Fourth part is **(4) Supported programming language** that is used to create test code such as test scripts test cases etc. The last one is stands for whether the tool is **(5) Available to download**. We present its **(6) URL** if it is.

3.6.12. ***Attributes of SUT***

Each evaluation for a testing approach (technique, tool etc.) requires an SUT in order to apply the approach on it. In addition, attributes of SUTs may vary. We have chosen to collect information about the SUT that are used in the articles. We have got following information:

1. **Number of SUTs**
2. **Names of SUTs**
3. **Size:** Size of source code of SUT in terms of line of code (LOC).
4. **Number of Screens:** Number of user interfaces that is presented by the SUT. In Android applications this number is equal to the number of a special type of class “Activity” number.
5. **Front-end Approach:** Technology used to develop the user interface.
6. **Version of Android:** Android version where the evaluation is done.
7. **Development Style:** Real Open-Source, Academic Experimental, or Commercial

3.6.13. ***Future Plan***

We wonder the future directions of our domain. In order to accumulate information about this issue, we have decided reserve a part of our repository. We have collected under the six headings which are; **(1) Develop a new Tool**, **(2) Improve the Tool**, **(3) Develop a new technique**, **(4) Improve the technique**, **(5) Make research more detailed** (New case studies e.g.), **(6) No Future plan.**

Table 4 - The classification scheme

RQ	**ATTRIBUTE**	**VALUES**	**MULTIPLE SELECTION**	**SINGLE SELECTION**	**TEXT**
RQ_1.1	Type of Testing Activity	Test-case Design (Criteria-based), Test-case Design (Human knowledge-based), Test Automation, Test Execution, Test Evaluation (oracle), Other	x		
RQ_1.2	Source of information to derive Test artifacts	Source Code, Requirements and Source code (gray-box), Requirements (Black-Box), logs, Inferred Model (automatic), Inferred Model (manual), Other	x		
	Type of test artifact generated	Test input (data), Test requirements (not input values), Expected outputs (oracle), Test driver (code), Other	x		
RQ_1.3	Test On	Emulator, Real Device, Both, Not Reported		x	
	Android Version				x
RQ_1.4	How to Simulate User Interactions	Coordinate based, Programmatic, Both (Combined),Not Reported		x	
	Approach of GUI Verification (Oracle)	Bitmap Comparison, Model (Finite State Machine, etc), Checking widgets via API, Manual, OCR (Optical Character Recognition), computer vision, Other, No Oracle	x		
RQ_1.5	Type of the Evaluation Method	Coverage (code, model), Mutation testing (fault injection), Manual comparison (with another result), Time/Performance, Detecting	x		

		real faults, Other			
RQ_1.6	Attributes of the Android Application SUT(s)	Number of SUTs, SUT names, Size in LOC, Number of Screens (Activities), Front-end approach			x
		Real Open-Source, Academic Experimental, Commercial		x	
RQ_2.(1-3)	Demographic/ bibliometric information of Paper	(Authors, Author Country, More than one country, Venue, Year, Article Type, Citations)			x
RQ_3.1	Type of Paper-Research Facet	Solution Proposal, Validation Research, Evaluation Research, Experience Papers		x	
RQ_3.2	Type of Paper-Contribution Facet	Test method / technique, Test tool, Test model, Metric, Process, Other	x		
RQ_3.3	Attributes of the Testing Tool Presented in the Paper	Name, 3rd Party Components (Frameworks, Programs), Supported Languages, Available for Download, URL			x
RQ_3.4	Technique Used	Symbolic execution, Static code analysis, Concolic testing (Dynamic Symbolic Execution), GUI Ripping, Model checking, Search-based testing, Model-based testing, Dynamic Analysis, Crawling, Random Testing, Record/Replay, Other	x		
RQ_3.5	Future Plan	Develop a new Tool, Improve the Tool, Develop a new technique, Improve the technique, Make research more detailed (New case studies e.g.), No Future plan	x		

3.7 Data Extraction

After we determined the classification scheme, we have extracted data from primary studies according to this scheme. The data extraction is a task that has to be objective. Personnel bias can decrease the quality of the results deduced from this study. In order to prevent this, we have used peer review technique. At least one researcher has verified the validity of the each data extracted by the author. This process went on the web using Google Drive System. This approach avoids the study from subjectivity.

At the end of the process, the data was collected on the repository. Before inferring final results from this data, we have decided to present this raw data on web to the public URL. Our online repository can be reached using the URL in [22].

CHAPTER 4

RESULTS

In this chapter we have presented the results we gathered from the systematic mapping of the articles in our pool. We have addressed the RQs using this information.

4.1. What are the natures of Android GUI testing? (RQ 1)

This research question is investigated under six sub questions.

4.1.1. Which are the testing activities applied in the primary studies? (RQ 1.1)

To provide an answer to this RQ, we collect the data of types of testing activities. These activities are adapted from J.Offutt's descriptions in his lecture notes [31]. Each of these activities can be executed individually or they can be executed sequentially. The activities can also be considered as complete testing life cycle when all of them applied.

The categories used in this facet are described in Section 3.6.2.Figure 5 shows the distribution of the papers to the six categories, the "other" category is used for catching additional activities to collect the data. We observe that 92% of researchers have made test case design (66,6% criteria-based, 26% human knowledge-based), 74% of researchers have made test automation, 92% of researchers have made test execution, 7% % of researchers have made other testing activities.

We have also another graph (Figure 6) in order to examine whether these testing activities done automatically or manually. For each category, we have presented the primary studies in Table 5. According to this data, we have observed that in all the papers, researchers have done test case design activity except [S21] and [S23]. In [21], researchers have done a static analysis without defining a test case. In [23], Garousi et al have measured the efficiency of Android' test suite which is publicly available. Thus they used predefined test cases to work on.

According to Figure 6, all of the test cases created based on a criteria, were generated automatically by the test tools and remaining test cased which were created based on human knowledge are created manually by the testers. The only difference is in [S8] that, Amafitano et al. have proposed an approach propped up the reusable event patterns to generate test cases either manually or automatically.

The test automation activity has done in 20 studies. [S21] and [S23] were again excluded from this category together with [S1, S4, S12, S14, S17]. This activity was not referred in these studies.

For test execution category, the results were very similar to test case design. This activity was used in 25 of papers in total. Only in [S1] and [S21], researchers have only defined their technique theoretically without presenting the execution process. The test execution process was done automatically for all of the studies. Only two of them [S9, S17] also supported manual execution.

Test oracle was also a commonly used testing activity. It was used in 19 studies. 15 of them used this testing activity automatically and 3 of them manually. In only one study [S16], it was used both manually and automatically. MobiGuitar, the tool proposed in [S16], while automatically verifying features of the SUT with assertion mechanism of JUnit[39], it also produces a crash report which can be analyzed manually to determine the faults.

Other than these testing activities, C. Hu et al, [S7] have made a bug categorization study for Android OS. In addition, Rountev at al [S21] have made a novel static analysis study on GUI code by modeling their flows, associations and relations.

Table 5- Testing Activities in Primary Studies

Type of Testing Activity		**References of Articles**
Test Case Design	Criteria-Based	[S1], [S2], [S3], [S4], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S14], [S16], [S17], [S18], [S22], [S25], [S27]
	Human Knowledge-Based	[S12], [S13], [S15], [S19], [S20], [S24], [S26],
Test Automation		[S2], [S3], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S13], [S15], [S16], [S18], [S19], [S20], [S22], [S24], [S25], [S26], [S27]
Test Execution		[S2], [S3], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S22], [S23], [S24], [S25], [S26], [S27]

Test Oracle	[S2], [S3], [S7], [S9], [S10], [S11], [S12], [S13], [S15], [S16], [S17], [S18], [S19], [S22], [S23], [S24], [S25], [S26], [S27]
Other	[S7], [S21]

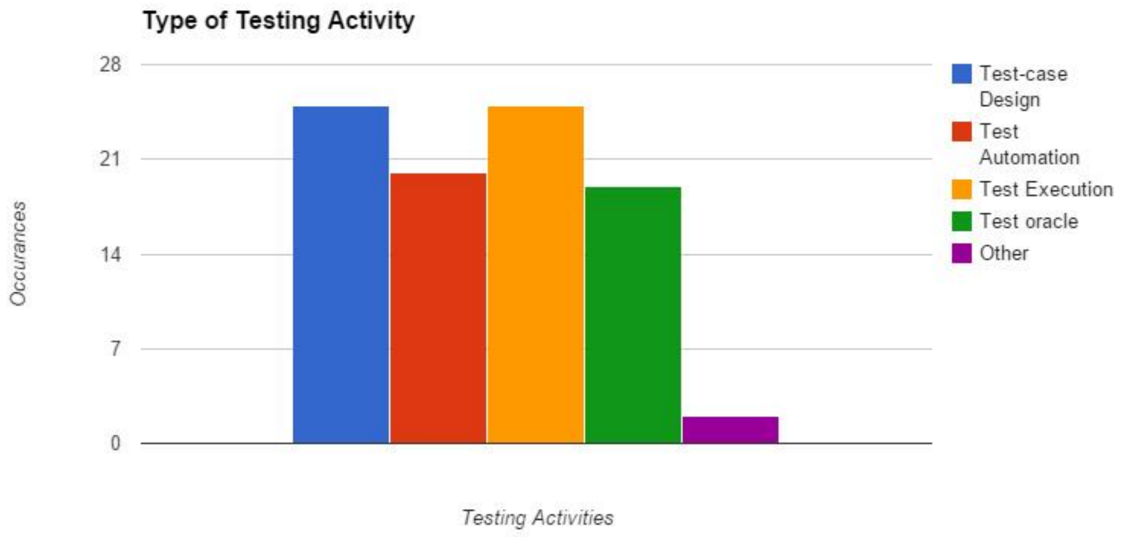


Figure 5- Type Testing Activity

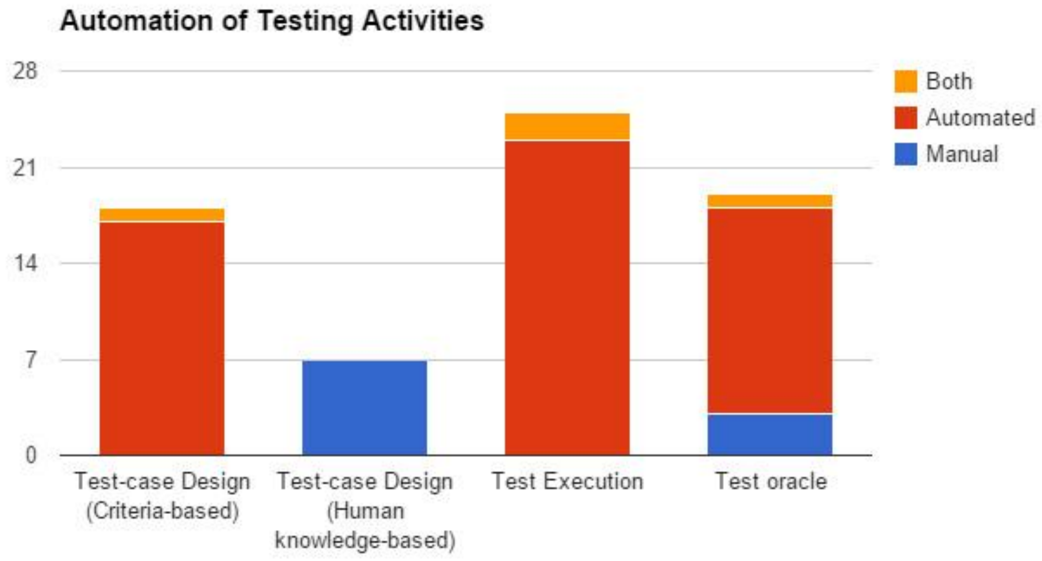


Figure 6- Automation of Testing Activities

4.1.2. What are the sources of information used to derive test artifacts? (RQ 1.2)

The sources of information that the test artifacts are derived from can vary from one technique to another. As described in Section 3.6.4, the categories of sources used to derive test artifacts are source code a.k.a white-box approach, test requirements and source code a.k.a gray-box approach, only test requirements a.k.a black-box approach, logs, inferred model (automatically or manually).

Recall that, the approach presented in an article can use more than one source to derive the test artifacts. Figure 7 shows the distribution of the articles in our pool according to source of information. According to these acquisitions, the most popular sources are the models (62%). In 55% of cases the models were extracted automatically. Only in two of the studies [S17, S18], the models were extracted manually. In 15 of 27 studies, more than one resource was used. In six cases, source code and extracted model were used together. There was only one study [S7] performs a log file analyses to detect potential bugs. This study has also used source code and the model extracted automatically to generate test cases. In [S10], Hu et al have presented a testing tool called AppDoctor which produces bug reports that helps developers to diagnose the results to reduce false positive bugs detected. We have evaluated this as "other" source of information to derive test artifacts.

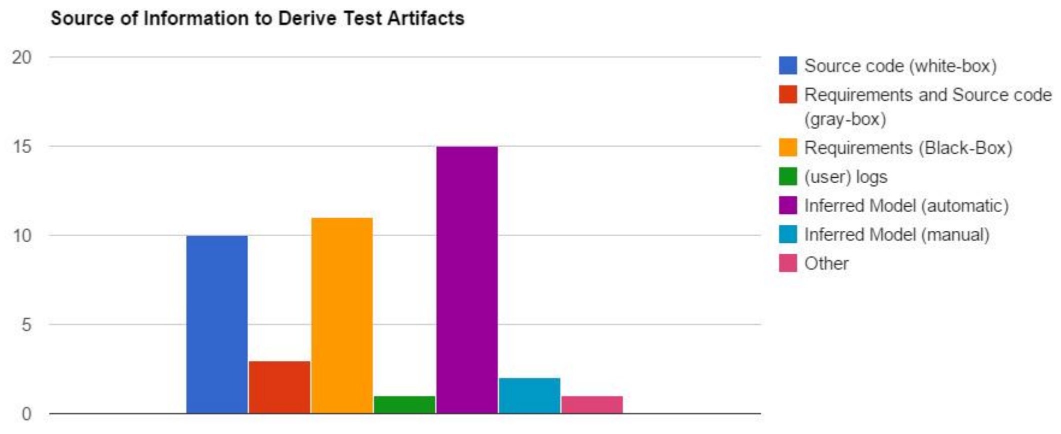


Figure 7- Source of Information to Derive Test Artifacts

4.1.3. Which are the test artifacts generated during testing process? (RQ 1.3)

The term test artifact is usually used to refer something produced during test process. As described in Section 3.6.9, types of test artifacts are test inputs, test requirements, test oracle, and test driver.

In this RQ, we have investigated the artifacts generated during test process described in the primary studies. This information gives an idea about working principle of the proposed techniques or tools.

In Figure 8, we have illustrated the generated test artifacts using the sources we discussed above. According to this information, almost every study (24 of 27 studies) produces a test input. The 2 of 3 studies remaining produces test requirements which do not consist of input values. The only study which does not produce both of them is [S26]. In this study, Sadeh et al make a discussion about existing testing approaches of the GUI code of Android applications. Finally they give a recommendation about the approaches.

In 14 studies, the test oracles were generated from the sources. Recall from Section 4.1, the test oracle activity was performed in 19 studies. There were five more studies where test oracle was used but not produced by the proposed approach. When examining these studies, in three of them test oracles were applied manually and in remaining two studies, test oracle was performed by using assertion mechanism which is based on testers' decisions.

Test drivers are more common than test oracles as generated test artifacts. In 20 studies these drivers were produced. The most popular framework is JUnit to write test codes. It have been used in five studies (see also 4.12).

There were no test artifacts generated other than these categories in the primary studies.

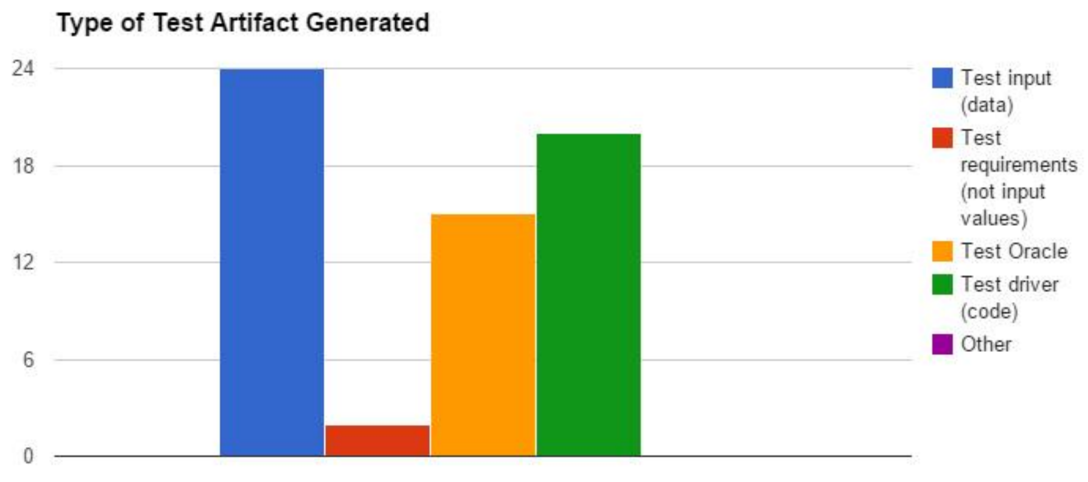


Figure 8- Type of Test Artifact Generated

4.1.4. Which are the testing environments used to run tests? (RQ 1.4)

Android is an OS which is able to run on mobile devices such as smartphones or tables. However these devices are not suitable to develop an application. Because the developing process generally goes on desktop-like systems, another system that runs the Android is required. While real devices are being used in some cases, emulators which are virtual machines that run Android as their OS can also be used for other cases. Due to the fact that development and testing processes are usually goes on parallel, these development environments are also used for testing too.

Because our research has focused on Android OS, we have collected the data about testing environment. In Figure 9, we have presented the percentages of testing environments that are preferred. As explained in Section 3.6.8, there were two choices to realize test on, an emulator or a real device. Some the researchers [S6, S10, S14, S17] declared that the proposed technique or tool supports both of these choices. We classify them as "Both". Moreover, some the researchers [S1, S2, S8, S12, S23, S25] does not report what they use as an Android system. Thus we classify them as "Not Reported".

In Figure 10, we have presented a graph to illustrate distribution of Android versions over the testing environment. Because the compatibility issue of techniques or tool, this information could be very useful. We have pointed the version numbers in the graph and we used "Not Reported" if there is no information about the Android version. The colors point the testing environment which is similar with Figure 10. For example, the only study which worked on Android version 1.5, have used an emulator as the testing environment. In another case there was a study which worked on Android version 2.2 but its testing environment was not specified whether a real device or an emulator. The “not reported” column also illustrates the studies where android version was not specified, e.g. if the color is blue, the study uses an emulator without specifying Android version or the color is green both Android version and testing environments are not specified.

When analyzing both of Figure 9 and Figure 10, it can be easily inferred that the most preferred environment to run tests is the emulators. The emulators were exclusively used in almost half of the studies (in 13 studies). If we included the studies which have used both of the systems, this number would be 17.

The second information that can be inferred from these figures is commonness of not reported information. Generally the Android versions of testing environment were not specified. In 27 studies, there were only 10 ten studies which have stated the Android version.

Actually for most cases, emulator can replace real devices or vice versa as a testing environment. However there was an interesting study [S24] that has a particular testing setup. In this study, Kaasila et al have established a cluster of smartphones connected to the internet. This system works as cloud testing center. The user of system uploads his/her SUT to the system and gets the results from his account of system's web page. This study was the only multi version system pointed in Figure 10.

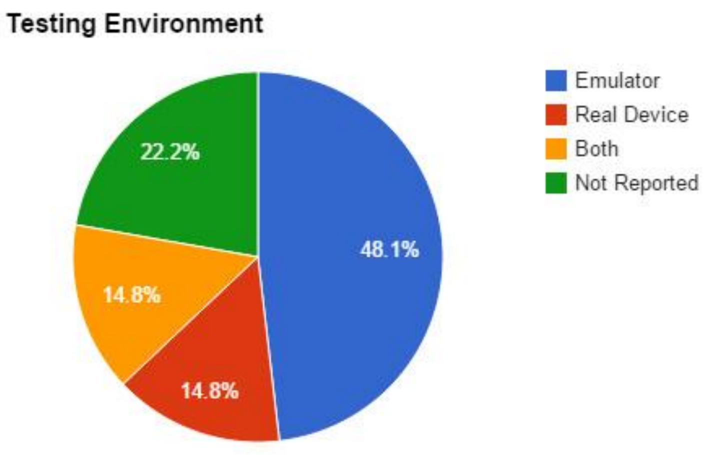


Figure 9 - Testing Environment

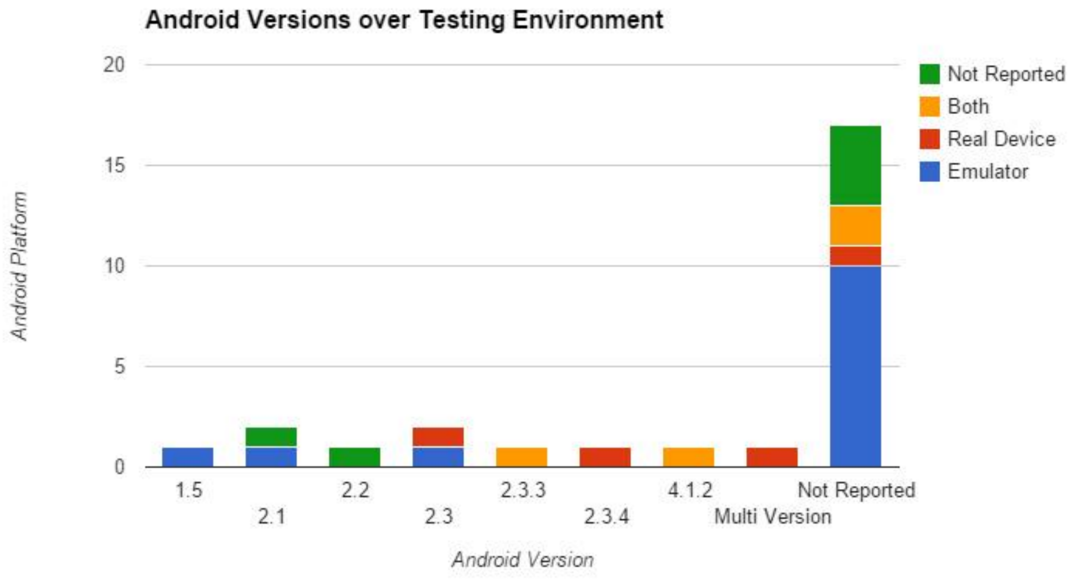


Figure 10- Android Versions over Testing Environment

4.1.5. How to simulate user interactions? (RQ 1.5)

In GUI testing, the main purpose of the test is covering all functions presented by GUI and verifying them all. GUI is used for users to interact with the program, so a tester should simulate the user behavior to activate the GUI features. The classification scheme was given in Section 3.6.6 in detail. According to our classification user actions can be simulated in two ways; coordinate based approach, and by capturing the widget object. In some of the articles we have discussed below, both of these approaches were combined while simulating the user interactions. We have used this classification in Figure 11. We have also listed the articles in Table 6.

Table 6- Approach of Simulating User interaction over Primary Studies

Approach of Simulating User Interaction	References of Primary Studies
Coordinate Based	[S4], [S5], [S7], [S14], [S15], [S19], [S20]
Capturing widget Object	[S1], [S2], [S3], [S8], [S10], [S14], [S16], [S17], [S23], [S25], [S26], [S27]
Both (Combined)	[S6], [S9], [S11], [S13], [S18], [S22], [S24]
N/A	[S21]

In the studies that coordinate based approach was used (the studies which uses both the approaches are included), the technique called record/replay (see Section 3.6.7) was commonly used. This technique was used in six studies in total [S4, S13, S15, S20, S21, S23] (see Section 4.13).

The trendiest way to capture widget object to make interactions on is extracting the GUI model. All occurrences of the model extraction techniques such as crawling [S1, S2, S8], GUI ripping crawling [S3, S8, S16, S27] (see also Section 3.6.7) were made on the studies where user interactions were simulated by capturing the widget objects (see Section 4.13).

The only study that did not simulate user interactions was [S21]. In this study, Rountev et al, did not execute any test cases.

4.1.6. How to verify GUI behaviors (Oracles)? (RQ 1.6)

In GUI testing the user interaction are usually used as test input in order to evaluate the features represented on the GUI. These inputs are processed by the SUT and it produces an output. While these outputs can be a state change on the GUI, they also can be a value change on a widget or a representation of a calculation result. Most of these outputs can be stated as GUI changes. Thus, in order to validate the execution, GUI can be monitored to compare with expected condition. This verification is carried out with GUI oracles.

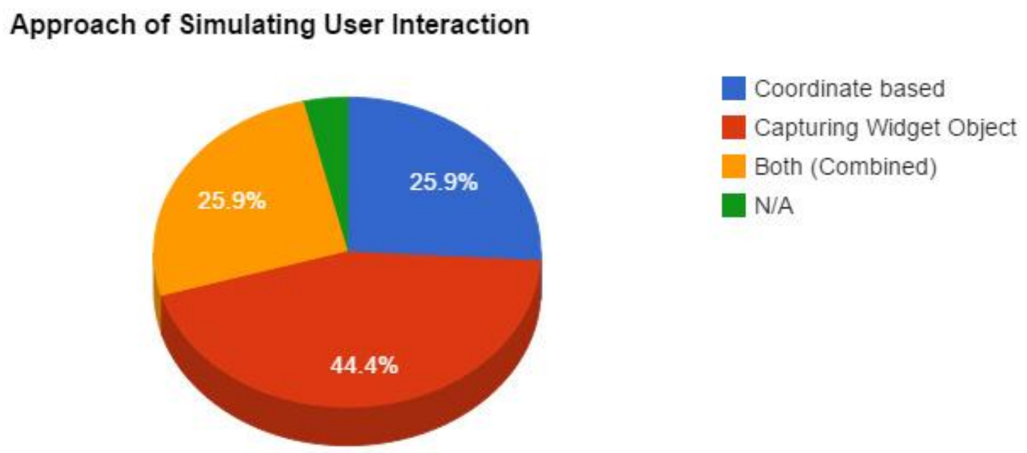


Figure 11- Approach of Simulation User Interaction

The GUI oracle data collected from primary studies has been represented in Figure 12. We have categorized oracle approach types in six categories. The classification scheme given in Section 3.6.5 consist of four major approaches: bitmap comparison, model, checking widgets via API, manual and optical character recognition (OCR) that is also known as computer vision, "other" for remaining approaches, and finally "no oracle". The oracle data has represented that 8 studies (30%) did not use an oracle to verify GUI behaviors. Remaining 19 studies were distributed almost uniformly over oracle approaches. We have demonstrated the matching of primary studies and oracle approach in Table 7

Table 7- Approach of GUI Behavior Verification over Primary Studies

Approach of GUI Behavior Verification (Oracle)	References of Primary Studies
Bitmap Comparison	[S15], [S19]
Model-Based	[S7], [S11], [S17]
Checking widgets via API	[S2], [S10], [S23], [S26]
Manual	[S9], [S16], [S22], [S24]
Optical Character Recognition (OCR) a.k.a Computer Vision	[S13], [S15], [S18]
Other	[S3], [S12], [S16], [S25], [S27]

The Bitmap comparison techniques are generally realized by taking a screenshot of actual state, to compare it with expected image of expected state. However in [S19], Lin et al have proposed a test tool called SPAG-C which takes a photo of the actual state with an external camera. This image was processed to verify the GUI of the SUT. In Model-Based verification, gathered model were used to verify GUI states [S7, S11] or GUI elements [S17]. The studies which checked value or states of widget by using Android API [S2, S10, S23, S26] have also simulated the user interaction by capturing widget object. The manual oracle was usually done by analyzing the outputs of testing process such as execution logs [S9, S24], crash reports [S16, S22], or screenshots [S24]. Using the OCR (a.k.a computer vision) technique to verify the GUI of the SUT is firstly defined in [S13] by Chang et al. They reach the values on the widgets by using this technique. Two more studies [S15, S18] have also used this technique. The other studies which were not use the above techniques makes their oracle by using JUnit's assertion mechanism [S3, S16, S25], by using graph construction algorithm [S12], or by detecting run time crashes [S27].

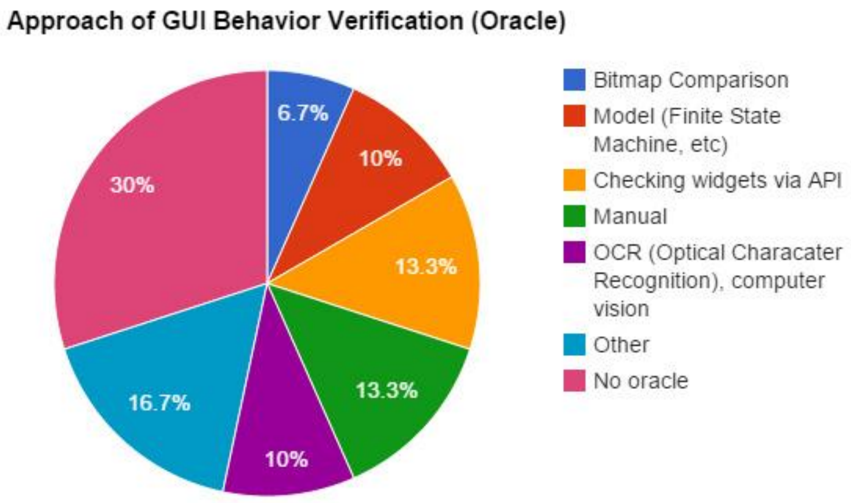


Figure 12- Approach of GUI Behavior Verification (Oracle)

4.1.7. What are the types of methods used to evaluation? (RQ 1.7)

Evaluation methods are used to verify the approach/tool presented in the articles in different ways. These methods measure capabilities of the approach/tool and present the results in term of exact metrics.

In Figure 13, we have shown the distribution of evaluation methods used in primary studies according to the classification scheme described in Section 3.6.10. The categories of the evaluation methods as coverage evaluations, mutation testing, comparing the techniques or the tools with other ones, time/performance evaluations, and the evaluations based on real fault detection capabilities. We have listed these categories and the primary studies in Table 8.

Table 8-Type of Evaluation Method over Primary Studies

Type of Evaluation Method	References of Primary Studies
Coverage	[S1], [S3], [S6], [S8], [S9], [S10], [S13], [S14], [S16], [S10], [S21], [S22], [S23], [S25], [S27]
Mutation Testing	[S2], [S23]
Manual Comparison	[S1], [S2], [S3], [S4], [S5], [S7], [S8], [S9], [S11], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S22], [S23], [S26], [S27]
Time/Performance	[S1], [S4], [S5], [S6], [S9], [S10], [S11], [S12], [S14], [S15], [S17], [S19], [S20], [S21], [S22], [S23], [S26]
Detecting Real Faults	[S2], [S3], [S4], [S7], [S8], [S9], [S10], [S11], [S12], [S16], [S18], [S20], [S23], [S24], [S27]
Other	[S12], [S13], [S26]

Although the mostly used evaluation method was manual comparison (74%) the usage of other techniques such as coverage (52%), time/performance (63%) and detecting real faults (48%) were also common. It could be inferred that these techniques are usually used with another one. All of these techniques were used more than half of the studies (coverage in 14 articles, manual comparison in 20 articles, time/performance in 17 articles, and detecting real faults in 13 articles) except mutation testing. Mutation testing was used in only 2 studies [S2], [S23].

In the “other” category, Zhang et al [12] have evaluated their technique by checking the warning produced by their tool manually. They measured the rate of the false positive with this evaluation. Chang et al [13] have also checked the test scripts if they still work in later versions. This evaluation was based on reusability. Sadeh et al [26] have made evaluations on three tools in order to compare them. They have used five evaluation criteria: ease of writing tests, ease of maintenance, error localization, relevance, and speed.

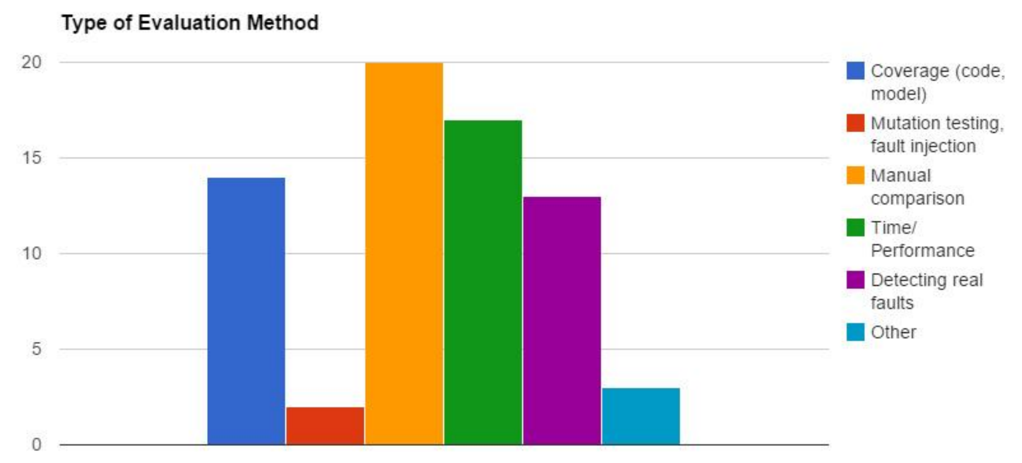


Figure 13- Type of Evaluation Method

4.1.8. What are the attributes of the systems under test (SUT)? (RQ 1.8)

The specifications of tested applications, call as SUTs, could be very important while determining whether the proposed approach is proper for another case. This information can also give a general idea about the focus areas of researches. Thus we have decided to collect the details of SUTs in order to represent this information to the community.

In our study, we have collected the following data about SUTs: how many SUT were used, names of SUTs, size of SUTs in terms of line of code (LOC), screen counts of SUTs, front-end approaches of SUTs, type of SUTs in terms of developing approaches such as real open-source, academic experiment, and commercial purposes applications which hide their implementations. After we applied these classifications, we have realized that most of this information was not reported in the articles. We have presented results in following figures (figure 14-18).

In 26 of the primary studies, SUTs were used to evaluation (In [S13] no SUT is used). In total, 334SUTs were tested (we excluded SUTs from [S24], because the author says that more than 20000 tests have been conducted since November 2011). In six of the studies [S2, S11, S17, S25, S26, S27], only one SUT was used. The average number of the SUTs is 17 SUTs per article for the articles where more than one SUT is used. The first argument about the data collected for attributes of SUTs is that there is not too much information reported in articles. The amount about reported information has been shown in Figure 14.

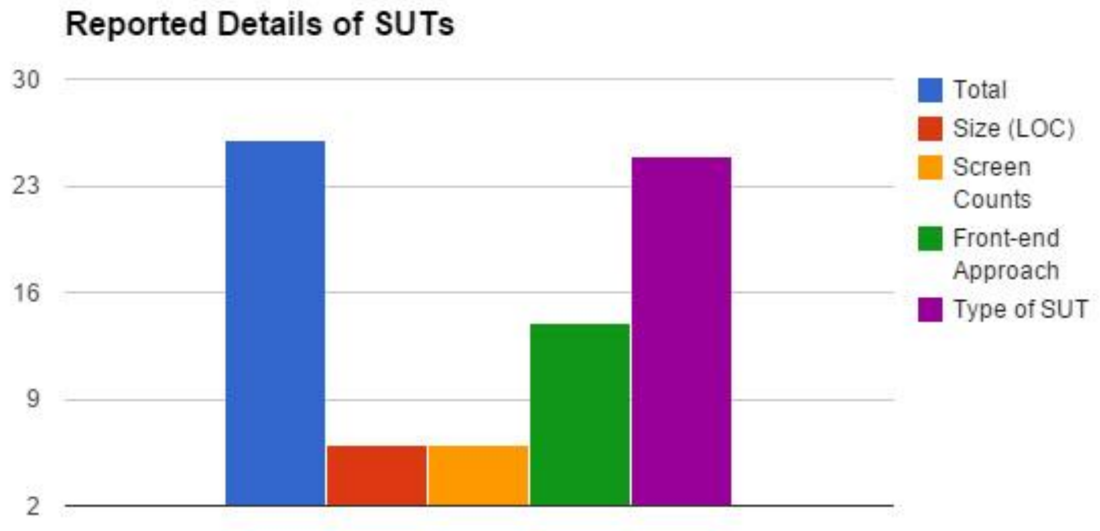


Figure 14- Reported Details of SUTs

We got the size of SUT information in terms of line of codes (LOCs). This information was reported for 31 of SUTs in only six articles (22%) [S1, S2, S8, S12, S23, S27]. The average size value is 53370 LOCs for each article. We have presented the size distribution over articles in Figure 15.

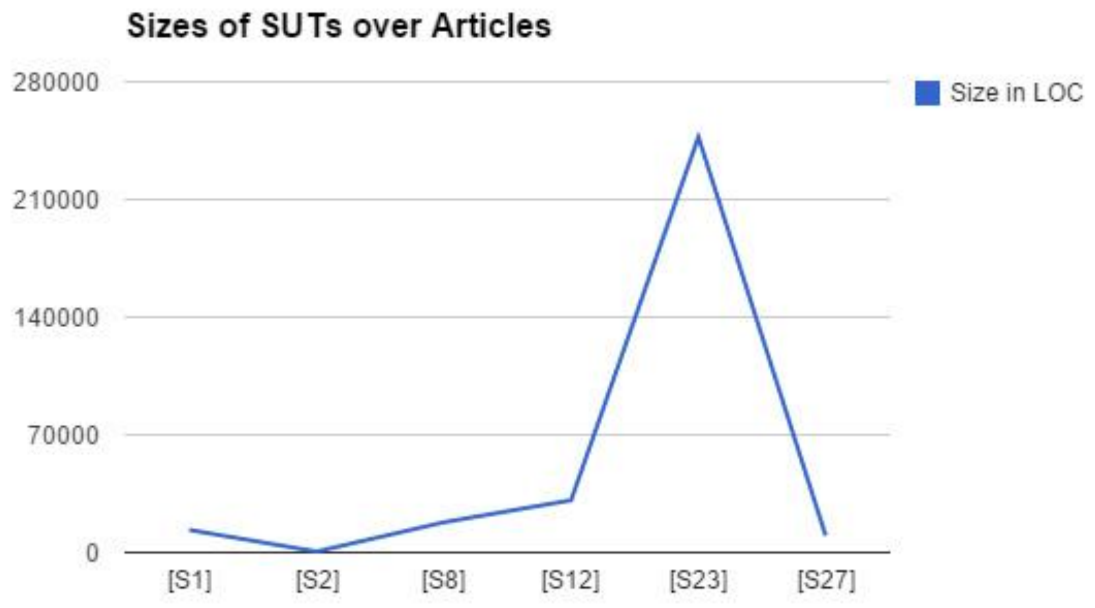


Figure 15- Sizes of SUTs over Articles

The screen counts of SUTs are valuable information for GUI testing. This information was also reported for 58 SUTs in six studies [S1, S2, S6, S21, S22, S25]. The average screen count value is 172 screens per article these studies that specify screen count of the SUTs. We have shown the screen counts distribution in Figure 16.

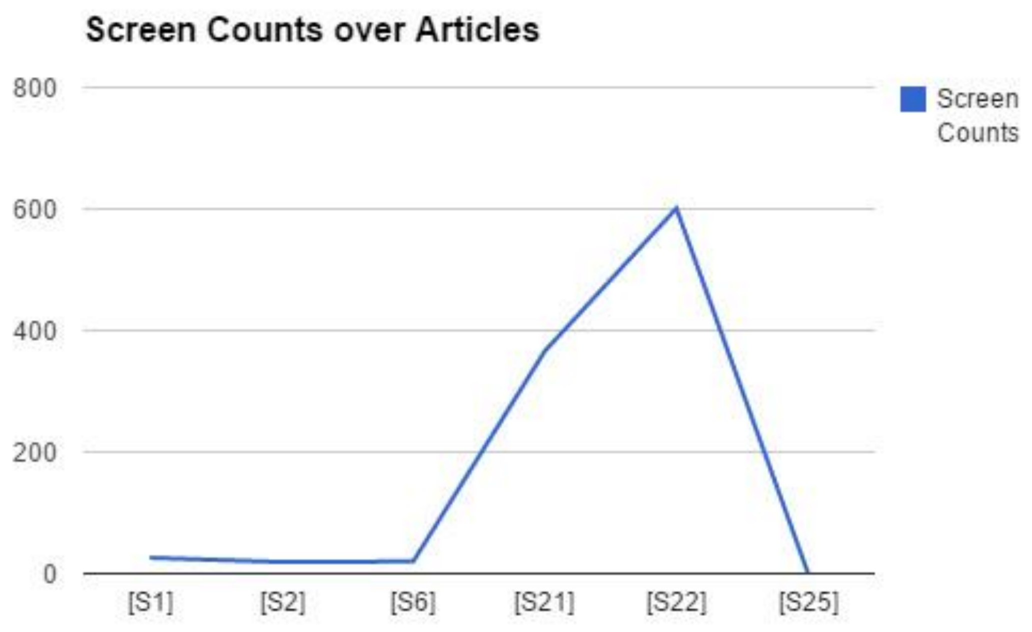


Figure 16- Screen Counts over Articles

The front-end approach information was reported for 151 SUTs in 14 studies. In Figure 17, we have presented the distribution of this approach. In almost every reported SUT which were tested in the studies, native Android is used as their front-end approach. There was only one SUT in a study [S17] different from other. In this study, S. Methong has used CeBIT2go as the SUT where the web based approach (HTML, CSS, and JavaScript) is used as its GUI.

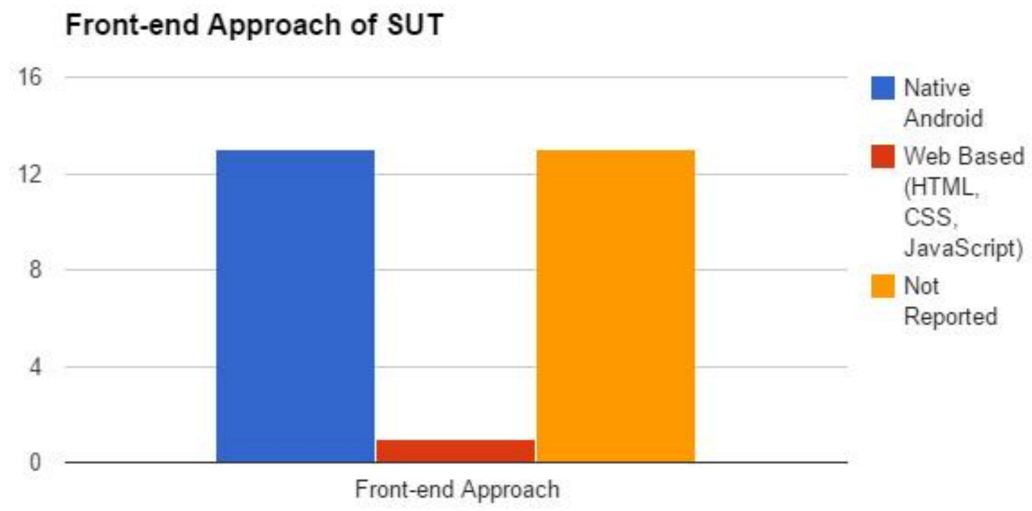


Figure 17- Front-end Approach of SUT

The type of SUT was the only information specified satisfactorily with the percentage of 92 (25 articles). We have presented the values of this classification in Figure 18. It is obvious that the most preferred type is commercial. Performing the tests on real world applications may provide to get more realistic results.

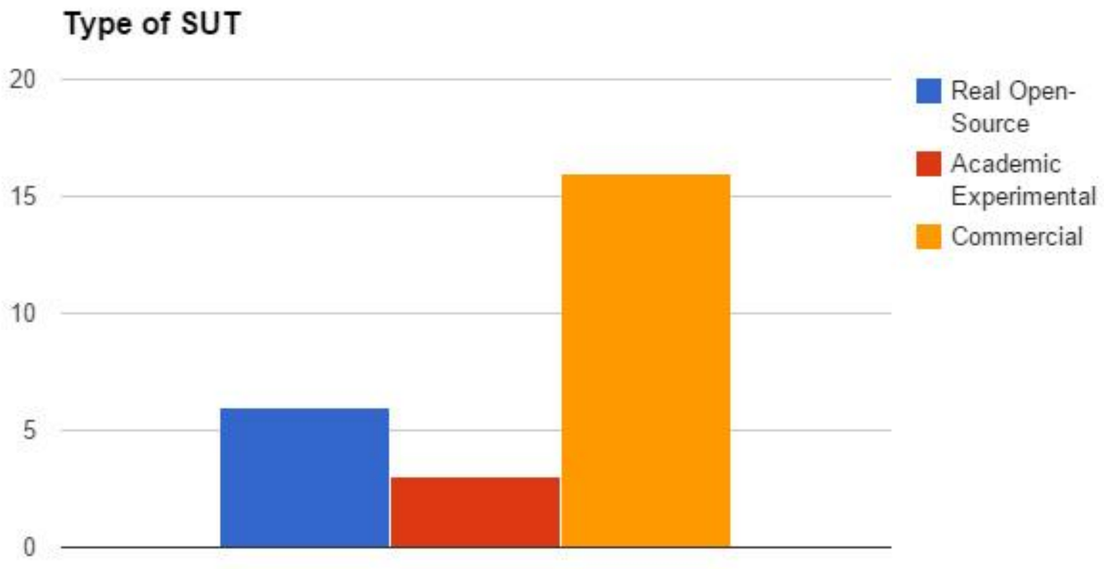


Figure 18- Type of SUT

4.2. What are the demographic and bibliometric aspects of the primary studies in Android GUI testing? (RQ 2)

We have analyzed this research question under three sub questions.

4.2.1. What is the articles count per year? (RQ 2.1)

In order to address this RQ, we got the date information of articles. Recall that, because our investigation starts from Android's first release date, we have searched for primary studies since 2009 up to October 11th, 2014. To present more comprehensive results, while illustrating the date information we have also pointed the types of studies published over years (Figure 19)

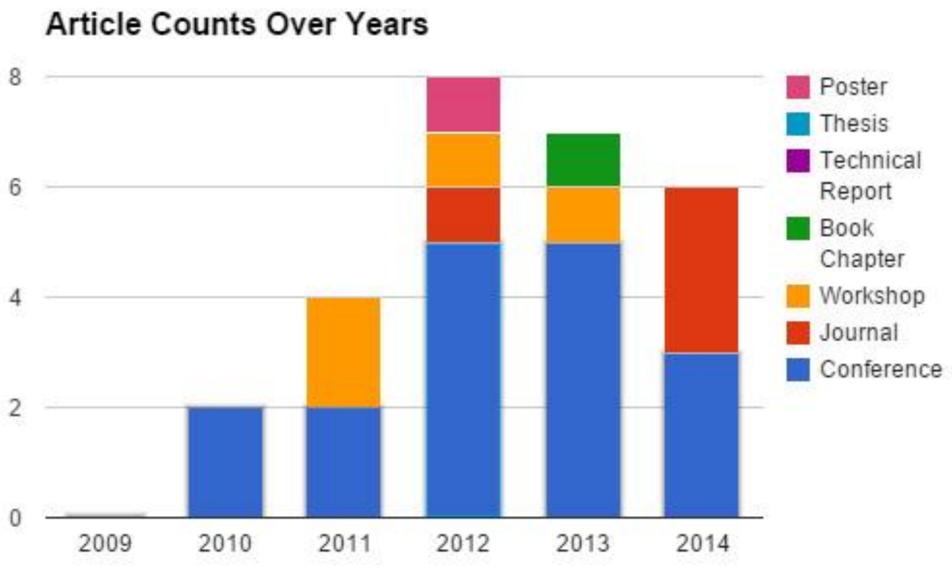


Figure 19- Articles Counts over Years

The article count can be considered as regular. Averages of last three years (2012 - 2014), 7 articles per years are published. The most common type of article that satisfies our inclusion/exclusion criteria was conference paper (17 papers).

4.2.2. Which are the most popular articles? Which are the mostly considerable the venues and the authors are in terms of the article count? (RQ 2.2)

The articles of the studies can be reached using several search engines which are specialized search to academic studies. We have also discussed about these search engines in section 3.3. However this investigation can be time consuming and tough task.

For this reason, determining some authors or some venues to follow make this research easier. Thus, in our study, we covered the entire domain and we have brought the information that demonstrates most considerable articles in term of citation count[9]. We have used this information to detect the most considerable venues and authors.

Firstly we have listed most considerable articles in term of normalized citation counts per year. Formula of normalized citation is described below. Table 9 consists of the article whose normalized citation is higher than five. There were 13 articles.

$$\text{Normalized Citation} = \frac{\text{Citations (as of December, 2014)}}{\text{Article age}}$$

Table 9- Articles Mostly Cited

Articles Mostly Cited				
Article Name	**Year**	**Citations (as of December, 2014)**	**Normalized citations**	**Ref.**
Guided GUI testing of android apps with minimal restart and approximate learning	2013	13	6.5	[S14]
Testing Android apps through symbolic execution	2012	25	8.33	[S25]
Targeted and depth-first exploration for systematic testing of android apps	2013	22	11	[S22]
A GUI crawling-based technique for android mobile application testing	2011	51	12.75	[S2]
Experiences of system-level model-based GUI	2011	52	13	[S11]

[9]Citation counts are taken from *http://scholar.google.com.*

testing of an Android application				
RERAN: timing- and touch-sensitive record and replay for Android	2013	26	13	[S20]
A grey-box approach for automated GUI-model generation of mobile applications	2013	31	15.5	[S1]
GUI testing using computer vision	2010	83	16.6	[S13]
Automated concolic testing of smartphone apps	2012	60	20	[S5]
Dynodroid: An input generation system for Android apps	2013	43	21.5	[S9]
Automating GUI testing for Android applications	2011	87	21.75	[S7]
Using GUI ripping for automated testing of Android applications	2012	66	22	[S27]

From the venues perspective, the distribution of articles was not very suitable to infer the active venues in this area. There were only three venues where more than one article has been published in this area. Thus making an inference would not be proper. We have listed them in Table 10.

Table 10- List of Venues

Venue Name (Abbreviation)	**Article Count**
Software, IEEE	3
ICSTW	2
OOPSLA	2
ICCET	1
FSE	1

CSNT	1
AST	1
ESEC/FSE	1
EuroSys	1
ICST	1
ISSTA	1
CHI	1
ICSM	1
FASE	1
IPCBEE	1
IWSEC	1
ICSE	1
CGO	1
Advances in Computers	1
MUM	1
ACM SIGSOFT Software Engineering Notes	1
ICSECS	1
ASE	1

In Table 11, we have listed the authors who have more than one article. It can be seen that D. Amalfitano, A. R. Fasolino, and P. Tramontana are the most productive researchers with 5 articles in this area.

Table 11- List of Most Considerable Authors

Author Name	**Article Count**
Domenico Amalfitano	5
Anna Rita Fasolino	5
PorfiroTramontana	5
TommiTakala	2
Ying-Dar Lin	2
TanzirulAzim	2
MayurNaik	2
Mika Katara	2
Chu	2
IulianNeamtiu	2
Salvatore De Camine	2
Atif M. Memon	2

4.2.3. What is the article distribution over countries? (RQ 2.3)

The articles may be published from all around the world. Counts of articles published from one country, may show the tendency to the subject. Thus we have gathered the counts for each country. However, we have decided that not only quantity but also popularity is an important metric while measuring the tendency. For this reason, we have also calculated citation normalization the citation per article (CPA) for each country. In order to get this value, first we summed up the citation counts for each article. Then we divided it to its age to get the average citation count over years. Finally, we have divided the summation of citation normalization to article count of country. As a result, we have had average citation normalization for each country. The calculation formula is given in below. We have represented both of the information in Figure 20.

$$\text{Citation Normalization per Article (CPA)} = \frac{\text{Sum of Normalized Citations for all Article from the Country (see Section 4.8)}}{\text{Article Count for the Country}}$$

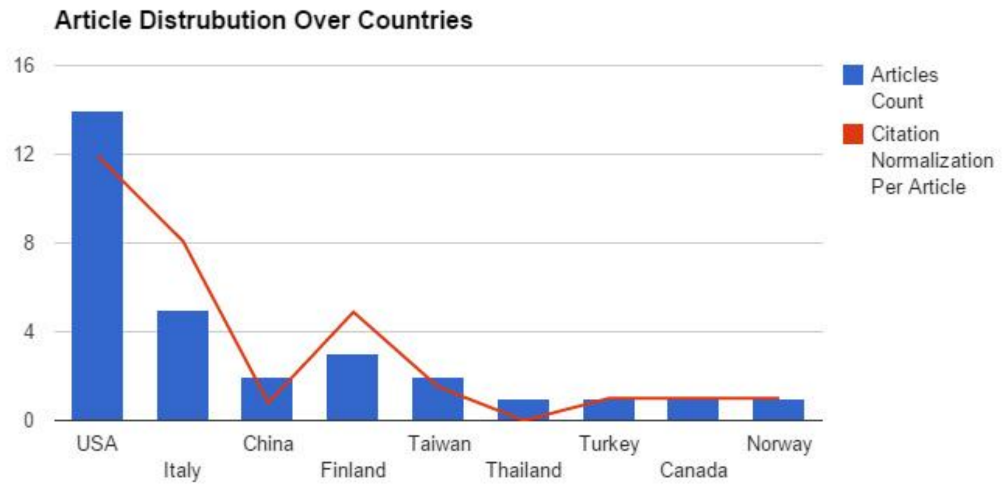


Figure 20- Article Distribution over Countries

It can be deduced that, the USA is the leading country for both quantity and quality criteria. Totally, 14 articles have been published from this country and the average citation count for each article is 12 per year. Italy is the second country in ranking. There were five article published from this country with 8 CPA and not very surprisingly, all of these article were written by D. Amalfitano, A. R. Fasolino, and P. Tramontana who are the most productive researcher in this area (see 4.8). Another notable country, Finland, also has three articles. These articles were also quite popular in the domain with 5 CPA. Other countries have not published adequate articles yet to analyze.

4.3. What are the trends and future direction in GUI testing of Android applications? (RQ 3)

We have examined this research question under five sub questions.

4.3.1. What are the types of articles published? (RQ 3.1)

As we have discussed in Section 3.6.2, in [30], Wieringa et al have created a classification scheme that defines type of articles according to their manner of approaching to the problems defined in the articles. This classification analyses the studies in six different approaches; Solution proposal, validation research, evaluation research, and experience papers, opinion papers, and philosophical papers. Due to the inclusion/exclusion criteria of our study, there is no opinion and philosophical paper in our pool. We have presented the results for this classification in Figure 21.

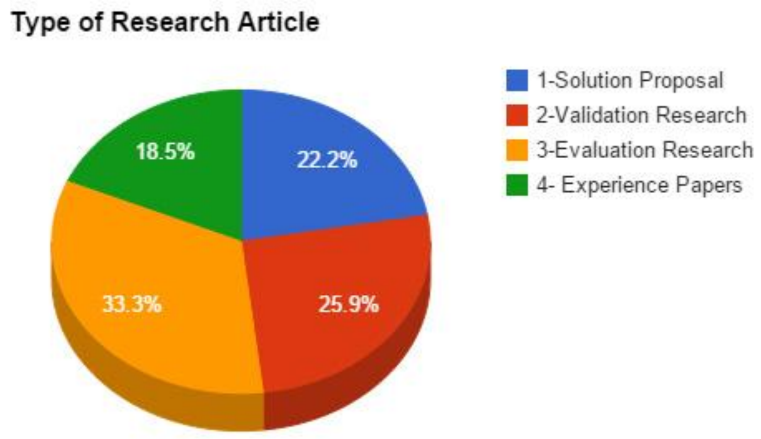


Figure 21- Type of Research Article

The percentages of articles are nearly same. The empirical studies (validation and evaluation researches) are more than half of the article pool (59%). The researchers usually make contributions such as techniques and tools (see 4.11).

Experience papers were based on several subjects such as suggestions of the authors about tools or techniques [S18, S26], applying an existing tool/technique to a case [S11, S17], or investigating the effectiveness of an existing suite [S23].

4.3.2. What are the contributions provided by researchers? (RQ 3.2)

We have decided that the distribution of the contributions of the articles could be a base knowledge in order to infer recent trends. Recall that, in Section 3.6.1, we have defined five types of contribution which are test method/technique, test tool, test model, metric, and process and the "other" for remaining contributions. The data collected from primary studies could be found in Figure 22.

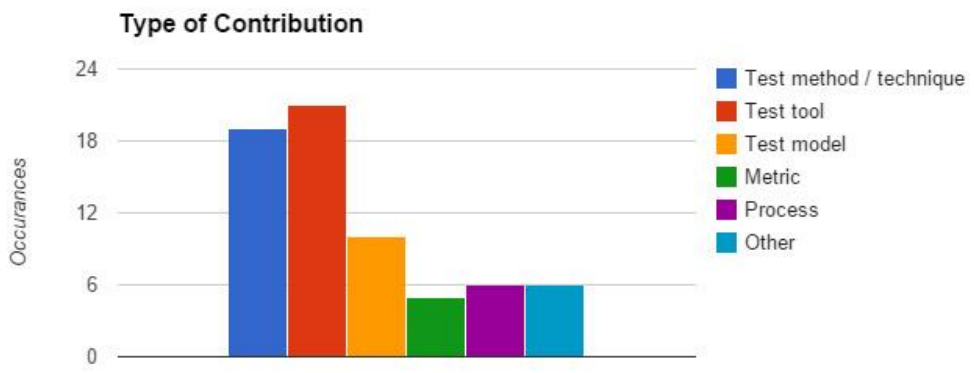


Figure 22- Type of Contribution

In total 19 test tools and 21 test techniques were proposed as contributions to the Android GUI testing domain. The proposed test techniques were usually implemented in a test tools that is also novel [S1, S2, S3, S4, S5, S6, S9, S10, S12, S13, S14, S16, S19, S20, S22, S25].

The contribution of test model, metric or process has to involve a novelty. For test models this novelty was usually extraction approach of the model. In [S27], researchers have presented a technique called GUI ripping that extract the GUI model using reverse engineering. This paper was also mostly cited paper in our repository (see Section 4.8).

In the "other" contributions category there are six studies. In [S7] and [S9], researchers have made a research to categories the bugs in Android GUI testing. [S6] defined the difficulties while performing an automatic GUI testing. Another study [S12] have addressed a novel problem about invalid access of a thread to main GUI thread. The last type of contribution was case study which was provided in 2 studies [S18, S23].

4.3.3. Which are the most significant testing tools? Which are 3rd party components used by the tools? (RQ 3.3)

The process of testing has many tasks to carry out. Performing these tasks manually requires too much work-power, time and attention. Also, this manual approach may decrease the quality of the verification. Therefore test tools are needed in order to apply the testing process more successfully.

In the articles we have analyzed, the researchers have proposed or used many tools in order to perform an testing activity. Recall that in Section 4.11 there were 19 articles that propose a test tool. Because we aimed to collect the most significant tools, we have listed the tools that were used or proposed in the most cited articles in our pool as presented in Table 9. We have made a little description about what they are able to do in Table12.

Table 12- Test Tools

Name of Test Tool	**Description**	**Reference**
A2T2	• Able to detect java run time crashes • Has a crawler component which responsible of extracting the model. • It produces a crash report • Simulates user interactions • Generates test cases automatically • Executes test cases • Makes Oracle via API	[S2]
ACTEve	• Used concolic testing • Simulates user interactions • Generates test cases automatically • Executes test cases • No Oracle	[S5]
AndroidRipper	• Use GUI ripping to extract a model • Produces a crash report • Generates test cases automatically • Executes test cases • Oracle can be done manually by using the crash report	[S3]
Automatic Android App Explorer (A3E)	• Uses RERAN to perform record/replay action • Uses a strategy named Depth-first Exploration to simulate user actions and to get model of the SUT. • Generates test cases automatically • Executes test cases • No Oracle	[S22]
Dynodroid	• Generates random inputs to the SUT, then observes the reactions and enhance the input for next time by using this value • Generates test cases automatically • Executes test cases both automatically or manually • Manual oracle with emulator logs.	[S9]

ORBIT	• Uses a crawler to extract model of the SUT. • Uses static analysis to support the crawler • Generates test cases automatically • No execution • No Oracle	[S1]
RERAN	• Uses record/replay technique • Manual test case design • Automatic execution of test cases • No oracle	[S20]
Sikuli Test	• Manual test case design • Automatic generation of test cases • Uses OCR to verify test cases	[S13]
SwiftHand	• Uses active learning to extract the model as finite state machine • Automatic generation of test cases • Automatic execution of test cases • No oracle	[S14]

These tools generally use 3rd party component such executables, frameworks, libraries while they are functioning. A tool can use more than component at once. Even, some of developers use another tool as an external program for its functionalities to avoid developing it from scratch. Thus this information would be valuable for testers, users, and especially developers.

In Figure 23, we have demonstrated the trendiest third party components which are used in the primary studies. We have demonstrated the components used by more than one tool in this figure. The complete list of components is given in Table 13. It is observed that JUnit and Robotium Framework are the most trendy third party components which were used in five projects for the testing tools.

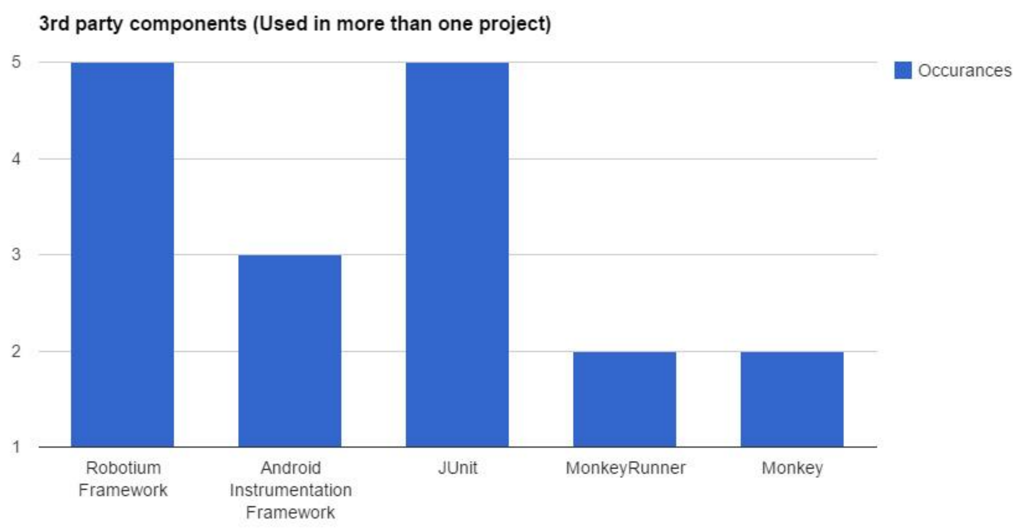


Figure 23- 3rd Party Components

Table 13- List of 3rd party Components

1. AndroidRipper	12. chimpchat	23. Robotium Framework
2. Android Driver	13. EmguCV	24. SCanDroid framework
3. Android Instrumentation Framework	14. EMMA	25. Selenium 2.0
4. Android Screencast	15. JUnit	26. Skuli
5. Android Sensor Framework	16. Microsoft WCF	27. Soot Analysis Framework
6. Android's Send Event Tool	17. MobileTest	28. Soot Framework
7. Apache CXF Framework	18. Modisco	29. SPAG
8. asmdex	19. Monkey	30. Symbolic Pathfinder
9. A-Tool by Symbio	20. Monkey Runner	31. TEMA Toolset

10. ATS4 AppModel	21. RERAN	32. Troyd Tool
11. axml	22. Robolectric	33. WALA static-analysis

4.3.4. What are the testing techniques/approaches used during test process? (RQ 3.4)

Determining commonly used techniques or approaches is an important task. A valuable knowledge is gathered as a result of this task. We believe that testers, tool designers, and even developers may use this knowledge for various purposes.

In Figure 24, we have demonstrated result of the data extraction according to our classification. In our classification scheme, we have included 12 techniques and put the "Other" category as 13th choice. Researchers could use multiple techniques in their studies. The techniques included in this scheme were defined in Section 3.6.7.

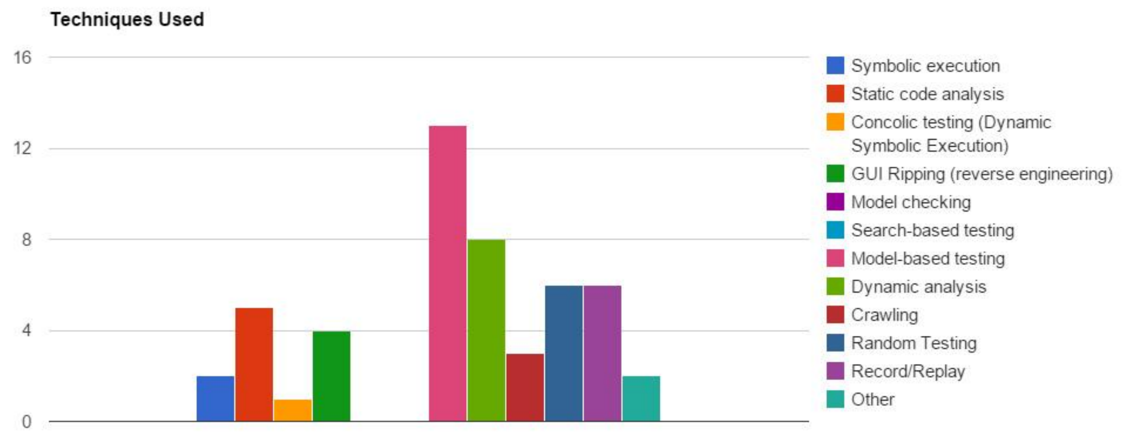


Figure 24- Techniques or Approaches Used In Researches

According to Figure 24, the mostly used technique is model-based testing [S3, S4, S6, S7, S10, S11, S12, S14, S16, S17, S18, S22, S25]. The GUI ripping technique was used in four articles [S3, S7, S15, S27]. All of these articles used a tool called AndroidRipper (see section 4.12) as a testing tool or a third party component. This tool has been proposed by Amalfitano et al who are of the most productive researchers in this area (see Table 11) in an article [S27] which is the most popular article in our pool (see Table 9). The model checking and search-based testing techniques were not used in any of the studies. Two of the studies use techniques that fell into the "other" category. W. Choi et al [S14] have used the active learning to extract the model of the application. Additionally in [S26], Sadeh et al have used the code refactoring.

4.3.5. What are the future directions of current researches? (RQ 3.5)

In most of the articles, researchers have declared their road map for future studies. These plans can be used to foresee the future trends, needs or points that are not addressed adequately yet. Therefore, we have presented future plans of researches of the primary studies with a classification scheme. In this scheme, we have gathered the plans under five main headings. As discussed in Section (3.6.13), the headings were; developing a new tool, improving the current tool, developing a new technique, improving the current technique, and making the research in more detailed manner. The researchers may aim to realize more than one goal in the future. The last heading was “no future plan”.

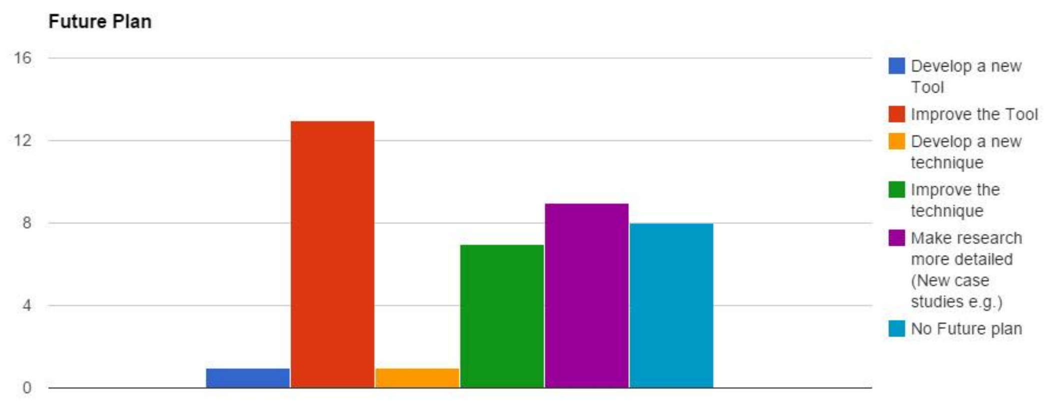


Figure 25- Future Plan

When analyzing the results, most of articles (19 of 27) plans were specified as future work. Most of these plans were on improving the actual test tool or the technique (in 16 studies when intersections were eliminated). In only one study [S8], the authors aimed to develop a new testing tool while improving existing one which is called Extended Ripper and making the research more detailed. Also A. Jääskeläinen et al [S18] intended to improve current condition of their tool and technique. However, they have suggested developing static tools to debug the models because of existing model debugging approach's slowness. Finally, developing a new technique was also set as a future goal in only one study [S2]. In this study, D. Amalfitano et al have aimed to investigate and develop more convenient techniques to improve the efficiency of the current test suites.

CHAPTER 5

DISCUSSION

The results are discussed in this section. We have summed up the inferences and trends for each RQ. We have also presented implications for our target audience who are researchers, testers, and test tool developers.

RQ 1.1 - The testing activities applied in the primary studies: In this section, the results show that 66,6% of articles are criteria-based test case design, while 26% of articles are human knowledge-based. There are only 2 studies [S21, S23] which do not use test case design. 74% of researchers have made test automation, 92% of articles have made test execution, and 7% of articles have made other testing activities which are bug categorization [S7] and a novel static analysis study on GUI code by modeling their flows, associations and relations [S21]. Test case design and test execution are performed at almost every study. Usage of the others activities, such as test automation and test oracle is quite common.

RQ 1.2 - The sources of information used to derive test artifacts: The categories of sources used to derive test artifacts are automatically inferred models (55%), only test requirements (40%), source code (37%), both test requirements and source code together (11%), manually inferred models (7%). and logs (3%). Although it seems that the most common resources to derive test artifacts are using models, the models are not used individually. In 15 of 27 studies, more than one resource is used to create test artifacts. 40% of these cases, source code and extracted model are used together.

RQ 1.3 - The test artifacts derived: The most frequent test artifacts generated during test process is test input (88%). Test driver (74%), test oracle (55%) and test requirements (7%) follows it. There is no other test artifact generated other than these categories.

RQ 1.4 - The testing environment used to run tests: In the test process of Android applications, there have to be a system that runs Android OS. This system can be a real device like phone or tablet, or an emulator, specifying this information in the studies is important. In 22% of articles, this information is not specified. In the studies that specify the testing environment, emulator is used frequently (48%). While 14% of researchers using only real devices as their testing environment, in the remaining articles both the systems are used (14%).
The Android version the tests run on is also valuable information. However this information is not declared frequently. In 62% of studies the Android version is not specified. Android version 1.5 is used as the running platform as once [S4], version 2.1 is used as twice [S1,S23]. The counts of Android version declared in the studies are one for v1.5 [S4], two for v2.1[S11, S23], one for v.2.2 [S2], two for v2.3[S9, S19], one for 2.3.3 [S8], one for 2.3.4 [S20], and one for 4.1.2 [S14]. There is only one study [S24] that works on multi Android versions.

RQ 1.5 - Ways of simulation of user interactions and (oracles): The user interaction is generally performed by capturing widget objects on the GUI (44%). The trendiest way to capture widget object to simulate user interactions is using the extracted the GUI model [S1, S2, S3, S8, S16, S27]. The second way to simulate user interactions on the GUI is coordinate based technique (26%). In 26% studies these techniques are used together. In the studies that coordinate based approach is used (the studies which uses both the approaches are included), the record/replay technique is commonly used [S4,S13,S15,S20,S21,S23]. There is only one article [S21] that does not simulate user interactions.

RQ 1.6 - The GUI oracle techniques: The oracle mechanism, which determines whether a test has passed or failed, is realized most frequently by checking widgets via API (15% of the studies) and manually (15% of the studies).Model-based (11%) oracles and Optical Character Recognition (OCR) a.k.a Computer Vision (11%) are the next frequent techniques. Bitmap Comparison is used in 7% of the studies as oracles. Besides, there are five articles which verify the GUI behaviors other than these ways. These studies makes their oracle by using JUnit's assertion mechanism [S3, S16, S25], by using graph construction algorithm [S12], or by detecting run time crashes [S27].

RQ 1.7 - The types of evaluation methods: The most common evaluation method is manual comparison (74%). The followers are the usage of other techniques such as coverage (52%), time/performance (63%) and detecting real faults (48%). We have observed that these techniques are usually used together. All of these techniques are used more than half of the studies except mutation testing. Mutation testing is used in only 2 studies [S2, S23].

RQ 1.8 - The attributes of the system under test (SUT): In 96% of the articles (except [S13]), one or more SUT is used to evaluation. While only one SUT is used in six studies [S2,S11,S17,S25,S26,S27], in remaining 19 articles, 328 SUTs are tested in total (we have excluded applications from [S24], because the author says that more than

20000 tests have been conducted since November 2011). The average of the SUTs is 17 SUTs per article for these articles.
We have observed that the details of the SUTs are not adequately reported. The percentages of the reported value are: 22% for size in LOC, 22% for number of screens, 52% for front-end approach, 93% for type (real open-source, academic experimental and commercial). The average value of the size of the SUT, in terms of LOC, is 53370 for each article which reports this information. The average screen counts are 172 for each article where the information also is reported. 90 percent of the reported front-end approaches of the SUTs are native android. In only one article web approach is used [17]. Most of the reported SUTs are commercial applications (64%). 24% of these SUTs are real open-source and remaining SUTs are academic experimental (12%).

RQ 2.1- Articles counts per year: The article counts over years seem uniform except 2009. While the averages of last three years (2012 - 2014) are 7 articles per year, there is no articles published in 2009. The most common type of article that satisfies our inclusion/exclusion criteria is conference paper (17 papers) which is also distributed uniformly over years.

RQ 2.2- The most popular articles and the mostly considerable the venues and the authors: We have listed the articles which are cited more than for 5 times per year. We got 12 articles (44%). We also listed the venues according to their article counts. There are only 3 venues which has published more than one article in this area. Thus making an inference will not be proper. Finally, we specified the authors in terms of article counts. Domenico Amalfitano, Anna Rita Fasolino, and Porfiro Tramontana are the most productive authors with five articles. They also worked together in all of these articles.

RQ 2.3- The article distribution over countries: USA is the leading country with 14 articles. The average citation count for each article is 12 citations per year. Italy is the second country in ranking. There are five articles published from this country. The average citation count for each article is 8 citations per year.

RQ 3.1- The types of articles published: 33% for the evaluation researches, 25% for the validation researches, 22% for the solution proposals, and 18% experience papers. The validation and evaluation researches can be considered as empirical studies (59%). This trend shows that authors not only propose a solution in this area but also evaluate their proposals in detailed experiments.

RQ 3.2- The contributions provided by researchers: We have classified the contributions which of the articles as: test methods/techniques (77%), test tools (70%), test model (37%), metric (18%), process (22%). The contributions other than these categories are proposed in six studies [S7, S9, S12, S18, S23]. The contributions are; bug categorizations for Android GUI testing [S7, S9], defining the difficulties while performing an automatic GUI testing [S6], addressing a novel problem about invalid access of a thread to main GUI thread [S12], and two case studies which [S18, S23].

RQ 3.3 - The most significant testing tools and the trendiest third party components used by the tools: In order to list the most considerable testing tools, we have investigated the mostly cited articles we have listed before. The mostly cited articles list consists of 12 articles. However 3 of them do not propose or use a tool to demonstrate. Therefore we have listed 9 tools as the most significant: A2T2, ACTEve, AndroidRipper, Automatic Android App Explorer (A3E), Dynodroid, ORBIT, RERAN, Sikuli Test, and SwiftHand.
We have broken down the third party components used by the testing tools. The most popular components were Robotium Framework and JUnit which were used in 5 tools.

RQ 3.4 – The testing techniques/approaches have been used: We have made a classification about techniques which consist of 9 techniques which are used in the articles: Model-based testing (48%), Dynamic analysis (29%), Random testing (22%), Record-replay analysis (22%), Static program analysis (18%), GUI ripping (15%), Symbolic execution (7%), Concolic testing (Dynamic Symbolic Execution) (3%), Model checking (0%), Search-based testing (0%). In a study, more than one technique can be used. The first significant observation from this section is that the most popular approach is model-based testing in this area. In contrast, search-based testing and model checking are not used in any of the articles.

RQ 3.5 - The future directions: Most of researchers (70%) have plans for future works. 84 percent of these plans (16 of 19) are on improving the actual test tool or the technique exclusively. In only one study [S8], the authors aim to develop a new testing tool. And also D. Amalfitano et al [S2] are the only researchers that intend to develop new techniques in order to improve the efficiency of the current test suites.

Implications from Researchers' Perspective

In this study, we have presented the information which can be used by the researchers to make inferences about our area. Using our study, a researcher can follow the trends about article types, contributions provided, testing activities and future directions of current studies. 27 studies are included in this SM. This count is less than expected for this 5-year-period (from 2009 to 2014). More researchers should make studies on this area. The most common article type is conference paper. Therefore, more mature articles such as journals are also required. In contribution facet, most of the researchers have proposed test methods and they also have implemented these methods into testing tools. These tools are used for evaluations so we suggest researchers to resume this approach. Testing activities are similar for most of the studies. Test case design (in most of cases, criteria based test case design has been preferred rather than human knowledge-based test case design), test automation, and test execution were applied in almost all of the studies. However test oracle was not very common. Therefore, researcher should pay more attention to the test oracle activities. For future studies, almost all of the researchers have declared the future goals about their topics. For that reason, in future, the studies will be examined further. This also can enhance the actual maturity level of the domain. On the other hand, in most of the studies, the version of Android where the testing process was performed was not declared. This lack of knowledge should be resolved for future studies.

We also demonstrated the evaluation techniques of the proposed testing tools or techniques and attributes of SUTs used in current studies. In primary studies, common evaluation techniques were manual comparison, coverage, time/performance, detecting real faults. On the other hand, mutation testing was used rarely. In most of cases more than one evaluation techniques were used. It is also useful information for researchers while determining the evaluation approaches of their studies.

The most popular articles, we presented, can be used to work on follow up studies in order to make the research deeper. In addition we have observed the most productive researchers who are Domenico Amalfitano, Anna Rita Fasolino, and Porfiro Tramontana from Italy. A researcher, who wants work on this domain, can also follow these researchers. On the other hand, because the articles were distributed uniformly over venues, we couldn’t determine the venues worth to follow.

This study is also a comprehensive literature review. It provides a well-defined set of studies which satisfy specific criteria. Therefore it helps the researchers while studying on subjects related to our area.

Implications from Testers' Perspective

Target audience of our study also consists of testers (a.k.a test engineers) who bridge the gap between quality control (QC) and quality assurance (QA). In this study, we have examined the testing process of GUI in many ways such as testing activities, derived test artifacts and their sources, preferred testing environments, approaches of user interaction simulations, and approaches of GUI oracles. Model based test was the most preferred testing technique for Android GUI testing. Therefore model generation techniques are also common such as crawling and GUI ripping. Test artifacts were usually derived from (manually or automatically) inferred models of the SUTs. In most cases not only one but also complementary sources were used such sources codes or requirements of the SUTs. Usual test artifacts were test inputs, test drivers and test oracles. For testing environments, emulators were used more than real devices. In some studies both of these systems were used. This may enhance the reliability of the evaluation. The user interaction was generally performed by capturing widget objects on the GUI. The most common way to capture widget object to simulate user interactions is using the extracted the GUI model. The oracle approaches, was performed frequently by checking widgets via API and manually. Model-based oracles and Optical Character Recognition (OCR) were also used in some of the studies. This information could be beneficial for the testers.

We have also presented the mostly preferred techniques with appropriate tools which can be used by the testers in GUI testing activities in Android applications. We have listed the most significant tools which were A2T2, ACTEve, AndroidRipper, Automatic Android App Explorer (A3E), Dynodroid, ORBIT, RERAN, Sikuli Test, and SwiftHand. These tools have been used for test processes such as test cases generation, test execution and test oracle.

Implications from Test Tool Developers' Perspective

Test tool developers are also in our target audience. In our study, we provided the mostly preferred test tools used in our domain. The implementation details of proposed testing tools are usually given in the related studies. These details often consist of development steps, preferred development languages, components, and evaluation techniques. This information can be guideline for developers. Although it is not gathered in this study, it can be reached by examining the related studies. We have also listed the abilities of the tool used in the most popular articles.

We have presented the third party components used in tool development as building blocks. The most popular components were Robotium Framework, JUnit, Android instrumentation framework, monkey tool, and monkey runner tool. Each component performs a specific task. Therefore these components can be placed in another tool in order to avoid developing the part which performs similar task from scratch.

CHAPTER 6

CONCLUSION & FUTURE WORK

Mobile devices are becoming popular day by day. With the smart phone concept, because their abilities and capacities are increased, new and more capable applications are developed every day. However the development process of these applications contains many challenges in several ways. One of these ways is that the operating systems that are used to run smart phones are new platforms. Thus, there are not too many experienced developers in the industry. Another challenge is the novelties of the user interface approaches. More attractive and capable screens could be presented in the application. Accordingly, the UI design became more complicated task.

Android is the most popular OS in mobile domain. This success of Android increases the number of applications that run on Android. Because there are over 1 million applications in the Google Play since 2013, any positive effect in the Android application testing process can effect countless application. Therefore we decided to focus on the Android OS.

Due to the challenges of development process of Android applications, the applications could be more error-prone. Therefore, testing process is very important task for mobile applications. In order to set light to this issue, we decided make an SM study that is a well accepted study in software engineering to make an investigation on the GUI testing of Android applications area to present an outline and find out the conditions of the researches such as amounts, types, frequencies, results. We have applied this technique to all of the related studies that are collected with a literature review from 2009 to November 11th, 2014.

In our SM study, we followed steps defined in [13] to apply SM studies to software engineering areas. Towards these steps, first we defined our goal and corresponding research questions (RQ). Main goal was classifying and analyzing the literature on the subject GUI testing of Android applications area to specify to the type of researches, results of their studies and their tendency. Given these purposes we defined our RQs. We asked three main RQs to define major titles and detailed sub-questions to get information in specified manners. Shortly our main RQs are about: (1) nature about Android GUI testing, (2) demographic and bibliometric aspects of the empirical studies in Android GUI testing, (3) the trends and future direction in GUI testing of Android application. We divided our first main RQ into 8 sub-questions. They are related following issues respectively: (RQ 1.1) testing activities applied, (RQ 1.2) sources of information used to derive test artifacts and (RQ 1.3) test artifacts generated during testing process, (RQ 1.4) Android systems used to run tests, (RQ 1.5) user interactions simulation approach and (RQ 1.6) verification of GUI behaviors approaches, (RQ 1.7) types of evaluation methods, and (RQ 1.8) attributes of the system under test (SUT). We have described three sub-questions to our second RQ. The sub-questions are about respectively: (RQ 2.1) articles count per year, (RQ 2.2) venues and authors are mostly considerable in terms of the article and citation counts, (RQ 2.3) the article distribution over countries. The subjects of sub-questions of our last RQ are: (RQ 3.1) types of articles published and 3rd party components used by these tools, (RQ 3.2) contributions provided by researchers, (RQ 3.3) testing tools are used or proposed, (RQ 3.4) techniques/approaches have been used, (RQ 3.5) future directions of current researches.

In this study, there are 27 articles published related to our domain between 2009 and November 11th, 2014. The article count is less than expected for this 5-year-period. Although most of these articles propose tools or techniques, it can be said that the maturity level of this area is not very high. The studies presented in the articles are rarely improved with the follow-up studies. There are 5 articles [S2, S3, S8, S16, S27] which can be considered as follow-up studies for each other. One of these studies [S27] proposes a tool called AndroidRipper which is also a well-accepted testing tool in this area. Another base study is done by Chang et al [S13]. The tool called Sikuli is used as a third-party component for the tools proposed in two articles [17, 19]. The study done by Gomez et al [S20] was also an inspirational work for an article [S22]. Azim et al [S22] use the tool called RERAN in their testing tool. There should be more follow-up studies to improve the maturity level of this area.

The model-based technique is the most preferred testing technique in this domain. In 13 studies [S3, S4, S6, S7, S10, S11, S12, S14, S16, S17, S18, S22, S25], the model of the SUT is extracted. However the technique is used to extract the model can vary. Most popular extraction techniques are: (1) GUI Ripping used in four articles [S3, S8, S16, S27] and (2) Crawling used in three articles [S1, S2, S5]. Note that in some articles [S1, S2, S27], although the model is extracted, researchers does not finish the testing process. Therefore there may not be a model-based testing.

All of the articles but one [S23], propose or use testing tools. However, not all of them are available to download. Only the tools used in 12 of articles are able to be reached on the web. We have listed these tools on Table 14. Note that AndroidRipper is used in two studies [S3, S27] so we have listed 11 testing tools.

Table 14- Testing Tool Available to Download

Tool Name	URL
AndroidRipper	https://github.com/reverse-unina/AndroidRipper
ACTEve	https://code.google.com/p/acteve/
Dynodroid	http://pag.gatech.edu/dynodroid
GUIErrorDetector	http://guierrordetector.googlecode.com
Sikuli	http://www.sikuli.org/
SwiftHand	https://github.com/wtchoi/SwiftHand
TEMA	http://tema.cs.tut.fi/
RERAN	http://www.androidreran.com/
Automatic Android App Explorer (A3E)	http://spruce.cs.ucr.edu/a3e/
Testdroid	http://testdroid.com/
Robolectric	http://robolectric.org/

The SUTs are usually used to evaluate techniques or tools. Therefore the specifications of the SUTs are important while deciding the suitability of these tools/techniques. However the details of SUTs are not reported adequately (see Figure 14). Likewise, the Android version is rarely specified too (see Figure 10). Researchers should pay more attention on the reporting issue.

The articles can be published from any country the world. However, it can be easily deduced that, the there are two dominating countries in this area: USA and Italy. From both of the countries, 19 articles (14 from USA, 6 from Italy) are published. This takes 70 percent of the article pool. Rest of the articles is published from 7 different countries.

We have collected the future plans information from each article to determine the future directions of the area. According to information, the mostly declared future plan is improving the actual tool proposed (48%). Only in the 29 percent of the articles, the future plan is not declared. Therefore, this area seems to be more attraction center for the researchers with improving participation in future.

For future work, we want to keep our repository [22] up to date. We will search for new articles periodically to involve them to our study. Thereby, the trends and novelties can be followed more precisely. In addition, a SLR study can be conducted as a follow-up complementary study based on this study. This SLR study will continue to work on the focus of our SM study with a further thought.

REFERENCES

References of Primary Studies

[S1] Wei Yang, Mukul R. Prasad, Tao Xie. (2013). A grey-box approach for automated GUI-model generation of mobile applications. *FASE.*

[S2] Domenico Amalfitano, Anna Rita Fasolino, Porfiro Tramontana. (2011). A GUI crawling-based technique for android mobile application testing. *ICSTW.*

[S3] Domenico Amalfitano, Anna Rita Fasolino, Porfiro Tramontana, Salvatore De Camine, Gennaro Imparato. (2012). *A toolset for GUI testing of Android applications.* ICSM.

[S4] Zhifang Liu, Xiaopeng Gao, Xiang Long. (2010). Adaptive Random Testing of Mobile Application. *ICCET.*

[S5] Saswat Anand, Mayur Naik, Hongseok Yang. (2012). Automated concolic testing of smartphone apps. *FSE.*

[S6] Peng Wang, Bin Liang, Wei You, Jingzhe Li, Wenchang Shi. (2014). Automatic Android GUI Traversal with High Coverage. *CSNT.*

[S7] Cuixiong Hu, Lulian Neamtiu. (2011). Automating GUI testing for Android applications. *AST.*

[S8] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Nicola Amatucci. (2013). Considering Context Events in Event-Based Testing of Mobile Applications. *ICSTW.*

[S9] Aravind Machiry, Rohan Tahiliani, Mayur Naik. (2013). Dynodroid: An input generation system for Android apps. *ESEC/FSE.*

[S10] Gang Hu, Xinhao Yuan, Junfeng Yang, Yang Tang. (2014). Efficiently, effectively detecting mobile app bugs with AppDoctor. *EuroSys.*

[S11] Tommi Takala, Mika Katara. (2011). Experiences of system-level model-based GUI testing of an Android application. *ICST.*

[S12] Sai Zhang, Hao Lü, Michael D. Ernst. (2012). Finding errors in multithreaded GUI applications. *ISSTA.*

[S13] Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller. (2010). GUI testing using computer vision. *CHI.*

[S14] Wontae Choi, George Necula, Koushik Sen. (2013). Guided GUI testing of android apps with minimal restart and approximate learning. *OOPSLA.*

[S15] Ying-Dar Lin, Chu, E.T.-H., Shang-Che Yu, Yuan-Cheng Lai. (2014). Improving the Accuracy of Automated GUI Testing for Embedded Systems. *Software, IEEE* .

[S16] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, Atif M. Memon. (2014). MobiGUITAR -- A Tool for Automated Model-Based Testing of Mobile Apps. *Software, IEEE* .

[S17] Methong, S. (2012). Model-based Automated GUI Testing For Android Web Application Frameworks. *IPCBEE* .

[S18] Antti Jääskeläinen, Tommi Takala, Mika Katara. (2012). Model-based GUI testing of Android applications. *IWSEC.*

[S19] Ying-Dar Lin, Rojas, J.F., Chu, E.T.-H., Yuan-Cheng Lai. (2014). On the accuracy, efficiency, and reusability of automated test oracles for android devices. *Software, IEEE* .

[S20] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, Todd Millstein. (2013). RERAN: timing- and touch-sensitive record and replay for Android. *ICSE.*

[S21] Atanas Rountev, Dacong Yan. (2014). Static Reference Analysis for GUI Objects in Android Software. *CGO.*

[S22] Tanzirul Azim, Iulian Neamtiu. (2013). Targeted and depth-first exploration for systematic testing of android apps. *OOPSLA.*

[S23] Vahid Garousi, Riley Kotchorek, Michael Smith. (2013). Test Cost-Effectiveness and Defect Density: A Case Study on the Android Platform. In *Advances in Computers.*

[S24] Jouko Kaasila, Denzil Ferreira, Vassilis Kostakos, Timo Ojala. (2012). TestDroid:Automated Remote UI testing on Android. *MUM.*

[S25] Nariman Mirzaei, Sam Malek, Corina S. Păsăreanu, Naeem Esfahani, Riyadh Mahmood. (2012). Testing Android apps through symbolic execution. *ACM SIGSOFT Software Engineering Notes* .

[S26] Ben Sadeh, Kjetil Ørbekk, Magnus M. Eide, Njaal C. A. Gjerde, Trygve A. Tønnesland, Sundar Gopalakrishnan. (2011). Towards Unit Testing of User Interface Code for Android Mobile Applications. *ICSECS.*

[S27] Domenico Amalfitano, Anna Rita Fasolino, Porfiro Tramontana, Salvatore De Camine, Atif M. Memon. (2012). Using GUI ripping for automated testing of Android applications. *ASE.*

Other References

[1] Last accessed November 25, 2014, from Strategy Analytics, Android Hits Record 85 Percent Share of Global Smartphone Shipments in Q2 2014: http://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=9921

[2] Last accessed November25, 2014, from http://www.qrcodescanning.com/smartphonehist.html

[3] Last accessed November 25, 2014, from http://www.statista.com/statistics/263401/global-apple-iphone-sales-since-3rd-quarter-2007/

[4] Last accessed November 25, 2014, from http://www.tutorialspoint.com/android/android_architecture.htm

[5] Last accessed November 25, 2014, from http://developer.android.com/guide/components/index.html

[6] Last accessed November 25, 2014, from http://developer.android.com/tools/help/monkey.html

[7] Last accessed November 25, 2014, from http://developer.android.com/tools/help/monkeyrunner_concepts.html

[8] (2009, April 27). Last accessed August 22, 2014, from http://android-developers.blogspot.com.tr/2009/04/android-15-is-here.html

[9] (2013, July 24). Last accessed 8 2014, 22, from http://www.androidguys.com/2013/07/24/sundar-pichai-there-are-now-more-than-1-million-android-apps/

[10] (2013, October 22). Last accessed August 22, 2014, from http://www.theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store

[11] B. Kitchenham, S. C. (2007). Guidelines for Performing Systematic Literature Reviews in Software engineering. *Evidence-Based Software Engineering* .

[12] Changjiang Jia, Y. T. (2013). Using the 5W+1H Model in Reporting Systematic Literature Review: A Case Study on Software Testing for Cloud Computing. *Quality Software (QSIC).*

[13] David Budgen, M. T. (2008). Using Mapping Studies in Software Engineering. *Psychology of Programming Interest Group* , (pp. 195-204).

[14] Érica F. Souza1, R. A. (2013). Ontologies in Software Testing: A Systematic Literature Review. *CEUR Workshop*, (pp. 71-82).

[15] Érica Ferreira de Souzaa, R. d. (2014). Knowledge management initiatives in software testing: A mapping study. *Information and Software Technology* .

[16] Frank Elberzhager, A. R. (2013). Analysis and testing of matlabsimulink models: a systematic mapping study. *International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation*, (pp. 29-34). New York.

[17] Garousi, V. *Online Paper Repository for Developing Scientific Software: A Systematic Mapping.* Last accessed August 22, 2014, from http://softqual.ucalgary.ca/projects/2012/SM_CSE/

[18] Garousi, V. *Online Paper Repository for GUI Testing.* Last accessed August 22, 2014, from http://www.softqual.ucalgary.ca/projects/2012/GUI_SM/

[19] Garousi, V. *Software Test-Code Engineering: A Systematic Mapping.* Last accessed August 22, 2014, from http://www.softqual.ucalgary.ca/projects/SM/STCE

[20] Gilmar Ferreira Arantes, P. d.-J. (2013). Functional Software Testing: A Systematic Mapping Study. *IARIA*, (pp. 11-17). Venice.

[21] Ishan Banerjeea, Bao Nguyena, Vahid Garousi, Atif Memon. (2013). Graphical user interface (GUI) testing: Systematic mapping and repository. *Information and Software Technology* , 1679-1694.

[22] Muzaffer Aydın, A. B.-C., V. Garousi*Online Paper Repository for GUI Testing of Android Applications: A Systematic Mapping.*Last accessed November 25, 2014, from http://tinyurl.com/k8x8que

[23] S. Dogan, A. B.-C., V. Garousi*Online SLR Data for Web Application Testing (WAT).* Last accessed August 22, 2014, from http://goo.gl/VayAr

[24] Serdar Doğan, A. B.-C., V. Garousi (2014). Web application testing: A systematic literature review. *Journal of Systems and Software* , 174-201.

[25] Upulee Kanewala, J. M. (2014). Testing scientific software: A systematic literature review. *Information and Software Technology* , 1219–1232.

[26] V. Basili, G. C. (1994). Chapter Goal Question Metric Approach. In *Encyclopedia of Software Engineering* (pp. 528-532). John Wiley & Sons Inc.

[27] Y. Jia, M. H. *Mutation Testing Paper Repository.* Last accessed August 22, 2014, from http://www.dcs.kcl.ac.uk/pg/jiayue/repository

[28] Zhang, Y. *Repository of Publications on Search based Software Engineering.* Last accessed August 22, 2014, from http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

[29] Atif M. Memon, Bao N. Nguyen B. N. (2010). Advances in Automated Model-Based System Testing of Software Applicaitions with a GUI Front-End. Advance in Computers, 121-162

[30] R. Wieringa, N. Maiden, N. Mead. (2005). Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering* , 102-107 .

[31] J. Offutt, P. Ammann. (2008). *Introduction to Software Testing.* Cambridge: Cambridge University Press.

[32] Koushik Sen, Darko Marinov, Gul Agha. (2005). CUTE: A Concolic Unit Testing Engine for C. *ACM SIGSOFT Software Engineering Notes*, (pp. 263-272). New York.

[33] J Zhi, V Garousi-Yusifoğlu, B Sun, G Garousi. (2014). Cost, Benefits and Quality of Software Development Documentation: A Systematic Mapping. *Journal of Systems and Software* .

[34] Muneer, I. (2014). Systematic Review on Automated Testing (Types, Effort and ROI).

[35] Pirzadeh, L. (2010). Human Factors in Software Development: A Systematic Literature Review.

[36] VG Yusifoğlu, Y Amannejad, AB Can. (2014). Software test-code engineering: A systematic mapping. Information and Software.

[37] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson. (2008). Systematic mapping papers in software engineering. International Conference on Evaluation and Assessment in Software Engineering .

[38] Atif M. Memon, Ishan Banerjee, and Adithya Nagarajan. (2003). GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. The 10th Working Conference on Reverse Engineering.

[39] Last accessed November 25, 2014, from http://junit.org/

[40] W. Afzal, R. Torkar, and R. Feldt, "A systematic mapping study on non-functional search-based software testing," in International Conference on Software Engineering and Knowledge Engineering, 2008, pp. 488-493.

[41] M. Palacios, J. García-Fanjul, and J. Tuya, "Testing in service oriented architectures with dynamic binding: A mapping study," Information and Software Technology, vol. 53, pp. 171-189, 2011.

[42] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and ValidationWorkshops, 2011, pp. 476 - 485.

[43] A. T. Endo and A. d. S. Simao, "A systematic review on formal testing approaches for web services," in Brazilian Workshop on Systematic and Automated Software Testing, International Conference on Testing Software and Systems, 2010, p. 89.

[44] P. A. d. M. S. Neto, I. d. C. Machado, J. D. McGregord, E. S. d. Almeida, and S. R. d. L. Meira, "A systematic mapping study of software product lines testing," Information and Software Technology, vol. 53, pp. 407 - 423, 2011.

[45] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," Information and Software Technology, vol. 51, pp. 957 - 976, 2009.

[46] Z. Zakaria, R. Atan, A. A. A. Ghani, and N. F. M. Sani, "Unit testing approaches for BPEL: a systematic review," in Proceedings of the Asia-Pacific Software Engineering Conference, 2009, pp. 316 - 322.

[47] E. Engström and P. Runeson, "Software product line testing - a systematic mapping study," Journal of Information and Software Technology, vol. 53, pp. 2 - 13, 2011.

[48] C. R. L. Neto, P. A. d. M. S. Neto, E. S. d. Almeida, and S. R. d. L. Meira, "A mapping study on software product lines testing tools," in Proceedings of International Conference on Software Engineering and Knowledge Engineering, 2012.

[49] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of searchbased test case generation," IEEE Transactions on Software Engineering, vol. 36, pp. 742 - 762, 2010.

[50] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," Journal of Information and SoftwareTechnology, vol. 53, pp. 14 - 30, 2010.

[51] R. V. Binder, "Testing object-oriented software: a survey," in Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems, 1996, p. 374.

[52] N. Juristo, A. M. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments," Empirical Software Engineering, vol. 9, pp. 7 - 44, 2004.

[53] P. McMinn, "Search-based software test data generation: a survey," Software Testing, Verification & Reliability, vol. 14, pp. 105 - 156, 2004.

[54] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," Software Testing, Verification, and Reliability, vol. 15, 2005.

[55] G. Canfora and M. D. Penta, "Service-oriented architectures testing: a survey," in International Summer Schools on Software Engineering, 2009, pp. 78 –105.

[56] C. S. Păsăreanu and W. Visser, "A survey of new trends in symbolic execution for software testing and analysis," International Journal on Software Tools for Technology Transfer, vol. 11, pp. 339 - 353, 2009.

[57] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing web services: a survey," Technical Report TR-10-01, Department of Computer Science, King’s College London2010.

[58] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," IEEE Transactions of Software Engineering, vol. 37, pp. 649 – 678, 2011.

[59] A. M. Memon and B. N. Nguyen, "Advances in automated model-based system testing of software applications with a GUI front-end," Advances in Computers, vol. 80, pp. 121–162, 2010.

[60] T. D. Hellmann, A. Hosseini-Khayat, and F. Maurer, "Agile Interaction Design and Test-Driven Development of User Interfaces - A Literature Review," in Agile Software Development: Current Research and Future Directions, 2010.

[61] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, "A Systematic Mapping Study of Web Application Testing," Elsevier Journal of Information and Software Technology, vol. 55, pp. 1374–1396, 2013.

[62] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical User Interface (GUI) Testing: Systematic Mapping and Repository," Information and Software Technology, vol. 55, pp. 1679–1694, 2013.

[63] V. G. Yusifoğlu, Y. Amannejad, and A. Betin-Can, "Software Test-Code Engineering: A Systematic Mapping," Journal of Information and Software Technology, In Press, 2014.

[64] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "A comprehensive Survey of Trends in Oracles for Software Testing," In Press, IEEE Transactions on Software Engineering, 2014.

[65] I. Singh, "A Mapping Study of Automation Support Tools for Unit Testing," MSc thesis, Mälardalens University, Sweden, 2012.

[66] C. Nie and H. Leung, "A survey of combinatorial testing," ACM Computing Surveys (CSUR), vol. 43, 2011.

[67] R. Lai, "A survey of communication protocol testing," Journal of Systems and Software, vol. 62, pp. 21–46, 2002.

[68] K. Inçki, I. Ari, and H. Sozer, "A Survey of Software Testing in the Cloud," in IEEE International Conference on Software Security and Reliability Companion 2012, pp. 18 - 23.

[69] J. A. McQuillan and J. F. Power, "A survey of UML-based coverage criteria for software testing," Technical Report, NUIM-CS-TR-2005-08, National University of Ireland, Ireland, 2005.

[70] A. C. D. Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A Survey on Model-based Testing Approaches- A systematic review," in Proceedings of the ACM International Workshop on Empirical Assessment of Eoftware Engineering languages and technologies, 2007.

[71] S. P. Shashank, P. Chakka, and D. V. Kumar, "A Systematic Literature Survey of Integration Testing in Component-Based Software Engineering," in International Conference on Computer and Communication Technology, 2010, pp. 562 - 568.

[72] F. Elberzhager, J. Münch, and V. T. N. Nha, "A systematic mapping study on the combination of static and dynamic quality assurance techniques," Information and Software Technology, vol. 54, pp. 1-15, 2012.

[73] M. Shafique and Y. Labiche, "A systematic review of state-based test tools," International Journal on Software Tools for Technology Transfer 2013.

[74] P. K. Singh, O. P. Sangwan, and A. Sharma, "A systematic review on fault-based mutation testing techniques and tools for Aspect-J programs," in IEEE International Advance Computing Conference, 2013, pp. 1455 - 1461.

[75] B. Haugset and G. K. Hanssen, "Automated Acceptance Testing-A Literature Review and an Industrial Case Study," in Agile Conference, 2008, pp. 27 - 38.

[76] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mantyla, "Benefits and limitations of automated software testing- Systematic literature review and practitioner survey," in International Workshop on Automation of Software Test, 2012, pp. 36 - 42.

[77] P. I. Chana and A. Rana, "Empirical evaluation of cloud-based testing techniques- a systematic review," ACM SIGSOFT Software Engineering Notes, vol. 37, pp. 1-9 2012.

[78] M. P. Usaola and P. R. Mateo, "Mutation Testing Cost Reduction Techniques: A Survey," IEEE Software, vol. 27, pp. 80 - 86, 2010.

[79] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," Proceedings of the IEEE, vol. 84, pp. 1090 - 1123, 1996.

[80] S. R. S. Souza, M. A. S. Brito, R. A. Silva, P. S. L. Souza, and E. Zaluska, "Research in concurrent software testing- a systematic review," in Proceedings of the Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging, 2011.

[81] P. McMinn, "Search-based software testing: Past, present and future," in IEEE International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 153 - 163.

[82] B. P. Lamancha, M. Polo, and M. Piattini, "Systematic Review on Software Product Line Testing," in International Joint Conference on Software Technologies, 2013, pp. 58-71.

[83] J. R. Barbosa, M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi, "Software Testing in Critical Embedded Systems- a Systematic Review of Adherence to the DO-178B Standard," inInternational Conference on Advances in System Testing and Validation Lifecycle 2011, pp. 126-130.

[84] Y. Singh, A. Kaur, B. Suri, and S. Singhal, "Systematic Literature Review on Regression Test Prioritization Techniques," SOURCEInformatica vol. 36, 2012.

[85] A. Kaur and V. Vig, "Systematic Review of Automatic Test Case Generation by UML Diagrams," International Journal of Engineering Research & Technology, vol. 1, 2012.

[86] M. Amar and K. Shabbir, "Systematic Review on Testing Aspect-oriented Programs: Challenges, Techniques and Their Effectiveness," Master Thesis, Blekinge Institute of Technology. Sweden,2008.

[87] Ç. Çatal and D. Mishra, "Test Case Prioritization: A systematic mapping Study " Software Quality Journal, vol. 21, pp. 445-478, 2013.

[88] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and Verification In Service-Oriented Architecture-A Survey," Journal of Software Testing, Verification and Reliability, vol. 23, pp. 261-313, 2013.

[89] A. Sharma, T. D. Hellmann, and F. Maurer, "Testing of Web Services – A Systematic Mapping," in Proceedings of the IEEE World Congress on SERVICES, 2012, pp. 346-352.

[90] M. J.-u. Rehman, F. Jabeen, A. Bertolino, and A. Polini, "Testing software components for integration: a survey of issues and techniques," Software Testing, Verification & Reliability, vol. 17, pp. 95-133 2007.

[91] G. Fraser, F. Wotawa, and P. E. Ammann, "Testing with model checkers: a survey," Software Testing, Verification and Reliability, vol. 19, pp. 215–261, 2009.

[92] S. Doğan, A. Betin-Can, and V. Garousi, "Web application testing: A systematic literature reviewr," Journal of Systems and Software vol. 91, pp. 174–201, 2014.

[93] M. A. A. Mamun and A. Khanam, "Concurrent Software Testing- A Systematic Review and an Evaluation of Static Analysis Tools," Master Thesis, Blekinge Institute of Technology. Sweden,2009.

[94] M. Shirole and R. Kumar, "UML behavioral model based test case generation: a survey," ACM SIGSOFT Software Engineering Notes, vol. 38, pp. 1-13, 2013.

[95] F. Elberzhager, A. Rosbach, J. Munch, and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," Inf. Softw. Technol., vol. 54, pp. 1092-1106, 2012.

[96] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," Softw. Test. Verif. Reliab., vol. 22, pp. 67-120, 2012.

[97] U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review," Information and Software Technology, vol. 56, pp. 1219-1232, 10// 2014.

[98] http://pages.cpsc.ucalgary.ca/~absharma/

TEZ FOTOKOPİSİ İZİN FORMU

ENSTİTÜ

Fen Bilimleri Enstitüsü	☐
Sosyal Bilimler Enstitüsü	☐
Uygulamalı Matematik Enstitüsü	☐
Enformatik Enstitüsü	☑
Deniz Bilimleri Enstitüsü	☐

YAZARIN

Soyadı	: AYDIN
Adı	: Muzaffer
Bölümü	: Bilişim Sistemleri

TEZİN ADI (İngilizce) : GUI Testing of Android Applications: A Systematic Mapping

TEZİN TÜRÜ : Yüksek Lisans ☑ Doktora ☐

1. Tezimin tamamından kaynak gösterilmek şartıyla fotokopi alınabilir. ☑
2. Tezimin içindekiler sayfası, özet, indeks sayfalarından ve/veya bir bölümünden kaynak gösterilmek şartıyla fotokopi alınabilir. ☐
3. Tezimden bir (1) yıl süreyle fotokopi alınamaz. ☐

TEZİN KÜTÜPHANEYE TESLİM TARİHİ : 06.01.2015