UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Triin Samuel

Problems and solutions in mobile application testing

Master's Thesis (30 ECTS)

Supervisor: Dietmar Alfred Paul Kurt Pfahl

Tartu 2016

Problems and solutions in mobile application testing

Abstract:

In recent years the amount of scientific papers published on the topic of mobile applications has significantly increased. However, few researchers test their assumptions and solutions in industry. This thesis aims to provide an overview of what current scientific literature considers problems and potential solutions in mobile application testing, and compare it to opinions of industry professionals. A literature review is performed to extract the list of problems and potential solutions, after which representatives of six Estonian companies involved in the field are interviewed to verify whether the problems and solutions proposed in the literature are relevant for industry. The study reveals that while the relevance of each problem is highly variable from one company to another, there are some key problems that are generally considered vital both by research and industry. However, the solution concepts proposed by scientific literature are often too theoretical, general or outdated to be of much interest to industry professionals.

Keywords:

Mobile applications, testing, quality assurance, fault detection, smartphones, tablets, mobile computing, literature review, interview, questionnaire

Probleemid ja lahendused mobiilirakenduste testimises

CERCS: P170

Lühikokkuvõte:

Mobiilirakenduste testimise alaste teadusartiklite arv on viimastel aastatel visalt suurenenud. Samas testivad vähesed mobiilirakendustega tegelevad teadlased oma oletusi ja lahendusi firmades. Selle lõputöö eesmärgiks on pakkuda ülevaade teaduskirjanduses mainitud mobiilirakenduste testimisega seotud probleemidest ja potentsiaalsetest lahendustest ning kõrvutada seda alal igapäevaselt tegutsevate professionaalide arvamusega. Kõigepealt viiakse selle töö käigus läbi teaduskirjanduse uuring probleemide ja potentsiaalsete lahenduste väljaselgitamiseks, misjärel intervjueeritakse kuue mobiilirakenduste testimisega tegeleva firma esindajaid, et välja selgitada, kas kirjalduses esile toodud probleemid on olulised ka tööstuses. Intervjuude tulemusena selgus, et kuigi firmad hindavad probleemide tähtsust väga erinevalt, on siiski olemas mõned võtmeprobleemid, mida peetakse oluliseks nii teadues kui ka tööstuses. Samas on teaduskirjanduses pakutud lahendused tihti liiga teoreetilised, üldised või vananenud, et firmade esindajatele huvi pakkuda.

Võtmesõnad:

Veatuvastus, testimine, tarkvara kvaliteet, nutitelefonid, tahvelarvutid, nutiseadmed, intervjuu, küsimustik, mobiilirakendused, nutirakendused

CERCS: P170

Table of Contents

1 Introduction

In the recent years, mobile devices have grown from futile entertainment gadgets to popular and ever-present media with a wide range of uses from social applications to business, medicine and others. This has brought the importance of testing mobile applications into highlight. As mentioned by various researchers [1, 2, 3, 4] mobile applications have some unique qualities that demand new or modified testing approaches to ensure effectiveness and efficiency. Accordingly, the number of scientific papers written about mobile application testing is steadily increasing. However, upon inspection of these papers one can see that the proposed methods are usually validated by the researchers themselves in a controlled environment, on few applications. Therefore, it is uncertain whether the proposed solutions are usable in industry and whether the problems mentioned in literature are actually relevant in real mobile application development and testing. In order to find answers to these questions, I decided to carry out a literature survey and then interview companies to assess the practical relevance of the information collected from literature.

The exact research questions are as follows:

- RQ1: What are the problems specific to testing of mobile applications as opposed to conventional applications, according to scientific literature?
- RQ2: What are the solutions (methods, tools) proposed by literature, if any?
- RQ3: According to literature, to what extent are these methods and tools used in industry?
- RQ4: Are the problems described in literature considered relevant by industry professionals?
- RQ5: Do industry professionals consider the solutions proposed in literature promising?

In addition to answering these research questions, the contributions of this thesis are:

- A list of mobile application testing problems extracted from scientific literature, with the relevance of each problem assessed by six Estonian companies active in mobile application testing.
- A list of solutions proposed in scientific papers, some of which with industry comments.
- A mapping between the problems and solutions.

This thesis consists of 7 chapters. Following an introduction and a brief overview of the topic, the methodology is introduced in Chapter 3. Chapter 4 covers the results of the literature survey and presents answers to research questions RQ1 to RQ3. More specifically, Chapter 4 aims to determine what scientific literature considers problems in mobile application testing, which solutions are proposed and how much these solutions are used in industry. In Chapter 5, the results of interviews conducted with six companies are presented. The purpose of the interviews is to evaluate how relevant the industry considers the problems and solutions mentioned in scientific literature. This addresses research questions RQ4 and RQ5, respectively. The thesis concludes with a summary and the list of references.

2 Background

The first device that could be considered a smartphone was IBM Simon [5] released in 1994. It had a touchscreen and enabled users to send e-mails, use maps and read news. While new and more advanced smartphones were developed and distributed starting from that point, smartphones as we know them now started gaining mainstream popularity only in year 2007 when Apple "reinvented the phone" by releasing the first iPhone [6].

Since then, smartphone sales have skyrocketed [7]. What initially were thought to be just enhanced phones and entertainment devices have now developed into a wide range of different devices capable of performing business tasks, simplifying everyday life and enabling users to be continually connected to their work, social circles and service providers [1, 8]. Mobile devices are challenging conventional computers [9]. Consequently, the criticality of mobile applications has significantly increased [1, 3]. This has forced developers to focus more on the quality of their applications and look for effective testing techniques.

Testing of mobile applications incorporates many of the problems inherent to software testing in general. However, mobile devices also have qualities that differentiate them from conventional computers and therefore create testing challenges that are either unique to or more relevant in the case of mobile applications.

According to Muccini et al [1], the most important distinctive characteristics of mobile applications are mobility and context-awareness. Mobility means that the application is designed to run on a moving device, like mobile phones and mp3 players. This requires the device to be fairly autonomous, at least energy-wise and brings additional restrictions. Context-awareness is the ability of the application to detect changes in the environment it runs in and react to them. These changes can range from changes in location or available resources to surrounding objects and tasks that the user is currently performing.

As displayed on Figure 1, the three dominating mobile operating systems (OS) are Android, iOS and Windows Phone. According to netmarketshare [10], Android was the most popular OS in the first quarter of 2016 with a 60% market share. iOS followed with 32%. Windows Phone was third with 3%, followed by Java ME having 2%.

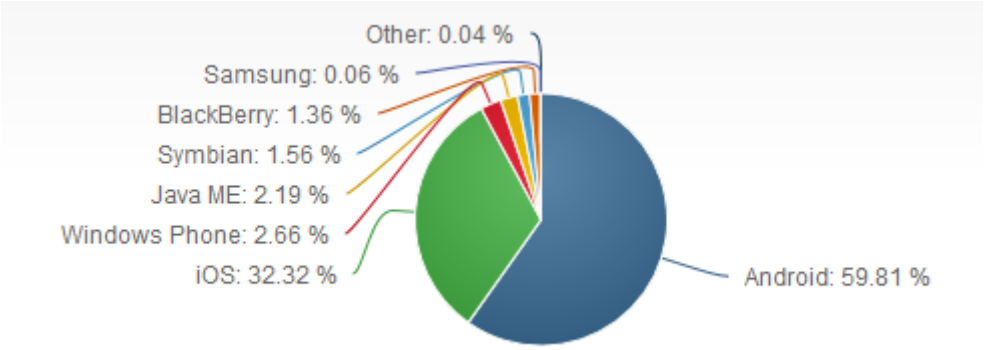

Figure 1: Mobile/Tablet operating system market share January, 2016 to March, 2016 [10].

Android is a free open-source operating system based on the Linux kernel. It is owned by Google and was released in 2008. Android applications are normally developed in Java, compiled to Java bytecode and then to Dalvik bytecode to be run on Dalvik virtual machine (DVM), with most of the code interpreted during runtime. From version 5.0, DVM

has been replaced by Android Runtime (ART) [11] that compiles the application to machine code during installation. Therefore, even though Android applications are commonly developed in Java, they cannot be run on Java Virtual Machine. Android applications mostly consist of Activities that communicate to each other via messages called Intents.

The second most popular operating system is iOS, a proprietary, closed source operating system released by Apple in 2007. The iOS operating system can be used only on Apple devices. This reduces its market but ensures better hardware-OS compatibility due to a smaller number of different devices. Applications for iOS are normally developed either in Swift or Objective-C. The core of iOS is based on Darwin, a Unix operating system also used for Apple OS X, and Cocoa Touch is used for the user interface.

Windows Phone (previously Windows Mobile, now Windows 10 Mobile) is a proprietary closed-source operating system developed by Microsoft and released in 2010. Applications for Windows mobile devices can be developed in various languages like C#, .NET, C++ and HTML5. The latest mobile operating system from Microsoft was released as Windows 10 Mobile, reflecting Microsoft's intention to essentially merge the desktop and mobile versions of Windows [12] so that same apps could be run on both of them.

3 Methodology

In this chapter I describe how I found relevant scientific articles, extracted information from them and set up interviews with industry professionals.

3.1 Methodology of literature survey

Finding relevant literature

In order to get familiar with the available material, I first conducted an informal search in the ACM Digital Library database as one of the most relevant databases in the field of computer science. I searched for articles related to problems in mobile application testing published year 2007 or later because that was the year when the first iPhone, as well as the first alpha version of Android was released. The first Windows Phone was released later, in 2010. Therefore, 2007 was the year mobile applications as we know them now started gaining mainstream popularity. Since I did not use any additional filtering, I got an excessive amount of results which I ordered based on relevance. I skimmed through the most relevant search results and manually chose 26 articles that seemed relevant to the question in hand by title.

Through reading the initial papers, I learned some additional keywords and search criteria that could be used. I also noticed that most of the results were conference papers and papers that mentioned problems usually also discussed solutions to them. Therefore I conducted a second, more formal and structured search for journal articles. Since some relevant papers might not be indexed by ACM digital library, I conducted the second search in 4 different databases.

Databases:

- ACM Digital library
- SpringerLink (Computer science)
- Scopus (Computer science)
- ScienceDirect (Computer science)

Search criteria:

- Only journal articles
- Published 2007 or later
- Full-text is available in the database

Through filtering and manual inspection I removed articles that matched any of the following exclusion criteria:

- Papers that were mainly about hardware-related, low-level communication or network issues, as opposed to end-user mobile applications
- Papers that my university doesn't have full access to
- Articles that do not analyse or make new contributions to the testing process itself. For example, if the paper was about developing a non-testing-related mobile application and at the end it was tested just to prove that the application works, then the article is not really about testing, even though it features it.

- Papers that are about mobile web application testing. Since web applications run in a browser or in a browser-like program, they don't inherit many of the challenges that native mobile applications have and are often more similar to web applications meant for desktop devices than to native mobile applications [13].
- Testing techniques that are not meant for consumer-oriented mobile applications.

The exact queries are listed in Appendix I. The second search yielded 374 results, 355 of which were unique. Out of these, 84 were left after manual filtering based on the title.

Therefore, the total set of abstracts to read was 26 + 84 = 110.

Based on abstracts, 57 papers were discarded, which results in a set of 110 - 57 = 53 papers to read. While reading I discarded two more papers because they had low relevance and one because it was superseded by one of the other papers in the set. This leaves the total number of papers included at 50.

Since Android was significantly more represented than other platforms in the set of found papers, many of the found challenges and solutions mainly concern the Android platform.

Extracting problems and solutions

For each of the papers that passed filtering, I highlighted all relevant parts while reading and marked which research question they concern. If a solution was proposed in the paper, I assigned an approximate category to it and wrote the most important keywords concerning the solution to the front page. After reading all of the papers, I went through all the highlighted parts concerning RQ1 and wrote out all the found problems. Researchers rarely used the word 'problem', but often highlighted 'challenges' to justify the necessity of the solution they were going to propose. Offering a solution clearly shows that they considered the 'challenge' something that needed to be solved, so I counted these as problems. Some problems were also collected from general discussion parts of the papers.

After extracting a list of problems, I went through the papers again to write summaries of the proposed solutions (RQ2). The solutions were based mostly on the highlighted parts and the keywords I had written on the papers while reading, but details often needed to be clarified from other parts of the paper.

Producing a problem-solution matrix

Explaining every solution proposed in literature to each industry professional would have resulted in unrealistically long interviews and exhausted interviewees. Therefore, I needed to restrict the set of solutions that I was going to introduce to each industry professional. Moreover, I wanted to make the interview beneficial to the interviewees. Therefore, I decided to only introduce solutions to problems that the specific interviewee considered relevant. Since I did not have any information about the perceived relevance of each problem prior to the interview, it was not possible to choose the set of solutions to explain beforehand. I needed a mapping of problems and solutions that I could use during the interview to choose which solutions to explain.

In order to find out which problems a given solution solves, I used my general knowledge of the solutions that I had gained from reading the papers, as well as the challenges that researchers presented as justifications for their solution. For each problem-solution combination there were 4 options:

- 'Y' - the proposed solution significantly contributes to solving the given problem
- 'Partly' - partly solves the problem

- ‘Maybe’ - might be useful, but more information is needed to know
- Blank – the proposed solution does not address this problem

In the first interviews I learned that theoretical papers were not of interest to the industry professionals, especially considering the limited time for each interview. Consequently, I slightly modified the matrix by marking columns containing very theoretical or general solutions with grey background colour. This resulted in the matrix proposed in Section 4.4.

3.2 Methodology of case study

Selection of industry professionals

I compiled a set of 23 potentially useful companies based on a Google search and my general knowledge. Then I explored web sites of the companies to select ones that:

- Operate in Estonia
- Deal with testing of native mobile applications. If a company develops native mobile applications, then testing is implied unless the home page hints that it is outsourced
- Are not a one-person company
- Seem professional enough to pay attention to the testing process

This restricted the list to 7 companies, which I contacted. Five of the contacted companies replied and were willing to participate. In addition to these, one of the chosen companies put me in contact with a very suitable, but less known company that I wasn’t aware of, which also agreed to participate. This resulted in a total of 6 companies to interview.

I asked to interview someone involved in testing native mobile applications. In two cases I used pre-existing in-company contacts to find a suitable person in the company to interview.

The interview process

The interview structure was as follows:

1) I introduced my research topic and the interview procedure, after which I collected some general information about the company. This information included number of employees involved in testing mobile applications, whether the company is oriented at testing or development, mobile platforms the company works with and experience with using or developing automated solutions for mobile application testing. In addition to this, before showing the list of problems acquired from literature, I asked whether the interviewee sees any notable challenges in mobile application testing.
2) I presented the list of testing problems found from literature and asked the interviewee to rate the relevance of each problem in actual mobile application testing. The answers were given on a multiple choice scale that also included options for “N/A” and “Already solved”. Small changes were made to the questionnaire after the first interview to improve ease of understanding. The final questionnaire can be found in Appendix II.

3) I looked at which problems the interviewee considered important (marked as "Definitely") and used the Problem-Solution mapping presented in Section 4.4 to extract the set of corresponding solutions proposed in literature. Thereafter I introduced some of these solution ideas to the professional and asked feedback on each of the explained solutions. Since the respondents were only interested in practically applicable solutions and time was scarce, I omitted articles that were very general or theoretical from the explanations. These are marked as grey in the problem-solution matrix. Some solutions were not explained due to time constraints.

The time planned for each interview was 1.5 hours. The first part took about 10 minutes while the duration of the second part was dependent on how fast the interviewee filled out the questionnaire, averaging at about 30 minutes. Duration of the third part was affected by how many problems the interviewee considered relevant in the questionnaire. Two respondents filled out the questionnaire faster than intended, which resulted in shorter interviews. One interview was with the permission of interviewee extended to 2h because there were many potentially relevant solutions and the interviewee had a slightly different IT background than myself and other respondents, so I had to adjust my explanations.

Participating companies

Fob Solutions

Fob Solutions is a mobile-oriented quality assurance company that on the side also provides development of web and native mobile applications. Fob Solutions has about 20 testers and some developers who work with Android, iOS and Windows Phone. I talked to the head of quality assurance.

Testlio

Testlio is an Estonian company that provides a community-based testing service. This means that Testlio manages the testing process and prepares everything necessary, but actual testing is performed by a network of approximately 200 freelance testers who are not employees of Testlio. Testlio works with Android, iOS, Windows Phone and to a lesser degree BlackBerry. Since testing is performed manually in Testlio and the company doesn't diagnose the found problems, questions related to test automation, device emulation and fault diagnosis were not applicable. The company does have its own platform to facilitate testing, but it mostly has management functionalities, not test running or generation. I interviewed a QA manager that I knew prior to the interview.

TestDevLab

TestDevLab is a Latvian quality assurance company that in addition to the more common testing services also provides battery, penetration and data usage testing. About 50 people are involved in Android, iOS and Windows Phone applications testing in TestDevLab. Even though the company officially resides in Latvia, it is common for their employees to temporarily move to where the client is. Therefore, I got a chance to talk to one of their QA engineers that lives in Estonia. TestDevLab QA engineers are not oriented to a certain platform, therefore my interviewee had worked with different platforms (web, iOS, Android) in different projects. TestDevLab is the author of a test automation tool called Apimation[1].

[1] https://apimation.com

Wazombi

Wazombi is an Estonian company focused on providing end-to-end solutions where everything from electrical engineering to UI design is done in one house. Since they are more oriented on development, they have one person specifically oriented at mobile application testing, whom I interviewed. Wazombi works with Android and iOS, but as learned from the interview, most of their Android applications are not Java-based. Instead, Xamarin and C# are used. Xamarin also constitutes the only test generation tool mentioned by case study participants.

Mooncascade

Mooncascade is an Estonian company that mainly provides mobile, responsive web and back-end development. From mobile platforms, Android, iOS and Windows Phone are used. There are four people working at mobile application testing. Some testing frameworks like Appium and Selendroid are used for test running. I interviewed the lead of the quality assurance team.

Mobi Lab

Mobi Lab is a mobile application design and development company, formerly a part of current parent company Mobi Solutions. They work with Android, iOS and Windows Phone. I interviewed the only dedicated tester, but developers are also responsible for testing the applications that they are making.

4 Results from literature survey

The results from the literature survey will answer RQ1-RQ3:

- RQ1: What are the problems specific to testing of mobile applications as opposed to conventional applications, according to scientific literature?
- RQ2: What are the solutions (methods, tools) proposed by literature, if any?
- RQ3: According to literature, to what extent are these methods and tools used in industry?

4.1 Problems in mobile application testing

In this section I will give an overview of problems and challenges that are specific to or especially relevant in the testing of mobile applications. I will do so by analysing scientific articles on the topic. Firstly, I will describe the process of finding relevant articles, after which I will present the found problems. This section will answer research question number 1: 'What are the problems specific to testing of mobile applications as opposed to conventional applications, according to scientific literature?'

Challenges specific to mobile application testing stem from the peculiarities of the domain. Therefore, I have grouped challenges according to their core causes. In reality each problem can have more than one cause, so the grouping below should be taken as an approximation made in an effort to simplify reading.

Fragmentation

There is a large variety of platforms [14, 8], operating system versions, hardware [15, 16, 17, 18, 19, 20] and screen sizes [1, 21, 20]. Testing the different combinations is important because applications behave differently not only on devices from different manufacturers, but also on devices from the same manufacturer [18]. 86% of Android developers think that fragmentation is a serious problem [18].

P1: Due to fragmentation, compatibility testing needs to be done on numerous different devices, which takes lots of time, effort [16, 18, 22, 21, 20], money [16, 23] and is difficult to automate [16, 3].

P2: It is difficult to programmatically test whether the application is rendered correctly on different (numerous) devices [1]. Even if all the required user interface elements are on screen, layouts can still differ based on OS version, screen size and orientation.

P3: Automated scripting of tests needs to be abstracted away from the device to be of any real use [21]. Even more so if the tests are meant to be used on different platforms [24].

P4: Since mobile phones are diverse in shapes and operating systems, it is challenging to enable natural interaction with the device when performing usability testing [25, 26]. In other words, to produce genuine results in usability testing, the user has to be able to use the same model of device, with the same input systems (scroll, wheel, custom menu buttons and styluses) and operating system that they are used to. Therefore, all of them have to be supported for a usability study with a wide scope.

P5: Performance variations across devices are large [21], making it difficult to optimize performance while ensuring that the application still works on all targeted devices.

P6: Mobile testing requires a relatively large set of test devices that is expensive [20] and must be kept up-to-date [24, 20].

P7: Testing all the different device configurations on emulators would require more computing power than most developers have [20], while testing in cloud is expensive [20].

External software dependencies

P8: Due to fragmentation, bugs in and interoperability problems between layers like application, application framework, operating system and hardware are frequent [1] and make it difficult to determine whether the fault is in the application being developed or on a lower level. It is not uncommon for apparent application bugs to actually be caused by faults in the operating system [1]. Also, since Android relies on framework libraries to guide the execution of the app, Android applications are prone to 'path-divergence problem'. In other words, since Activities inside an Android application are linked to each other by an outside party, it is normal for a test value to cross application boundaries and therefore difficult to eliminate outside impact while testing [27].

P9: In order to write tests for an application, the tester needs to have a mental representation of the software under test. This is not easy as systems nowadays are typically complex and tightly coupled with environment. Therefore, testers need to know both the software and its environment well. [28]

P10: An environment model has to be created in order to simulate external dependencies during testing, but this requires expert domain knowledge and doing it manually is tedious and time-consuming. There is no mobile-applications-specific tool for it [29].

Frequent external communication

P11: Inputs from lots of different sources (users, sensors, connectivity devices) have to be considered [1, 14, 17, 9].

P12: It is not viable to control the state or behaviour of external dependencies during testing [27, 22, 30, 31] or analysis [29, 32] of the application. The Android system or another app can send an event to the concerned app anytime and in smartphones these events are much more frequent than in traditional software systems [27]. This makes it difficult to build models, validate test results and diagnose bugs that appear only under certain external conditions [33, 17].

P13: Current emulators are unable to properly simulate the sensors, GPS, connectivity and device-based limitations of real devices [1, 34, 22, 21].

P14: Existing testing approaches consider mobile applications in isolation, but in reality inter-application communication via intents and content providers is common [1]. Since this is often not considered, poor validation of incoming inter-application messages (Intents) often goes undetected. This enables malicious applications to access sensitive user data and perform operations they don't have permissions for, causing security problems [35].

P15: Due to time constraints and the high number factors that affect application behaviour, testing is often focused on expected behaviour while testing for unexpected events gets little attention. As a result, applications are prone to failure from unexpected events [36].

P16: In order to properly validate pervasive applications, they need to be debugged in distributed mode and heterogeneous environment [19, 30, 26, 9].

P17: External resources are often unreliable and have transient failures that are difficult to discover [37, 9].

P18: Testing exception-handling code related to external resources requires the possibility to put the external resource into a prescribed state [37].

P19: Many exceptions related to external resources only occur after very specific action sequences, making them unlikely to be discovered by unsystematic testing [37].

Variable user and usage context

P20: There are lots of users and they are diverse, so different user profiles should be tested [38, 39].

P21: Usability testing can be difficult if the experience level of users is very variable or very different from the standard. For example, most elderly people interact to mobile applications completely differently from younger people [40]. They also tend to require more detailed instructions and be afraid of breaking the system or appearing unintelligent (18). These differences make it more difficult for a tester to assess the usability themselves, as well as to conduct a proper experiment. Given the wide range of smartphone users, this is especially relevant to mobile applications.

P22: Creating realistic testing scenarios and prototypes is difficult because the devices are used in different places, situations, with different settings [19, 26, 41, 42]. This also implies that context-aware applications cannot be properly evaluated in laboratory conditions [43].

P23: Some user scenarios are difficult to script, for example the physical interaction required to realistically test a sports app [24].

P24: In real environment there are many dependencies that do not appear in lab testing [33, 41].

P25: In addition to application behaviour, user actions and environmental changes should also be logged [19]. The whole usage context (physical location, multitasking, goal, environmental changes) of the application might have an effect on application usage [44, 19], but cannot be automatically captured [41]. User actions and environmental changes should also be recorded in addition to application behaviour. Current mobile prototyping techniques don't save any information collected during testing for later usage.

P26: Social applications are only meaningful within a true social context. Therefore, a social context has to be created for testing them, which is difficult to achieve in a laboratory or with few users [43, 31].

P27: The traditional task-based field testing approach might not apply in mobile applications domain because users' interactions with mobile applications are not necessarily task-based [41].

Fast evolution

P28: Methods of human-device interaction are constantly changing (physical keyboard, touch screen, normal tap, multi-touch, slide), making it difficult to simulate these actions and therefore automate testing [14, 30].

P29: Application requirements are constantly changing due to frequent integration with new applications [45] and technologies [16], changes in market trends [16, 20] as well as changing user and environment requirements [19]

P30: New mobile technologies often lack reliability due to short lead time [44, 9].

Limited resources

P31: Difficult to monitor the large number of dependencies without significantly impacting resource usage [33, 22].

P32: As a result of the previous problem, built-in failure logging of mobile operating systems is insufficient. Mobile OS-s do log information about failures, but the data collected is so limited that it's often almost useless for finding sources of problems [33].

P33: Due to limited resources, system response time to touch depends on resource utilization a.k.a. how many other applications are currently running [1].

P34: Due to screen size limitations, input mechanisms for mobile devices are more complex and rarely offer alternatives [44].

Novelty

P35: Few tools for verifying mobile applications [46, 30, 47, 48, 39].

P36: Some non-functional properties, like usability [19, 30, 44] and accessibility [44, 49] are difficult to measure because they are very vaguely defined for mobile applications [19, 30]. Published material on the evaluation methods of these qualities in mobile applications is scarce and there is no consensus on which measures should be used for evaluation [30].

P37: Performance testing is often late due to lack of efficient methods for performance testing [34] and emulator-based development – during development emulators are used in place of real devices to simplify testing. Since emulators don't behave exactly the same way as real devices do (See P13), performance testing on them is rare and therefore performance problems are often discovered at the end of production cycle when testing on real devices. Fixing faults that late is difficult and time-consuming [34].

P38: There is no quality framework or a set of design principles for mobile applications, like there are for web [39, 49]. This is especially relevant for testing the non-functional qualities of user interfaces meant for older people [39].

P39: There are no wide-spread principles about how to conduct usability field studies in the mobile application domain [41]. It is not known how to prioritize different user-centered design evaluation criteria in case of mobile applications [19, 30].The mobile applications domain in general lacks clear best practices [9, 50].

Limitations related to platform implementation

P40: Even though Android applications are developed in Java, they are compiled to a special format called Dalvik bytecode that can be run on Dalvik Virtual Machine (DVM). In order run Android applications on Java Virtual Machine (JVM) and use testing tools meant for Java, the applications need to be transformed into Java bytecode [27, 46, 21, 37]. In newer Android versions DVM is replaced with Android Runtime (ART) [11], but the problem still remains. Android applications are also heavily dependent on a proprietary set of (Android) libraries that are not available outside the device or emulator [27]. Therefore, typically either an Android device or an emulator has to be used for testing.

P41: Running tests on Dalvik VM is slow (applies to Android) [46].

P42: In mobile apps it is not trivial to determine when a page has finished loading [17, 51].

P43: It is difficult to perform A/B and multivariate testing of native mobile applications because the application cannot be changed after installing [52].

P44: Constructs specific to mobile application languages have to be taken into account when producing data or control flow graphs and assessing test coverage [1, 22].

P45: The only reliable way to perform a clean restart of a mobile application is to remove and reinstall it, which significantly increases test duration [53].

P46: Capturing screen content and user-device interactions during field testing is difficult because on Android UI interactions cannot be captured automatically [30, 54].

P47: Since the environment is complex and developers don't have low-level access to OS, it is difficult to identify all the factors on which the application's behaviour depends and conditions under which a bug appears [33].

Others

P48: Due to the expected short development cycle and cheaper cost of mobile applications, speed of testing is more important than in the case of desktop applications [14, 9].

P49: Due to the constraints of mobile devices and platforms, using general-purpose software development methodologies and testing practices might not be viable in the mobile applications domain. Therefore domain-specific adaptions to the general processes are needed [9].

4.2 Proposed solutions

In this section I will answer research question 2: "What are the solutions (methods, tools) proposed by literature, if any?"

I will do so by describing the tools and methods proposed in literature for solving the problems described in the previous section. The solutions are divided into loose groups based on where they could be useful.

Theoretical

S1: 3 principles for mobile application testing [4] by Santos and Correia:

1) Use both emulators and real devices. The former is more cost-effective while the latter is necessary non-functional testing.
2) Automate as much as possible
3) Set up a lightweight testing strategy

S2: Survey of current research in designing and evaluating pervasive applications [19]**.** Tang, Yu et al. introduce the challenges in developing pervasive applications, highlight principles and techniques for prototyping, review available prototyping and testing tools and propose open research topics.

Their 3 principles for creating prototypes are:

1) Prototypes should be constructed early and fast
2) Every prototype should have a clear purpose
3) Prototypes should only contain elements that are strictly necessary for the defined purpose

All in all, this paper gives a very good overview of the topic.

S3: Methodological aspects of usability testing [25]. Bastien performed a review of works that aim at clarifying test procedures or developing tools for conducting user tests. More specifically, he focused on topics that would be relevant for the health care and medical field. In part of his work, he also looked into testing mobile applications. In addition to highlighting some problems, he underlined that mobile applications cannot by definition be reliably verified in a laboratory because of their inherent mobile nature. In addition to that, he noted that diary studies are a type of user testing that are potentially useful in user testing of mobile applications, but have not really been leveraged.

S4: Model-based vs exploratory testing [45]. Nascimento and Machado compare and evaluate model-based and exploratory testing in the context of feature testing. They conduct a small 2-feature case study and come to the conclusion that exploratory testing requires less effort and enables learning about the application under test while model-based testing has better support for test re-execution at the cost of higher initial effort. Therefore, they propose that the best solution would be to first apply exploratory testing and later use the information gained from it as an input for model-based testing.

S5: Using prototypes in game development [31]. Koivisto and Suomela discuss using prototypes for testing pervasive games. For context-aware games, prototyping in early stages is important because for applications, the real-world environment significantly affects gameplay. They creating a software prototype of the core functionality of the game should be done early in the design process and preferred to paper prototypes, especially if the game is very dependent on sensor inputs. They also noted that ideally people unfamiliar with the game should be included in testing, but it is acceptable to use only team members in some tests just because it is much faster.

S6: MTaaS infrastructure [55]. Gao et al propose a general infrastructure for Mobile Testing-as-a-Service. They present two testing cloud concepts: one consisting of real devices and the other of emulators. On these GUI-based functional tests, mobile quality of service tests and mobile feature tests can be performed. Since they only offer a general concept of what they consider a previously unexplored idea, models and potential features of such systems are described, but not implementation.

S7: Another cloudtesting solution [21]**.** Baride and Dutta introduce the general concept of cloud-based mobile application testing. They propose a centralized system that offers mobile testing as a service for multiple platforms, using both emulators and real devices. As the paper is relatively short and most of it deals with analysing challenges in mobile application testing and introducing cloud, the contribution of this paper remains very general and therefore probably low in practical value.

General tools and methods

S8: MobiBug [33]. Based on developer interviews and analysis of trouble tickets, Agarwal, Mahajan, Zheng and Bahl came to the conclusion that mobile applications are currently difficult to debug because current failure reporting gives very little information for diagnosing and reproducing problems. Based on their observations, they came up with a crowdtesting system called MobiBug that could be implemented by vendors of mobile operating systems. MobiBug has 3 main principles:

- **Spatial spreading**. Since making a device log all information that might be relevant for reproducing a bug would significantly impact its performance, it is wise to spread work across multiple devices. MobiBug divides the work between all devices that are using the given operating system so that each device measures only specific attributes chosen by the server based on device, operating system, failures that need to be diagnosed, etc. Data concerning undiagnosed failures is automatically uploaded via network or when the device is connected to a PC, depending on the criticality of the bug. If the problem is already diagnosed, uploading data is not necessary.
- **Statistical inference**. A probabilistic model is built in order to determine failure conditions and missing data is filled in using statistical inference. Developers can contribute to model building speed by providing information on known dependencies of their application. Based on this model the server chooses specific measures for each device to collect.

- **Adaptive sampling**. Server checks the model by occasionally sending queries about properties already learned by the model. If the results are not as expected, the model is refined according to new results.

S9: Crowdtesting framework iTest [15]. iTest is a cloud-based crowdsourced testing framework that enables developers to use the devices of registered end-users for testing applications on a large number of device-software combinations. In order to use it, the developer has to integrate the iTest development kit into their application and upload the application. When a registered tester with an iTest client app comes online, their username, location and technical parameters are uploaded, based on which the iTest server chooses applications to send to the tester. Tester selects a web service from a list, tests it and results are automatically uploaded to the iTest server where they become accessible for developers of that application. While the paper focuses on web service testing and web-based (e.g. not native) applications, an Android application is used as a prototype and the framework is general enough to be used on different platforms.

S10: Tool for symbolic execution of Android apps [27]**.** Mirzaei, Malek et al extended the Symbolic PathFinder Java tool to model Android libraries on JVM using stubs and mocks. The stubs are used to compile Android applications on JVM. They return random values for primitive types and empty instances in case of objects. Mocks avoid the path-divergence problem by simulating how the Android framework normally manages the application lifecycle, communication with other applications, etc. In addition, their approach is capable of generating drivers to simulate user inputs and sequences of events for automated testing. In contrast to the many GUI-based methods, Mirzaei, Malek et al. derive the model of application under test from source code using program analysis. According to authors, the method has a high code coverage, but no experiments to prove the claim were mentioned in the paper.

S11: JPF-Android, environment and library modelling [46, 29, 32]**.** Heila van der Merwe et al. have written multiple relevant papers about testing of Android applications.

Firstly, they developed **JPF-Android** [46]**,** an application verification tool based on Java Pathfinder (JPF). JPF-Android is a functional model of the Android framework that enables running Android applications on Java Virtual Machine as opposed to the usual slower Dalvik Virtual Machine while keeping the defect detection capabilities of JPF. In addition to porting JPF functionality to JVM, JPF-Android enables testers to script user and system inputs. Even though JPF-Android only models the core libraries, it can detect race conditions, deadlocks and different property violations.

In 2015 they turned to **simulating external dependencies**. More precisely, they explored the possibilities of using OCSEGen and Modgen for generating stubs that imitate the external environment and concluded that the slicing and side-effect analysis features of the tools could be useful.

Having found means to simulate the Android core classes and external dependencies that can result default values, they turned to imitating more complex dependencies. They **improved JPF-Android** [32] so that it uses parameters and return values from real method calls for simulating the environment during testing. As such, it can be used for testing complex applications that symbolic execution would fail to identify useful inputs for. Admittedly, in order to achieve good results with JPF-Android, it is necessary to have good code coverage during the inputs recording phase. However, JPF-Android is still useful as it is capable of performing very complex analyses and

therefore identifying problems that would pass other test systems. The tool is available on BitBucket[2].

S12: Execution points [48]**.** Aranha and Borba propose a system similar to function points for estimating manual testing effort and a model for predicting the number of execution points. The execution point value of a test depends on test size and test complexity. Each fundamental characteristic or action a test can include is first assessed by experts on an ordinal scale based on how much they affect testing effort and then execution point values are assigned to each ordinal value. If the test requirements are specified in standardized natural language, then the characteristics included in each test case can be automatically parsed and therefore after each characteristic has once been assigned a value by experts, test execution effort can be calculated in an automated way. Based on EP-s, the paper also proposes some models for further estimating test effort and capacity.

S13: Mobile testing framework used in practice [24]**.** Haller analyses the current state of mobile testing from a somewhat business-oriented perspective and describes the mobile testing framework employed in Swisscom. From reviewing the comments of 1000 app ratings in 3 app stores, he concluded that app stores are a useful way of getting user feedback for improving applications and that many of the apps in app stores do not employ basic testing techniques. He suggests paying attention to device compatibility, testing apps in their real context and proposes a simple framework for categorising apps in order to choose an appropriate testing strategy for each app.

Finally, he describes the automated testing framework deployed in Swisscom, built on Perfecto Mobile. Swisscom uses a keyword-driven test language that enables testers to write scripts in a keyword-driven language that abstracts away the specifics of different devices and platforms. The result is an HTML-report with screenshots from each device before and after every test step, enabling fast identification of problems that are difficult to programmatically detect. Since the approach proposed by Haller is already implemented and in use in a real software company, it has a lot of potential practical value for other companies concerned with mobile application testing.

GUI-based testing

S14: Test patterns for Android applications [50]**.** Morgado and Paiva extend the work of Moreira and Paivra on PBGT, a tool for pattern-based GUI testing of web applications. Even though the PBGT was not aimed at mobile applications, an experiment conducted on Android by Costa et al. [56] gave hope that it could also be useful for mobile applications if mobile-specific test strategies were developed. Therefore, Morgado and Paiva present the formal definitions of 3 patterns that can be used for UI testing of Android applications. The patterns are: Side drawer, Orientation and Resources dependency. The idea is that user interfaces that are similar in design can be seen as using the same design patterns and therefore should have a common testing pattern. Formalizing these testing strategies should make them more reusable and encourage formalization of additional testing patterns.

[2] http://heila.bitbucket.org/jpf-android

S15: Extension of MobileTest [14]. Zhifang, Bin and Xiaopeng extend MobileTest, a functional testing framework for mobile applications proposed by Jiang, Long and Gao [57]. Their approach uses an interruption and exception mechanism technique for control and enables concurrently controlling multiple devices. Various other techniques for test automation are also discussed, for example using image comparison and OCR for GUI-based evaluation. They show that their improved version of MobileTest is not yet efficient enough to be used in practice as currently using MobileTest costs even more than manual testing.

Record-and-replay

S16: VALERA [54]**.** Hu, Azim and Neamtiu propose a tool for recording and replaying test scenarios on real devices. VALERA instruments the Android framework and intercepts communications with user, network and sensors to record event sequences with exact timestamps so that appropriate delays could be used during replay. During replay, unscheduled events are also allowed to run to tackle the non-determinism described in P12. If some external event that VALERA cannot control (for example event from another application) happens during recording, but not at replay, execution continues after a specified timeout and the situation is logged so that the user can decide whether the missing event was relevant or not. The tool cannot guarantee deterministic execution of apps that don't use the Android UI toolkit, but can still replay them. Even though it records the whole externally visible state as opposed to just UI inputs, event schedule and low overhead both during record and replay (1.7% and 2.34% overhead, respectively) enable it to be very precise. VALERA doesn't record VM instructions or memory operations. Since it modifies already compiled code, it does not need access to source code and should work both on the just-in-time compilers used before Android 5.0 as well as new ART system used since Android 5.0. The tool is publicly available[3].

Model-based

S17: Evaluating model-based testing in the context of mobile applications [38]. Farto and Endo performed an experiment to determine whether model-based testing is a viable solution in case of mobile applications. In their experiment they gathered a group of 5 professionals and 10 undergraduate students that altogether developed 3 Event Sequence Graph models of the application under test in a limited timeframe. Then the 3 models were joined and researchers generated test cases from the models. They concluded that while model-based testing is usable on mobile applications, creating a model of the application requires expert knowledge and can be difficult. The fact that people involved in developing the application under test participated in modelling the application for model-based testing can also be considered a threat to the credibility of this experiment.

[3] http://spruce.cs.ucr.edu/valera/

S18: In **„Hybrid model-based testing for mobile applications“** [28], Haeng-Kon Kim introduces a Hybrid Model Based Testing (HMBT) tool capable of managing the complexity needed for performing model-based testing in the automotive industry and illustrates the working principles of this tool by using it to test a mobile application. First a state machine diagram is created by user of HMBT, then a set of feasible paths is extracted and the resulting concept lattice is trimmed using different algorithms to avoid state space explosion. They conclude that even though model-based testing requires skills and domain knowledge from testers, it can be useful, particularly for high-complexity systems where quality is important.

Model-learning

S19: Swifthand [53]**.** Choi, Necula and Sen propose a technique for automated GUI-based testing of mobile applications. The main advantage of their system over others is that while choosing test inputs it prefers ones that don't require restarting (in practice reinstalling) the application and therefore significantly decreases the test execution time. First a basic model of the application is generated using machine learning, then test inputs are derived from it such that each execution tries to visit unexplored states of the application and during execution the model is refined. To prune the search space, states are merged aggressively, which might result in over-generalization. If an inconsistency is discovered between the model and real application, the model is corrected using passive learning.

This approach ensures high coverage that is reached fast and does not require the initial model of the application to be very precise. A limitation of this tool/implementation is that according to the article it only runs on Android version 4.1 or higher. Since the tool instruments the Dalvik virtual machine, which is replaced with ART runtime from Android 5.0 [11], it might also not work on Android 5.0 and above. Additionally, SwiftHand cannot test apps that require internet connectivity. The tool is available on GitHub.[4].

S20: MobiGuitar [58]. Automated model-based GUI testing of mobile apps. The tool developed by Amalfitano et al first builds a state-machine model of the application's GUI by traversing the app and saving the list of actions that can be performed at each state. To decrease the size of the model, it treats two screens as equivalent if the objects in them have the same type properties and ID-s. Thereafter, JUnit test cases consisting of previously collected enabled events are generated, covering all edges of the GUI model. Last, the test cases are executed and reports are produced. According to authors, MobiGUITAR enables testers to choose input values for tests and generates better crash reports for debugging than some other popular tools like Monkey and Dynodroid. The tool is available at the AndroidRipper Github repository[5].

S21: A^{3}E [22] - a tool for systematic exploration of Android applications without requiring access to application source code. A^{3}E combines two different algorithms that have different goals.

The Targeted Exploration algorithm involves constructing an activity transition graph by static analysis and then quickly traversing the graph to cover all activities. The graph also includes activities that can be called from outside the application, therefore this approach achieves high activity coverage and achieves it fast with the help of the constructed model. However, activity coverage only measures the number of Activities

[4] https://github.com/wtchoi/SwiftHand

[5] https://github.com/reverse-unina/AndroidRipper/wiki

(screens) displayed, not how thoroughly these activities are tested. For that a second approach, Depth-First Exploration, is used.

In case of Depth-First Exploration, the tool extracts a list of visible GUI elements for each application entry point and interacts with the elements like a user would. First it chooses an element from the list and fires the corresponding event handler. If this results in a new Activity being displayed, the same algorithm is recursively applied there, backtracking if there are no more GUI elements to interact with. This takes more time and cannot test Activities exclusively available from outside, but ensures better method coverage for the Activities explored.

With the help of RERAN, the tool records all the steps it performs so that they can later be replayed and also used for bug reproduction in case the application crashes during testing. The authors evaluated the tool by organizing a 7-user experiment with 25 popular Android applications. In the experiment, users acquired only 30% activity coverage and 6% method coverage, both of which the tool significantly outperformed. Even though a library for user and sensor inputs was developed for this research, the tool still has the common limitations of not being able to test complex gestures, usage of external applications and native (C/C++) code. A^3E is open-source and available online.[6]

Search-based

S22: AGRippin [59]. A search-based tool for testing Android applications, by Amalfitano et al. AGRippin uses a combination of genetic algorithms and model learning by hill climbing where test cases are seen as chromosomes and actions on GUI interfaces as genes. At each iteration the algorithm searches for pairs of test cases that include the same or equivalent action A and swaps the post-A content of the two test cases. This is done only on a small portion of test cases at each iteration and results are propagated by fitness, resulting in an evolution-like strategy for achieving an efficient test suite with high effectiveness and source code coverage. Hill climbing is used for selecting test cases that cover some new source code at each iteration.

The technique is completely automated and is designed specifically for testing Android applications. Amalfitano et al. have previously also researched automated random testing [60] and developed model-based mobile testing tools Android Ripper [61] and Mobiguitar [58]. AGRippin was shown to have moderate advantage over its predecessor Android Ripper, based on tests with 5 Android apps. The executables are available on Github.[7]

Performance testing

S23: A tool for unit-testing performance [34]**.** Kim, Choi and Yoon propose an Eclipse plug-in for automated performance testing of mobile applications. The tool consists of a test case generator, test runner, prototype of performance predictor and a test result analyser. First, the test generator determines which methods need to be tested and generates test cases for measuring various properties, for example time from a user click to executing the resulting action. Developers add time limits to these methods. Then test runner executes the custom, PJUnit test cases and presents the results, where pass means that a method execution stayed within the limits specified by developers. After further development the performance predictor can be used to acquire approximate real device test results while testing on emulator.

[6] http://spruce.cs.ucr.edu/a3e/
[7] https://github.com/reverse-unina/agrippin

Reliability testing

S24: Thor [36] is a tool that tests robustness of Android apps. Thor doesn't generate test cases from scratch, but adds neutral event sequences to already composed tests. An event sequence is neutral if it should not affect the application state and therefore test result, for example Pause followed by Resume. The tool is light-weight, effective and relatively fast, but requires access to source code and since it requires already composed test cases, has to be used in combination with other testing methods. Source code and usage instructions can be found on Github.[8]

S25: VanarSena [17] is a cloud service for reliability testing of Windows Phone applications. Ravindranath, Nath, Padhye and Balakrishnan analysed 25 million Windows Phone crash reports to find the causes and constructed a system that detects common faults in applications. The developer only provides an application binary, which is then instrumented and multiple instances are run in parallel, using monkeys that simulate inputs from user and various sensors, as well as generate reports if the app crashes. The tool is fast, scalable and easy to use, therefore suitable for nightly integration testing. The tool was tested on 3000 published apps and found failures from a third of them, as well as plenty of previously unreported bugs. Considering that only 1.5 hours on average was spent on each app, VanarSena can be considered both effective and efficient.

S26: An approach for amplifying exception handling code [37]. Zhang and Elbaum propose an approach that exposes already written exception-handling code to more potentially problematic scenarios by mocking external resources and returning various expected and unexpected values to the application under test. The application is instrumented using AspectJ to record method calls to external resources and determine possible exceptions to throw. Test cases are duplicated to cover possible exceptions (up to a bound) and during execution incidents of abnormal termination and long execution time are caught. For each failure, a report containing information about the mocked API, mocking pattern, type of exception and call trace is automatically generated. The tool is capable of discovering and providing debugging information for complex and previously unreported problems that would otherwise be difficult to reproduce. The approach was shown to outperform CAR-Miner, another tool for detecting errors in exception-handling code.

[8] https://github.com/cs-au-dk/thor/wiki

Compatibility

S27: TESALIA [20]**.** TESALIA is a tool for modelling the cost and value of test cases in product line testing. Like a software product line, the Android platform has lots of different versions that build on each other, have different features and run on different hardware. Since testing all of the combinations is not realistic, some choice has to be made on which combinations to test. The different models and features can be described using a feature model that includes the test cost and value of each feature. TESALIA can automatically analyse the model to calculate the optimal set of configurations to test in order to achieve maximum value with bounded cost. First it derives all valid configurations, then prunes the tree using a cost function and finally prioritizes the tests so that the test cases with maximum added value can be executed first. It was theoretically shown that testing the configurations suggested by TESALIA is more efficient and achieves higher value than the current common practice of just testing on the 20 most sold devices. It is worth noting that TESALIA is a general solution where the testing cost and value of each feature can be set by user, so it is useful not only for handling the Android fragmentation problem, but for solving any similar cost-value optimization problem that can be mapped onto this model. The software is licenced under LGPLv3 and available on Github.[9]

S28: A tool for handling Android fragmentation [18]**.** Ham and Park try to tackle the Android fragmentation problem with a compatibility test system. According to them, OS fragmentation has been solved by Google, but device fragmentation, consisting of hardware and API fragmentation, continues to be a problem. Therefore, they proposed two methods to solve it. Firstly, code level test method searches for code that is not optimized according to target devices. For example, using absolute pixels instead of density independent pixels might result in a situation where for some devices the pixel coordinates specified in code do not exist in reality because the screen of that device is smaller. Secondly, the API level test method targets usage of API methods that might differ between devices and API levels. The advantage of this method over existing methods is that instead of installing the application under test on every device every time, there is a database that logs whether a specific API call works on a specific device and API version. As a result, most queries will be answered using the API Compatibility Knowledgebase without having to install the application on device. The authors demonstrated the effectiveness of the system by testing a small self-developed Android application, which leaves room for additional method validation.

S29: TestDroid [16] is an online platform for performing automated UI tests on real devices. It enables recording user interactions for running them later, random automatic exploration of UI without any human effort, as well as the possibility for developers to construct complicated test scripts manually. Since the devices are owned by TestDroid, it is not a crowdtesting solution, but rather a testing platform as a service. The service is available on the TestDroid website.[10]

[9] http://tesalia.github.io/
[10] http://testdroid.com/

Usability and user testing

S30: Living Lab [62]**.** Bergvall-Kåreborn and Larsson tried out a Living Lab approach for testing a public transportation mobile application by interviewing people at bus stops. Based on their experiment, they highlighted some challenges and lessons related to performing fields tests. Firstly, the real-life context of the experiment made it difficult to follow similar procedure with all participants or ensure the diversity of the sample group by anything other than visible characteristics. Secondly, in order to get any feedback on further development from users, they usually need to be first be provided some ideas as stimuli. Additionally, it is important to spend time in the actual application usage environment, be flexible about the testing process and make the testing as easy as possible for users by avoiding excess information and users having to figure things out on their own.

S31: Usability metrics model for mobile applications [30]**.** Hussain et al analysed 26 articles out of a set of 409 found journal articles concerned with human-computer interaction. The result is a set of GQM model measures for evaluating the usability of mobile applications, more precisely 17 objective and 19 subjective measures. An important property of this solution is that it is customizable and meant to evolve as technology does. They tested the measures by conducting user experiments with two different applications and devices, finding that some of the objective measures could not be collected due to technical reasons and using too many metrics at the time can make analysis of the results too complex.

S32: Multivariate testing of native mobile applications [52]**.** Holzmann and Hutflesz propose an aspect-oriented programming approach for A/B and multivariate testing of native mobile apps, without the need to redeploy the application for introducing changes. It works by modifying the build process so that packages with alternative views could be downloaded from server before execution of the application, a practice making the application potentially vulnerable to other injections. The application developer can design the different views as usual and only has to add annotations for user action logging to application code. In the future the annotations could potentially be added by an Eclipse plugin. Currently the solution is only implemented for Android applications, but the authors plan to implement it on other platforms as well.

S33: Toolkit for usability testing [26]**.** Ma, Yan et al propose a toolkit for collecting fine-grained GUI events during user testing to detect usability problems. The approach requires integrating a custom SDK and some changes to the application source code for logging the events. The collected events along with timestamps and properties of relevant windows are sent to a remote server in batches where they can be analysed either manually or automatically. Automatic analysis requires developers to record an optimal flow for a given task, which user data is compared to in order to detect usability problems. Based on an experiment with one application and 12 users, the tool can detect most of the problems that are detected by laboratory testing and some additional ones. An advantage of the tool over laboratory testing is that event sequences are recorded and can therefore be manually replayed. On the other hand, users' facial expressions during laboratory testing can make it easier to detect causes of detected usability problems. Even though the current solution was implemented for Android version up to 2.3, the authors are planning to develop a similar framework for newer Android versions and the approach should also be viable for other platforms.

S34: Accessibility and usability evaluation framework [44]**.** Billi, Burzagli et al propose a methodology for evaluating the accessibility and usability of mobile applications. The main principles of their methodology are early assessment and the usage of mobile-oriented methods. More specifically, they propose to first assess accessibility on a very early prototype, solve the found issues and then usability on a more complete one. Accessibility should be assessed by a set of users guided by a mentor and usability should be evaluated by experts based on a set of usability heuristics adapted to the mobile application domain. The paper focused on application accessibility for blind and visually impaired people, but should in principle be applicable for other disabilities.

S35: Gamification of field evaluations [43]**.** Rapp et al propose a methodology for evaluating social apps. The idea is that in order to properly evaluate social applications, the test users need to have some inner motivation for using the application. Therefore, they propose using gamification to motivate the user. This also has the additional benefits of engaging the test users faster and lessening the need for monetary compensation. Their findings were based on an actual successful field study with a food-related social application at a food fair.

Security testing

S36: Security testing of Android applications [35] Avancini and Ceccato propose a method for testing the security of communication among Android applications. On Android platform, applications are sandboxed by default and have to explicitly ask permission in order to access other resources. However, they can use Intents to send messages to and call other applications. If the application receiving an Intent doesn't validate the content of it, the sending application can essentially access any resources that are accessible to the receiving application. To avoid that, Avancini and Ceccato present an Eclipse plugin that integrates with Android Development Tools and automatically generates JUnit test cases to detect classes that don't properly validate data from incoming intents. They do so by making a copy of the application, but without permissions. Then, a potentially privacy-threatening intent is sent to both the original application and the duplicate without permissions. If this produces an error in the duplicate application, but not the original one, then we can conclude that a malicious intent reached a method call that needs special permissions and therefore the original application has a security problem.

S37: Another approach for inter-application security in Android [63]. Guo et al propose an automated solution for security testing of Android applications. First it tries to detect potential weaknesses by matching against predefined rules, then extracts parameters necessary for attack from the source and tries to perform the attack using Robotium test framework. The tool takes an .apk file as an input and uses reverse-engineering to extract manifest and source files from it. The effectiveness was successfully verified by testing 20 popular applications published on Google Play. In contrast with the solution proposed by Avancini and Ceccato that only detects Intent-based vulnerabilities, the approach by Guo et al works with both Intents and Content Providers.

S38: FlowDroid [23]**.** FlowDroid is a tool for detecting data leaks in Android applications, based on the Soot framework. It takes an apk file as input, unpacks it and parses the code, layout files and manifest to find lifecycle methods and callbacks, as well as calls related to data handling. From these a control-flow graph is generated for accurately representing the application lifecycle and traversed as the movement of data from sources to drains is observed. Since whole the path on the control-flow graph is being tracked, FlowDroid can report all the assignments that might have caused a data leak. The authors compared FlowDroid to two commercial tools, AppScan Source and Fortify SCA, and found that FlowDroid finds more leaks and also has fewer false positives. The open-source project is publicly available.[11]

S39: APSET [47]**.** APSET is a tool for detecting intent-based vulnerabilities in Android applications. The .apk file, vulnerability patterns constructed by experts and Android documentation are used to automatically generate partial component specifications and class diagrams. Then vulnerability properties and ioSTS (input/output Symbolic Transition Systems) test cases are constructed, the latter of which are later translated to JUnit test cases and executed either on an emulator or a real device. If source code is available, then it is analysed and used in creating XML- and SQL-injections. An XML-report including information about crashes, exceptions and non-compliance with the Android framework rules is returned as an output. The idea is that instead of defining vulnerabilities for each application, general vulnerability patterns can be used. The tool was tested on 50 popular applications from Android market (now Google Play) and 20 applications under development, out of which it managed to find vulnerabilities in 62 (88%), therefore it can be considered effective. The tool is publicly available on Github.[12]

4.3 To what extent are these solutions used in industry?

In this section I answer RQ3: "According to literature, to what extent are these methods and tools used in industry? "

Most of the solutions proposed in Section 4.2 were evaluated either on one or a few applications familiar to researchers or on a more representative set of applications acquired from app stores. However, in both cases the evaluation was performed by researchers themselves, usually in a controlled environment. Only one paper specifically claimed to discuss a solution used in a specific company, namely Klaus Haller who described the solution used in Swisscom (S13). There were also some tools that were evaluated on published apps (S16, S19, S21, S25, S37, S39, etc.) and one that was partly tested on apps currently under development in Oppenium (S39).

However, many authors were listed as associated with some company:

- S1 – Both authors list Microsoft e-mails
- S5 Using prototypes in game development – Both authors are associated with Nokia Research Center
- S6 – One of the authors is associated with Fujitsu Laboratories
- S8 MobiBug – All of the authors are listed as working for Microsoft Research
- S10, S11 – One of the authors is associated with NASA Ames Research Center
- S25 VanarSena – 3 of the 4 authors are associated with Microsoft Research

[11] https://blogs.uni-paderborn.de/sse/tools/flowdroid/
[12] https://github.com/statops/apset

- S29 TestDroid – The first author is one of the founders of the TestDroid testing platform available online.
- S30 Living Lab – One of the two authors is associated with Ericsson Research
- S35 Gamification of field evaluations – One of the authors is associated with Telecom Italia

Even though in these cases using the solution in industry was not mentioned in the paper, it is highly probable that some of them are used in the associated institutions. It is worth noting that most of them are big well-known companies and these findings might not reflect the situation in the rest of the industry. Small companies might well be significantly less involved with research, in both creation and usage.

4.4 Problem-solution matrix

The created problem-solution mapping can be seen on figures 2 and 3. For easier viewing, I have made the original file also available on Dropbox[13] (downloading is advisable).

The meanings of possible cell values are described in Section 3.1.

[13] https://www.dropbox.com/s/ia8vgjr7a8ppxkr/Problem-solution%20matrix.ods?dl=0

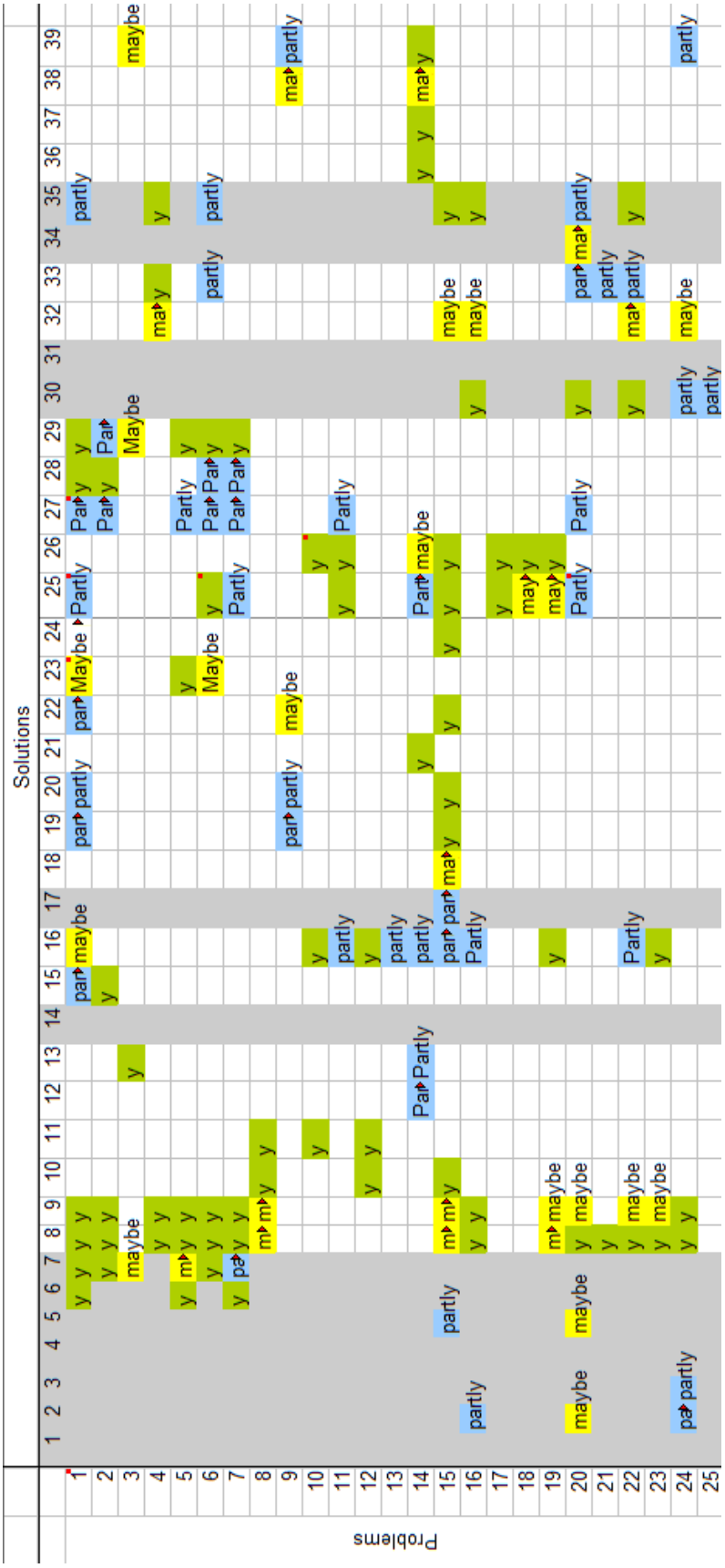

Figure 2: Problem-solution matrix: problems 1-25

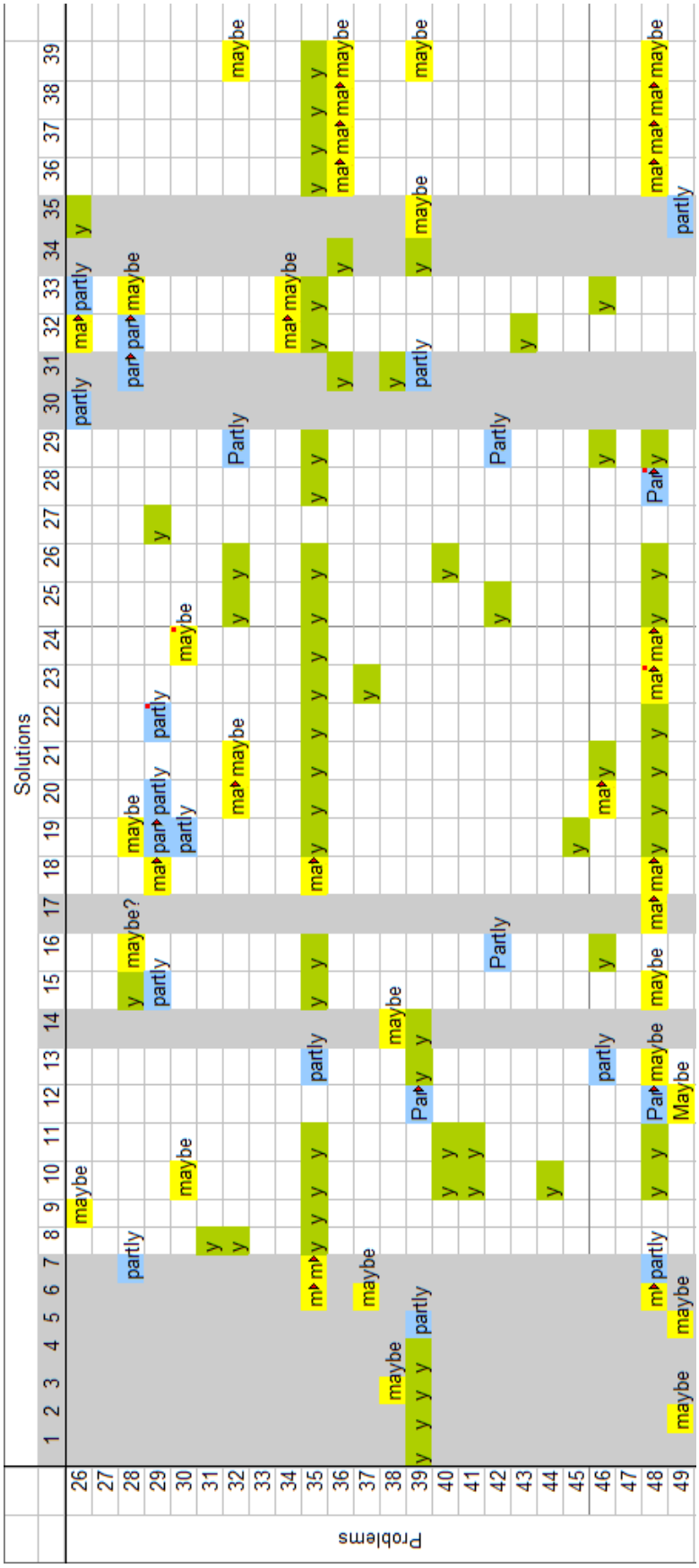

Figure 3: Problem-solution matrix: problems 26-49

4.5 Summary

In conclusion, 49 problems and 39 potential solutions were identified in the literature survey and listed. These lists answer research question 1 and 2, respectively. For research question 3, the result is less clear. Even though only one paper specified that the proposed solution is already used in industry, it is likely that some of the others are as well, considering that many authors were associated with companies active in the industry. Therefore, it can be said that the solutions are used in industry, but the extent of this usage cannot be adequately determined just based on scientific literature.

4.6 Limitations

This study was performed by one person in a limited amount of time. Since I had to be able to read and analyse all the included papers, doing a full formal literature study was not viable. Therefore, it is possible that some relevant papers were not found or were filtered out by the used search criteria. This is especially relevant for articles that did not include any of my chosen keywords in title or abstract. It is also worth mentioning that I was not previously familiar with the specifics of each scientific database.

Additionally, since the information was extracted from papers by just one person and without formally specifying what constitutes a problem or solution, it is bound to be somewhat subjective. As there was just one author, it was not possible balance this subjectivity by including another person into the assessment of potential problems and solutions. The same applies to connecting problems with their potential solutions in the problem-solution matrix.

Lastly, since papers from 2007 to 2016 were used in the study, it is possible that some of the problems mentioned have already been solved.

5 Results from case study

This chapter answers RQ4 and RQ5:

- RQ4: Are the problems described in literature considered relevant by industry professionals?
- RQ5: Do industry professionals consider the solutions proposed in literature promising?

5.1 Are the problems described in literature considered relevant by industry professionals?

Pre-questionnaire information

Five of the six interviewed companies expressed that fragmentation was a significant problem in mobile application testing before seeing the list of problems proposed in literature. For all of them it was the only problem that they mentioned and in multiple cases the interviewees expressed it spontaneously before I asked any questions. Testlio was the only company who didn't consider fragmentation as a significant problem because their community-based approach already ensures a high number of different platform, OS version, device and screen size combinations.

The interviewee from Testlio mentioned two challenges in mobile application testing. The first one was that there is a lack of fine-grained tools that testers could use to record GUI interactions leading to a fault. The ideal approach would be able to capture videos, screenshots with click positions and have better logs than the current approaches. The other mentioned problem was applications that need to be tested in very specific geographical locations, especially on iOS where location information is more difficult to mock than on Android.

Questionnaire answers

The questionnaire answers provided by all of the companies are listed in Table 1. Questions corresponding to each question number can be seen from Appendix II. The three companies that mainly focus on testing are displayed on the left while companies whose main area of business is mobile application development are displayed in the right half of the table.

Table 1: Relevance of problems

	Testing companies			Development companies		
Question	Fob Solutions	Testlio	TestDevLab	Wazombi	Mooncascade	Mobi Lab
1	Definitely	Solved	Definitely	Maybe	Definitely	Definitely
2	Solved	NA	Probably not	Definitely	Definitely	Maybe
3	Definitely	NA	Definitely not	Maybe	NA	Definitely
4	Maybe	Solved	Definitely not	Definitely	Probably not	Definitely not
5	Maybe	Solved	Definitely	Definitely not	Maybe	Definitely
6	Definitely	Solved	Definitely	Definitely not	Definitely	Definitely
7	Definitely not	NA	Definitely not	Solved	Definitely	Definitely
8	Definitely not	Definitely	Definitely	Probably not	Definitely	Probably not
9	Probably not	Maybe	Definitely	Definitely not	Definitely not	Definitely not
10	Maybe	Maybe	Probably not	Maybe	Probably not	Definitely not

11	Probably not	NA	Definitely	Solved	Maybe	Probably not
12	Definitely	Maybe	Definitely	Probably not	Probably not	Definitely not
13	Definitely not	NA	Definitely	Solved	Definitely	Definitely
14	Probably not	Maybe	Probably not	Probably not	Maybe	Definitely not
15	Maybe	Solved	Definitely not	Definitely not	Probably not	Definitely not
16	NA	Definitely	Probably not	Definitely not	Maybe	Definitely
17	Maybe	Maybe	Maybe	Definitely not	Maybe	Definitely not
18	Probably not	NA	Definitely	Maybe	Maybe	Definitely not
19	Probably not	NA	Definitely not	Probably not	Definitely	Definitely
20	Definitely	Definitely	Probably not	Probably not	Definitely	Maybe
21	Maybe	Definitely	Definitely	Solved	Probably not	Maybe
22	Maybe	Maybe	Definitely not	Solved	Definitely	Probably not
23	Definitely	NA	Definitely not	Definitely	Definitely	Maybe
24	Maybe	Solved	Probably not	Probably not	Definitely	Probably not
25	Probably not	Definitely	Definitely not	Probably not	Maybe	Probably not
26	Probably not	Definitely	Probably not	Definitely not	Probably not	Definitely not
27	Definitely	Definitely	Maybe	Probably not	Definitely	Definitely not
28	Definitely	Maybe	Definitely	Probably not	Probably not	Definitely not
29	Probably not	NA	Maybe	Maybe	Maybe	Maybe
30	Probably not	NA	Maybe	Definitely not	Definitely not	Definitely not
31	Maybe	Maybe	Maybe	Maybe	Definitely	Definitely not
32	Definitely not	Maybe	Definitely	Definitely not	Probably not	Definitely not
33	Probably not	NA	Definitely not	Solved	Probably not	Definitely not
34	Probably not	Maybe	Definitely not	Solved	Maybe	Definitely not
35	Maybe	NA	Definitely not	Definitely not	Probably not	Definitely not
36	Solved	Maybe	Probably not	Solved	Solved	Definitely not
37	Solved	Maybe	Probably not	Probably not	Solved	Definitely not
38	NA	NA	Definitely	Definitely	Definitely not	NA
39	NA	NA	NA	NA	NA	Probably not
40	Probably not	NA	Definitely not	Definitely not	Probably not	Probably not
41	Solved	Maybe	Definitely not	Solved	Solved	Solved
42	NA	NA	Definitely	Definitely not	Definitely	Probably not
43	Probably not	Probably not	Definitely not	Definitely not	Solved	Solved
44	Solved	Definitely	Definitely not	Solved	Solved	Solved
45	NA	NA	NA	Solved	Solved	Probably not
46	Definitely not	Probably not	Definitely not	Probably not	Maybe	Definitely not
47	Definitely not	Maybe	Probably not	Solved	Solved	Probably not
48	NA	Definitely	Probably not	Maybe	Probably not	Definitely not
49	NA	Maybe	Definitely not	Definitely	Definitely	Definitely not

The surveyed companies didn't completely agree on the relevance of any of the listed problems. However, some patterns can be pointed out.

While there was no problem that all of the companies uniformly considered definitely irrelevant, there were many that none of them considered very important. Testing inter-application communications (question 14) was not considered very important. The same

goes for some questions related to more sophisticated testing techniques like simulating external dependencies (question 10), automatic page-load detection (question 40) and ensuring completely clean application restart (question 43). None of the companies considered the lack of testing methods, tools or theory a significant problem (questions 33-35, 37). Modelling applications before testing was not popular and some companies mentioned that testing on all devices is not needed because a set of supported devices is chosen before development. It can be argued that this practice of choosing a set of supported devices itself shows that testing an application on all potentially suitable devices is too difficult, expensive or time-consuming.

Some things that were not considered problems by companies mainly focused on development were still considered potentially problematic by testing companies. These included acquiring a mental model of a complex application (question 9), the unpredictability of external dependencies during testing (question 12), ignoring unexpected user behaviour (question 15), users' variable mobile device usage experience (question 21), insufficient OS failure logging (question 30) and the usability and accessibility aspects of complex input mechanisms (question 32). My guess is that testing companies do testing more thoroughly or are just more aware of their testing processes. In addition to this, if testing is performed by developers or at least in the same company, then the people doing the testing probably have a better overview of how the application is intended to function.

Since Testlio was the only one of the companies that actively uses a community-based testing approach as opposed to just testing in the company, different problems were sometimes considered relevant by them. Notably, fragmentation (question 1) and the large number of test devices to buy (question 6) that were considered problems by most companies are not a problem for Testlio because their testers use personal devices for testing. On the other hand, they are subject to some challenges that are not relevant for any other companies. For example, since their testers are working remotely, they need more advanced UI recording tools (question 44) than companies that perform testing locally. The large number of test devices (question 6) is also not a problem for Wazombi who mainly provides end-to-end services that include both hardware and software development.

The lack of design principles (question 36) was considered already solved by guidelines provided by mobile operating systems and cross-platform principles were not considered necessary. The problem of not being able to modify a mobile application after installing (question 41) was considered solved by either automatic updates provided by app stores or specialized software that can be embedded into applications for A/B testing. The fact that testing is expected to be faster for mobile applications was not considered a big obstacle because mobile applications on average were said to contain less functionality than desktop applications.

In conclusion, it can be said that industry professionals consider some of the problems mentioned in literature very important while others are not considered relevant at all. Companies had assessed the relevance or same problems differently from each other. Part of it can be attributed to the fact that companies perform testing and development differently and use different tools, therefore different problems are relevant for them. There was one problem that the industry and researchers strongly agreed on – fragmentation, especially in the case of Android. Problems related to external communication, non-functional properties and lack of industry-wide standards were generally not considered very relevant. As an answer to research question 4 it can be said that companies consider only some of the problems mentioned in literature relevant in practice.

5.2 Do industry professionals consider the solutions proposed in literature promising?

The goal of this section is to determine to which extent industry professionals consider the solutions provided in literature potentially useful in practice. This section answers research question number 5: "Do industry professionals consider these solutions promising?"

Industry feedback regarding the solutions introduced in Section 4.2 is displayed in Table 2. Altogether, eleven solutions were presented to industry professionals based on the problems that they considered relevant. The following notation is used:

- 'Yes' – the company found this solution potentially useful
- 'Partly' – the solution was perceived to solve some of the corresponding problem(s), but has significant short-comings.
- 'Maybe' – details not mentioned in the paper are needed to tell whether the solution is useful
- 'No' – the companies do not consider the proposed solution useful
- '-' – no clear opinion was received from the interviewee. This only happened during the first interview where the solution descriptions were relatively abstract. To avoid this situation, more practical descriptions were used in later interviews.

Table 2: Relevance of solutions

		Solutions										
		S8	S9	S10	S11	S16	S21	S23	S25	S26	S28	S29
Companies	Fob Solutions	No	Exists			-			-			Exists
	Testlio	Yes	Partly			Yes	Yes					Yes
	Wazombi	No	No	N/A	N/A	No				N/A	Partly	
	TestDevLab	Yes	No	Yes	Maybe	Yes		No	Yes	Yes	Exists	Partly
	Mooncascade	Exists	Exists		Maybe	Yes				Yes	No	No
	Mobi Lab	Yes	No			Yes		Yes	Exists	Yes	Yes	Exists

Follows a description of how each introduced solution was received by the industry professionals, along with comments from the respondents.

S8 MobiBug: This solution was presented to all of the participating companies. Respondents from Testlio, TestDevLab and Mobi Lab considered it potentially useful. Wazombi representative commented that since even devices of the same model don't function com-

pletely identically, a model that assumes they do might be inaccurate. The interviewee from Mooncascade said that nowadays OS built-in logging is already more fine-grained than stated in the article and 3rd party libraries for monitoring fault configurations exist, therefore this solution basically already exists.

S9 iTest: None of the interviewees considered iTest a very useful innovation. Some mentioned that a solution of this kind already exists. Others were sceptical of whether this approach would work well because people rarely give any feedback when things work (Mobi Lab) and it is difficult to ensure a representative variety of user profiles in registered testers (Testlio). TestDevLab respondent expressed that the success of this approach is highly dependent on the tester incentive mechanism and therefore the technical solution alone does not bring much value.

S10 Symbolic execution of Android apps: This solution was described to two companies. Interviewee from TestDevLab considered it potentially useful while Wazombi representative said the solution would not be applicable for them because it can only handle applications written in Java.

S11 JPF-Android: As with the previous solution, this solution is not applicable to Wazombi whose applications are not Java-based. TestDevLab was hesitant about whether this would work and Mooncascade said the tool would be useful if it could also emulate drivers of all kinds of sensors and the developers would manage to keep the tool up to date with new OS versions.

S16 VALERA: Most of the companies liked the concept of VALERA and thought it would be useful. The Wazombi representative was more sceptical due to the fact that VALERA does not record memory operations.

S21 Tool with 2 approaches for automated model-based testing: This solution was only presented to Testlio representative who found it useful.

S23 Unit-testing performance: The solution for performance unit-testing was presented to interviewees from Mobi Lab and TestDevLab. The former found it useful while the latter commented that the duration of method execution can depend on things outside of the developer's control, for example network conditions, therefore long execution duration cannot be attributed to just performance.

S25 VanarSena: TestDevLab considered it useful. Mobi Lab said that something similar is already used for Windows Phone applications. This makes sense since three of the four authors were listed as associated with Microsoft.

S26 An approach for amplifying exception-handling code: The approach was presented to four companies. For Wazombi, this solution wasn't applicable due to being Java-based, but the others considered it useful.

S28 Knowledge base for compatibility testing: Opinions about this solution were mixed. The Mobi Lab representative thought that it could work and Wazombi said it could work partly, for API version based problems. Mooncascade was sceptical about how an appropriate level of granularity could be set for recording results – if every combination of application version, device, OS version, etc would be recorded separately, then very few queries would get a reply from database while in other cases there is a high probability of over-generalization. TestDevLab said that a solution like this is probably already integrated to some testing software.

S29 TestDroid was not very well-received. Fob Solutions and Mobi Lab said that this solution already exists, which is not difficult to verify. Even TestDroid itself is available

online and is not the only cloud-based testing platform. Testlio said that in principle the approach is plausible while TestDevLab thought it might be useful only for small developers who do not have access to an extensive set of test devices, but not for medium and big software development companies. Mooncascade was also of the opinion that for companies of significant size, it is better to have their own set of devices as cloud-based solutions are expensive, unreliable and do not support various testing styles and frameworks.

In total, there were 3 solutions that were found at least somewhat acceptable by all companies that considered them applicable. One was solution 21 that was only presented to one company. Solution 10 and 26 were proposed to two and four companies, respectively, and were considered relevant by companies that found it applicable. One company even wanted to try out solution 26. None of the respondents considered iTest (S9) a good solution and most were sceptical about solutions 28 and 29.

On the other hand, there were many solutions that did not bring any additional value to the interviewed companies. Upon hearing the solution concepts, many interviewees expressed that the general concept of the solution is familiar to them or already exists. However, they had not marked the corresponding problems as "Already solved" in the questionnaire part of the interview. This implies that the concepts they already knew either do not fully solve the proposed problem or the professionals have not thought about using this concept to solve the given problem. The latter is compatible with my general observation from the interviews that companies consider the additional challenges of mobile application testing inevitable and do not think about the possibility to eliminate or lessen them. From that perspective, help from the scientific community at tackling the limitations could be useful.

From the previously unknown solutions, some were just not considered useful. The main problems were that the solutions proposed by scientific papers were either too theoretical, general or concerned problems that were already solved for the industry. Part of the latter problem could definitely be attributed to the fact that articles published from 2007, almost 10 years ago, were used in the literature study. However, the two least supported solution concepts, TestDroid and iTest, were published in 2014 and 2012, respectively. Therefore, it is reasonable to assume that either the field of mobile application testing is developing so fast that only papers published less than two years ago provide practical value for companies or research is somewhat detached from the industry.

Another observation from the interviews is that additional attention could be paid to the different tools and frameworks that are used for developing and testing mobile applications. For example, not all Android applications are developed in Java and cloud-testing platforms would be more useful if they supported different testing frameworks.

5.3 Summary

None of the problems mentioned in literature were considered uniformly relevant by all industry professionals. However, most companies considered fragmentation a serious problem and usually mentioned it before being handed the questionnaire. Many of the problems mentioned in literature were not considered important by industry. To answer research question RQ4, some of the problems raised in scientific literature were also considered important by the industry while some were definitely not. Most problems were considered relevant by some of the participating companies, but not others.

Some of the solutions proposed in literature were considered useful, but companies also pointed out many ways in which the solutions could be improved. Only a small subset of the solutions proposed in literature was presented to companies, mostly because many solutions were very theoretical or general. A large portion of solutions were already said to

exist by industry professionals. On the other hand, some solutions were well-received and one company even spontaneously expressed interest in trying out one of the presented solutions. As an answer to research question RQ5, it can be said that companies consider a few of the solutions potentially useful.

5.4 Limitations

While both testing and development oriented companies with somewhat different properties were included in the study, the initial list of companies was compiled mostly based on my general knowledge about potentially relevant companies in Estonia. Therefore, it is possible that the participants of this study do not fully represent all companies involved with mobile application testing in Estonia.

During interviews participants were asked to assess the potential suitability of some solutions proposed in literature. Since it would be unreasonable to expect participants to read the relevant scientific articles, I shortly explained each solution concept that the interviewees were asked to assess on the spot. As a result, my personal bias and the quality of my explanations might have had an effect on the perceived usefulness of the solutions.

Due to time constraints and my assessment of what the companies would find interesting, some potential solutions were not presented. The decision of which solutions to present was made by me on the spot. This creates the possibility that the set of solutions proposed might not be the most optimal.

6 Conclusions

In this thesis a literature review and semi-structured interviews with industry professionals were conducted to find answers to five research questions concerning challenges and solutions of mobile application testing as seen by researchers and industry.

In the literature survey, 50 papers relevant to the given topic were discovered. By reading and analysing the papers, 49 potential problems or challenges in the mobile application testing domain were discovered. The second part of literature survey revealed 39 potential solutions, some of which were implemented software tools while others were just theoretical concepts. Although one or more of the solutions were also used in practice, in most cases scientific literature did not give much information about the industry usage of the proposed solutions. In addition to the tasks stemming from research questions, a mapping between the found problems and solutions was created.

Following the literature survey, interviews were conducted with professionals from six Estonian companies to assess the relevance of the problems and solutions described in scientific literature. Firstly, the respondents were asked to fill out questionnaires where they had to assess the importance of each of the problems extracted from scientific literature. This revealed that there is no industry-wide consensus about which problems and solutions are considered relevant. Different problems were seen as important depending on whether the company was focused solely on testing, whether testing was performed in the company and which tools were used. However, most companies considered fragmentation - the multitude of different hardware, screen sizes and operating system versions - a significant problem.

In the second part of the interview, some solutions proposed in industry were introduced to each industry professional. A few of them were considered potentially useful in practice, some even very much so. Many solutions were perceived as already existing, which seems plausible considering the amount of authors associated with some company active in the field. A significant portion of the proposed solutions were shown to have shortcomings. Many of the solutions described in literature were not presented to industry professionals because they were too theoretical or abstract to be practical to industry professionals in their current state.

All in all, research is addressing some problems that are considered very important by the industry. However, there is plenty of room for making research more usable by industry as many of the currently proposed solutions are too abstract, already exist in industry before a research paper is published or solve problems that are not considered relevant by the industry. Industry feedback presented in this paper can be used as a starting point for creating more practical mobile application testing tools.

Acknowledgements

I would like to thank Fob Solutions OÜ, Testlio OÜ, Mobi Lab OÜ, Wazombi Labs OÜ, Ltd. "TestDevLab" and Mooncascade OÜ for their time and feedback on the potential problems and solutions.

7 References

[1] H. Muccini, A. Di Francesco and P. Esposito, "Software Testing of Mobile applications: Challenges and Future Research Directions," in *AST '12 Proceedings of the 7th International Workshop on Automation of Software Test*, Piscataway, NJ, USA, 2012.

[2] S. Paul, "Role of mobile handhelds in redefining how we work, live and experience the world around us: some challenges and opportunities," in *Proceedings of the Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, New Delhi, India, 2010.

[3] A. I. Wasserman, "Software Engineering Issues for Mobile Application Development," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, Santa Fe, New Mexico, USA, 2010.

[4] A. Santos and I. Correia, "Mobile testing in software industry using agile: Challenges and opportunities," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*, Graz, Austria, 2015.

[5] N. T., "Did you know what was the first smartphone ever?," PhoneArena, 31 July 2014. [Online]. Available: http://www.phonearena.com/news/Did-you-know-what-was-the-first-smartphone-ever_id58842. [Accessed 10 May 2016].

[6] Apple Inc, "Apple Reinvents the Phone with iPhone," [Online]. Available: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html. [Accessed 12 May 2016].

[7] Statista Inc, "Number of smartphones sold to end users worldwide from 2007 to 2015 (in million units)," [Online]. Available: http://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. [Accessed 14 May 2016].

[8] C. Martinie and P. Palanque, "Design, Development and Evaluation Challenges for Future Mobile User Interfaces in Safety-Critical Contexts," in *Proceedings of the 2015 Workshop on Future Mobile User Interfaces*, Florence, Italy, 2015.

[9] L. Corral, A. Sillitti and G. Succi, "Software assurance practices for mobile applications," *Computing*, vol. 97, no. 10, pp. 1001-1022, 2015.

[10] Net Applications, "Mobile/Tablet Operating System Market Share January, 2016 to March, 2016," [Online]. Available: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1&qpsp=68&qpnp=1&qptimeframe=Q&qpmr=10&qpdt=0&qpct=3. [Accessed 12 May 2016].

[11] Google, "Performance focus," [Online]. Available: http://developer.android.com/about/versions/lollipop.html#Perf. [Accessed 8 April 2016].

[12] C. Albanesius, "Nadella Raises Eyebrows With Plans to 'Streamline' Windows," PC Magazine, 23 July 2014. [Online]. Available: http://www.pcmag.com/article2/0,2817,2461253,00.asp. [Accessed 13 May 2016].

[13] A. Charland and B. Leroux, "Mobile Application Development: Web vs. Native," *Commun. ACM*, vol. 54, no. 5, pp. 49-53, 2011.

[14] L. Zhifang, L. Bin and G. Xiaopeng, "Test Automation on Mobile Device," in

Proceedings of the 5th Workshop on Automation of Software Test, Cape Town, South Africa, 2010.

[15] M. Yan, H. Sun and X. Liu, "ITest: Testing software with mobile crowdsourcing," in *1st International Workshop on Crowd-Based Software Development Methods and Technologies, CrowdSoft 2014 - Proceedings*, Hong Kong, China, 2014.

[16] J. Kaasila, D. Ferreira, V. and Kostakos and T. Ojala, "Testdroid: Automated Remote UI Testing on Android," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, Ulm, Germany, 2012.

[17] L. Ravindranath, S. Nath, J. Padhye and H. Balakrishnan, "Automatic and scalable fault detection for mobile applications," in *MobiSys 2014 - Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, Bretton Woods, New Hampshire, USA, 2014.

[18] H. Ham and Y. Park, "Designing knowledge base mobile application compatibility test system for android fragmentation," *International Journal of Software Engineering and its Applications,* vol. 8, no. 1, pp. 303-314, 2014.

[19] L. Tang, Z. Yu, X. Zhou, H. Wang and C. Becker, "Supporting rapid design and evaluation of pervasive applications: challenges and solutions," *Personal and Ubiquitous Computing,* vol. 15, no. 3, pp. 253-269, 2011.

[20] J. A. Galindo, H. Turner, D. Benavides and J. White, "Testing variability-intensive systems using automated analysis: an application to Android," *Software Quality Journal,* vol. 42, no. 2, pp. 365-405, 2014.

[21] S. Baride and K. Dutta, "A Cloud Based Software Testing Paradigm for Mobile Applications," *SIGSOFT Softw. Eng. Notes,* vol. 36, no. 3, pp. 1-4, 2011.

[22] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of Android apps," *ACM SIGPLAN Notices,* vol. 48, no. 10, pp. 641-660, 2013.

[23] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel, "FLOWDROID: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices,* vol. 49, no. 6, pp. 259-269, 2014.

[24] K. Haller, "Mobile Testing," *SIGSOFT Softw. Eng. Notes,* vol. 38, no. 6, pp. 1-8, 2013.

[25] J. C. Bastien, "Usability testing: a review of some methodological and technical aspects of the method," *International Journal of Medical Informatics,* vol. 79, no. 4, pp. e18-e23, 2010.

[26] X. Ma, B. Yan, G. Chen, C. Zhang, K. Huang, J. Drury and L. Wang, "Design and implementation of a toolkit for usability testing of mobile apps," *Mobile Networks and Applications,* vol. 18, no. 1, pp. 81-97, 2013.

[27] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani and R. Mahmood, "Testing Android Apps Through Symbolic Execution," *SIGSOFT Softw. Eng. Notes,* vol. 37, no. 6, pp. 1-5, 2012.

[28] H.-K. Kim, "Hybrid model based testing for mobile applications," *International Journal of Software Engineering and its Applications,* vol. 7, no. 3, pp. 223-238, 2013.

[29] H. van der Merwe, O. Tkachuk, B. van der Merwe and W. Visser, "Generation of Library Models for Verification of Android Applications," *SIGSOFT Softw. Eng. Notes,* vol. 40, no. 1, pp. 1-5, 2015.

[30] A. Hussain, N. Hashim, N. Nordin and H. Tahir, “A metric-based evaluation model for applications on mobile phones,” *Journal of Information and Communication Technology,* vol. 12, no. 1, pp. 55-71, 2013.

[31] E. M. I. Koivisto and R. Suomela, “Using Prototypes in Early Pervasive Game Development,” in *Proceedings of the 2007 ACM SIGGRAPH Symposium on Video Games*, San Diego, California, USA, 2007.

[32] H. van der Merwe, O. Tkachuk, S. Nel, B. van der Merwe and W. Visser, “Environment Modeling Using Runtime Values for JPF-Android,” *SIGSOFT Softw. Eng. Notes,* vol. 40, no. 6, pp. 1-5, 2015.

[33] S. Agarwal, R. Mahajan, A. Zheng and V. Bahl, “Diagnosing Mobile Applications in the Wild,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Monterey, California, 2010.

[34] H. Kim, B. Choi and S. Yoon, “Performance testing based on test-driven development for mobile applications,” in *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, ICUIMC'09*, Suwon, S.Korea, 2009.

[35] C. M. Avancini A., “Security testing of the communication among Android applications,” in *2013 8th International Workshop on Automation of Software Test, AST 2013 - Proceedings,* San Francisco, CA, USA, 2013.

[36] C. Q. Adamsen, G. Mezzetti and A. Møller, “Systematic Execution of Android Test Suites in Adverse Conditions,” in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, Baltimore, MD, USA, 2015.

[37] P. Zhang and S. Elbaum, “Amplifying tests to validate exception handling code: An extended study in the mobile application domain,” *ACM Transactions on Software Engineering and Methodology,* vol. 23, no. 4, pp. 32:1-32:28, 2014.

[38] G. De Cleva Farto and A. Endo, “Evaluating the model-based testing approach in the context of mobile applications,” *Electronic Notes in Theoretical Computer Science,* vol. 314, pp. 3-21, 2015.

[39] B. C. Zapata, J. L. Fernández Alemán, A. Idri and A. Toval, “Empirical Studies on Usability of mHealth Apps: A Systematic Literature Review,” *Journal of Medical Systems,* vol. 39, no. 2, pp. 1-19, 2015.

[40] S. Diewald, B. Geilhof, M. Siegrist, P. Lindemann, M. Koelle, M. Halle and M. Kranz, “Mobile AgeCI: Potential Challenges in the Development and Evaluation of Mobile Applications for Elderly People,” in *Computer Aided Systems Theory -- EUROCAST 2015: 15th International Conference*, Las Palmas de Gran Canaria, Spain, 2015.

[41] A. Oulasvirta, “Rethinking experimental designs for field evaluations,” *IEEE Pervasive Computing,* vol. 11, no. 4, pp. 60-67, 2012.

[42] B. Biel, T. Grill and V. Gruhn, “Exploring the benefits of the combination of a software architecture analysis and a usability evaluation of a mobile application,” *Journal of Systems and Software,* vol. 83, no. 11, pp. 2031-2044, 2010.

[43] A. Rapp, F. Cena, C. Gena, A. Marcengo and L. Console, “Using game mechanics for field evaluation of prototype social applications: a novel methodology,” *Behaviour and Information Technology,* vol. 35, no. 3, pp. 184-195, 2015.

[44] M. Billi, L. Burzagli, T. Catarci, G. Santucci, E. Bertini, F. Gabbanini and E. Palchetti, “A unified methodology for the evaluation of accessibility and usability of mobile applications,” *Universal Access in the Information Society,* vol. 9, no. 4, pp.

337-356, 2010.

[45] L. H. d. Nascimento and P. D. Machado, “An Experimental Evaluation of Approaches to Feature Testing in the Mobile Phone Applications Doman,” in *DOSTA '07 Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting*, Dubrovnik, Croatia, 2007.

[46] H. van der Merwe, B. van der Merwe and W. Visser, “Verifying Android Applications Using Java PathFinder,” *SIGSOFT Softw. Eng. Notes,* vol. 37, no. 6, pp. 1-5, 2012.

[47] S. Salva and S. R. Zafimiharisoa, “APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities,” *International Journal on Software Tools for Technology Transfer,* vol. 17, pp. 201-221, 2015.

[48] E. Aranha and P. Borba, “Estimating manual test execution effort and capacity based on execution points,” *International Journal of Computers and Applications,* vol. 31, no. 3, pp. 167-172, 2009.

[49] L. C. Serra, L. P. Carvalho, L. P. Ferreira, J. B. S. Vaz and A. P. Freire, “Accessibility Evaluation of E-Government Mobile Applications in Brazil,” *Procedia Computer Science,* vol. 37, pp. 348-357, 2015.

[50] I. C. Morgado and A. C. R. Paiva, “Test Patterns for Android Mobile Applications,” in *Proceedings of the 20th European Conference on Pattern Languages of Programs*, Kaufbeuren, Germany, 2015.

[51] X. S. Wang, A. Balasubramanian, A. Krishnamurthy and D. Wetherall, “Demystifying Page Load Performance with WProf,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, Lombard, IL, 2013.

[52] H. P. Holzmann C., “Multivariate testing of native mobile applications,” in *12th International Conference on Advances in Mobile Computing and Multimedia, MoMM 2014*, Kaohsiung, Taiwan, 2014.

[53] W. Choi, G. Necula and K. Sen, “Guided GUI testing of Android apps with minimal restart and approximate learning,” *ACM SIGPLAN Notices,* vol. 48, no. 10, pp. 623-639, 2013.

[54] Y. Hu, T. Azim and I. Neamtiu, “Versatile Yet Lightweight Record-and-replay for Android,” *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications,* vol. 50, no. 10, pp. 349-366, 2015.

[55] J. Gao, W.-T. Tsai, R. Paul, X. Bai and T. Uehara, “Mobile testing-as-a-service (MTaaS) - Infrastructures, issues, solutions and needs,” in *Proceedings - 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, Miami, FL, United States, 2014.

[56] P. Costa, A. C. R. Paiva and M. Nabuco, “Pattern Based GUI Testing for Mobile Applications,” in *9th International Conference on the Quality of Information and Communications*, Guimaraes, Portugal, 2014.

[57] J. Bo, L. Xiang and G. Xiaopeng, “MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices,” in *Proceedings of the Second International Workshop on Automation of Software Test*, Washington, DC, USA, 2007.

[58] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta and A. M. Memon, “MobiGUITAR: Automated Model-Based Testing of Mobile Apps,” *IEEE Software,*

vol. 32, no. 5, pp. 53-59, 2015.

[59] D. Amalfitano, N. Amatucci, A. R. Fasolino and P. Tramontana, “AGRippin: A Novel Search Based Testing Technique for Android Applications,” in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, Bergamo, Italy, 2015.

[60] D. Amalfitano, N. Amatucci, A. R. Fasolino, P. Tramontana, E. Kowalczyk and A. M. Memon, “Exploiting the Saturation Effect in Automatic Random Testing of Android Applications,” in *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, Florence, Italy, 2015.

[61] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine and A. M. Memon, “Using gui ripping for automated testing of android applications,” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Essen, Germany, 2012.

[62] B. Bergvall-Kåreborn and S. Larsson, “A Case Study of Real-world Testing,” in *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, Umeå, Sweden, 2008.

[63] C. Guo, J. Xu, H. Yang, Y. Zeng and S. Xing, “An automated testing approach for inter-application security in android,” in *9th International Workshop on Automation of Software Test, AST 2014 - Proceedings*, Hyderabad, India, 2014.

Appendix

I. Search queries for structured search

All of the queries were run on 17.03.2016.

Scopus: 138 results.

(TITLE("mobile app*" OR "Android app*" OR "iOS app*" OR "Windows Phone app*" OR "Symbian app*" OR "BlackBerry app*") OR KEY("mobile app*" OR "Android app*" OR "iOS app*" OR "Windows Phone app*" OR "Symbian app*" OR "BlackBerry app*")) AND (TITLE (test* OR reliability OR verification OR validat* OR evaluat* OR "quality assurance") OR KEY(test* OR reliability OR verification OR validat* OR evaluat* OR "quality assurance")) AND SUBJAREA(comp) AND PUBYEAR > 2006 AND SRC-TYPE (j) AND LANGUAGE(english) AND NOT(TITLE (web* OR "cross-platform") OR KEY(web* OR "cross-platform"))

ACM: 33 results.

(keywords.author.keyword: (+(mobile android iOS "Windows phone" Symbian BlackBerry)) OR title: (+(mobile android iOS "Windows phone" Symbian BlackBerry))) AND (keywords.author.keyword: (+(test* "quality assurance" validat* verification evaluat* reliability)) OR title: (+(test* "quality assurance" validat* verification evaluat* reliability))) AND NOT (title:(web* OR "cross-platform") OR keywords.author.keyword:(web* OR "cross-platform"))

Refinements:

Published since: 2007

All Publications: Periodical

SpringerLink: 178 results.

("mobile app*" OR "Android app*" OR "iOS app*" OR "Windows Phone app*" OR "Symbian app*" OR "BlackBerry app*") AND (test* OR reliability OR verification OR validat* OR evaluat* OR "quality assurance") AND NOT (web OR "cross-platform")

within English, Computer Science, SWE, Article

ScienceDirect: 25 results.

pub-date > 2006 and (TITLE("mobile app*" OR "Android app*" OR "iOS app*" OR "Windows Phone app*" OR "Symbian app*" OR "Blackberry app*") OR KEY("mobile app*" OR "Android app*" OR "iOS app*" OR "Windows Phone app*" OR "Symbian app*" OR "Blackberry app*")) and (TITLE(test* OR reliability OR verification OR validat* OR evaluat* OR "Quality assurance") OR KEY(test* OR reliability OR verification OR validat* OR evaluat* OR "Quality assurance")) [All Sources(Computer Science)]

II. Questionnaire

Which of the following do you consider problems/challenges in mobile application testing? A problem can be defined as something that reduces the effectiveness or efficiency of testing, or hinders the testing process in some other way.

Problem/Challenge	Definitely	Maybe	Probably not	Definitely not	N/A	Already solved
Fragmentation						
1. Due to fragmentation (device, OS, API ver), compatibility testing takes a lot of time, effort and money	☐	☐	☐	☐	☐	☐
2. Due to fragmentation, it is difficult to programmatically test whether the application is rendered correctly on all devices	☐	☐	☐	☐	☐	☐
3. Cross-platform tests need to be very abstract, which makes automating them difficult	☐	☐	☐	☐	☐	☐
4. It is challenging to provide natural interaction during user tests because each user is used to a specific OS and device model	☐	☐	☐	☐	☐	☐
5. Performance varies a lot across devices, which makes it difficult to optimize applications to work both on low-performance and high-performance devices	☐	☐	☐	☐	☐	☐
6. Large set of test devices to buy and maintain	☐	☐	☐	☐	☐	☐
7. Emulating all device-OS version configurations is computationally expensive. (Alternative to testing on devices)	☐	☐	☐	☐	☐	☐
External software dependencies						
8. Problems on levels below the application and between layers (application, framework, OS, hardware)	☐	☐	☐	☐	☐	☐
9. It is difficult for a tester to acquire a mental model of the application because the systems are complex and applications are tightly coupled	☐	☐	☐	☐	☐	☐

Problem/Challenge	Definitely	Maybe	Probably not	Definitely not	N/A	Already solved
with environment						
10. There are so many external dependencies that it is difficult to create an environmental model for simulating them	☐	☐	☐	☐	☐	☐
Frequent external communication						
11. Inputs from lots of different sensors have to be considered when testing	☐	☐	☐	☐	☐	☐
12. Cannot control behaviour of external dependencies during testing and analysis	☐	☐	☐	☐	☐	☐
13. Emulators cannot properly simulate sensors	☐	☐	☐	☐	☐	☐
14. Existing testing tools/approaches consider mobile applications only in isolation. Incoming messages (Intents) are therefore usually pooly validated.	☐	☐	☐	☐	☐	☐
15. Testing focuses on expected user behaviour, skipping the unexpected	☐	☐	☐	☐	☐	☐
16. Mobile applications need to be evaluated in distributed mode and heterogeneous environment	☐	☐	☐	☐	☐	☐
17. External resources often have transient failures	☐	☐	☐	☐	☐	☐
18. For testing exception-handling, it has to be possible to put all external resources into any desired state	☐	☐	☐	☐	☐	☐
19. Some external resource exceptions only appear after very specific input sequences	☐	☐	☐	☐	☐	☐
Variable user and usage context						
20. Due to the wide variety of mobile device users, multiple different user profiles have to be considered	☐	☐	☐	☐	☐	☐

Problem/Challenge	Definitely	Maybe	Probably not	Definitely not	N/A	Already solved
21. Difficult to test usability because users have very different experience levels with using mobile devices	☐	☐	☐	☐	☐	☐
22. Difficult to create realistic physical and situation context for testing context-aware applications	☐	☐	☐	☐	☐	☐
23. User scenarios involving lots of sensor usage are difficult to realistically script	☐	☐	☐	☐	☐	☐
24. Many real-life dependencies don't appear in lab	☐	☐	☐	☐	☐	☐
25. Usage context (situation, goal, distractions) that has a significant effect on usage of mobile applications cannot be recorded.	☐	☐	☐	☐	☐	☐
Fast evolution						
26. Methods of human-device interaction (keyboard, touch screen, tap, multi-touch, slide) are constantly changing	☐	☐	☐	☐	☐	☐
27. Application requirements constantly change to integrate new applications and technologies, as well as adapt to market trends	☐	☐	☐	☐	☐	☐
28. Fast-evolving devices and platforms lack reliability	☐	☐	☐	☐	☐	☐
Limited resources						
29. Difficult to monitor large number of dependencies without affecting performance	☐	☐	☐	☐	☐	☐
30. Insufficient OS built-in failure logging	☐	☐	☐	☐	☐	☐
31. System response time depends on how occupied the device is with other applications	☐	☐	☐	☐	☐	☐

Problem/Challenge	Definitely	Maybe	Probably not	Definitely not	N/A	Already solved
32. Complex input mechanisms that hinder accessibility and usability only for some users are difficult for others to test	☐	☐	☐	☐	☐	☐
Novelty						
33. Few testing tools for mobile applications	☐	☐	☐	☐	☐	☐
34. Non-functional properties are vaguely defined for mobile applications	☐	☐	☐	☐	☐	☐
35. Late performance testing due to lack of efficient methods	☐	☐	☐	☐	☐	☐
36. There is no mobile application specific quality framework or set of design principles	☐	☐	☐	☐	☐	☐
37. There are no mobile-specific best practises for conducting tests and usability assessments	☐	☐	☐	☐	☐	☐
Limitations related to platform implementation						
38. Android applications cannot be run on JVM and need special libraries that are available only in Android devices and emulators	☐	☐	☐	☐	☐	☐
39. Running tests on Dalvik VM is slow	☐	☐	☐	☐	☐	☐
40. Difficult to programmatically determine when a page finishes loading	☐	☐	☐	☐	☐	☐
41. Application cannot be changed after installing	☐	☐	☐	☐	☐	☐
42. Mobile-specific constructs have to be considered when producing control or data flow graphs and assessing test coverage	☐	☐	☐	☐	☐	☐

Problem/Challenge	Definitely	Maybe	Probably not	Definitely not	N/A	Already solved
43. The only reliable way to do a clean restart of the application is to reinstall it	☐	☐	☐	☐	☐	☐
44.UI interactions cannot be automatically recorded	☐	☐	☐	☐	☐	☐
45. No low-level access to OS for debugging	☐	☐	☐	☐	☐	☐
Others						
46. Testing is expected to be faster than for desktop applications	☐	☐	☐	☐	☐	☐
47. Social applications require testing in social context, which is difficult to artificially create	☐	☐	☐	☐	☐	☐
48. General-purpose development methodologies don't suit with the limitations of mobile devices and platforms	☐	☐	☐	☐	☐	☐
49. Traditional task-based testing approaches don't apply because interactions with mobile devices are not necessarily task-based	☐	☐	☐	☐	☐	☐

III. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Triin Samuel** (date of birth: 17.12.1990),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Problems and solutions in mobile application testing,

supervised by Dietmar Alfred Paul Kurt Pfahl,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **19.05.2016**