Puslished in: ACM SIGSOFT Software Engineering Notes Nov.'13, Vol. 38, DOI: 10.1145/2532780.2532813

Mobile Testing

Klaus Haller
Swisscom IT Services
Pfingstweidstr. 51
CH-8005 Zürich
Switzerland

klaus.haller@swisscom.com

ABSTRACT

Mobile apps are everywhere. Some apps entertain and others enable business transactions. Apps increasingly interact with complex IT landscapes. For example, a banking app on a mobile device acts as a front end that invokes services on a back-end server of the bank, which might contact even more servers. Mobile testing becomes crucial and challenging. This paper follows a user-centric testing approach. The app's architecture matters for testing, as does its user base and usage context. Addressing these factors ensures that test cases cover all relevant areas. Most apps need test automation for two reasons: agility and compatibly. Agile projects test frequently, such as every night, to detect bugs early. Compatibility tests ensure that an app runs on all relevant devices and operating system versions on the market. Thus, testers execute test scripts on many devices. This demands for a private device cloud and a mobile test automation framework. Swisscom IT Services followed this path, enabling us to address the major quality issues we identified for mobile apps: pre-usage failures (installation fails, app crashes during startup) and lack of basic regression testing (upgrades buggier than predecessor).

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Testing tools data generators, coverage testing); D.2.9 [**Management**]: Life cycle; Software quality assurance (SQA); K.6.1 [**Project and People Management**]: Life cycle; K.6.3 [**Software Management**]: Software development, Software process

General Terms

Management, Reliability, Verification.

Keywords

Software Testing, Mobile Apps, Mobile Devices, Test Automation, Software Quality Management, Test Strategy

1. INTRODUCTION

In 2012, worldwide smartphone sales topped 675 million units [1]. Tablets are also sold and smart watches and glasses will soon follow. As users move from PCs to mobile devices, companies had to follow this trend and invest in mobile apps.[1] This change represents a challenge for testing because of, for example, the various devices, short innovation cycles for operating systems and hardware, and apps invoking services on back-end servers. Testing is mature, but some concepts must be fine-tailored for mobile apps, which is the aim of the paper.

This paper is based on mobile testing projects from Swisscom IT Services, one of the leading Swiss IT service providers (outsourcing, workplace, SAP, financial industries), with more than 2,600 employees in Switzerland, Austria, and Singapore.

This paper follows Drucker's philosophy: "Efficiency is doing things right. Effectiveness is doing the right things." The focus is on understanding the value that tools and concepts offer to testing mobile apps. This paper provides answers to the following questions.

- Which tests are specific for mobile apps? (Section 2)
- What do users like or dislike about mobile apps? (Section 3)
- How do the architecture, user context, and user base affect which test cases are needed? (Section 4)
- What are the new infrastructure and automation needs? (Section 5)

Figure 1 summarizes this user-centric approach to testing.

We present the Swisscom Mobile Testing Framework in Section 6 before discussing related work in Section 7 and concluding with a short summary (Section 8).

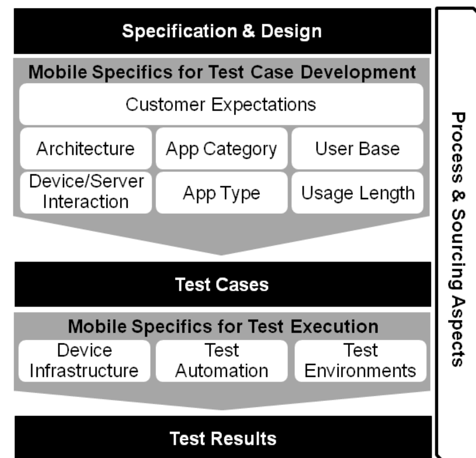

Figure 1: Mobile Specifics for Testing

2. MOBILE APPS AND THE TEST PROCESS

Similar to any other software system, mobile apps require testing. Testing is typically structured into unit, unit integration, system, and system integration tests. This structuring applies for the V-model, which performs the tests in a strict order. For agile projects, these test levels overlap more (see [2] for details).

For mobile apps, *unit and unit integration tests* focus on tests, which run either on the mobile device or on the back-end server. Front-end tests for apps can be basic, such as for mobile apps with static HTML pages. They can also be complex, such as for gaming apps, which run primarily only on the mobile device. Normally, developers perform the unit and unit integration tests (Figure 2).

[1] To prevent misunderstandings: We subsume also web applications optimized for mobile access under the term *mobile app*. A discussion of the various architectural options follows.

Test cases, which run code on the device and the back-end server, belong to *system tests*. *System integration tests* cover test cases for which the back end interacts with other servers, for example, as a result of user actions on the mobile device. System and system integration tests are the main domain of testers. These tests are similar to (web) client/server application tests.

Device tests and testing in the wild are two new types of tests for mobile apps. *Device tests* are tests using various mobile devices to ensure compatibility. The concept is to perform the same test case on various devices. *Testing in the wild* means testing an app in its real usage context. An app for commuters must be tested in trains, trams, buses in the city, and tunnels. A hiking app must work in valleys and on mountains. Testing in the wild is a specific type of exploratory test for mobile apps.

Finally, in the past, the main triggers for tests were new software releases, bugs and bug-fixes. A new trigger exists in the mobile world: new devices and operating system versions. Apps have to be tested against them as soon as they are on the market, preferably even sooner.

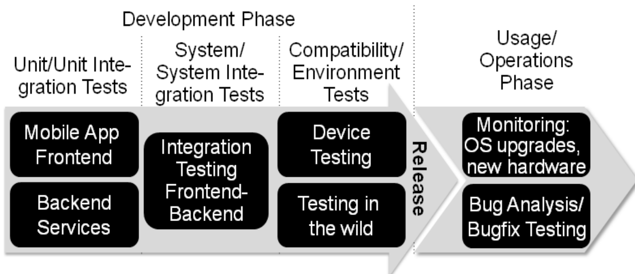

Figure 2: Mobile Testing during the Software Lifecycle

3. Customer Expectations

The V-model represents old school testing. Projects start with a specification. Developers implement that specification. Testers test the implementation against the specification. If the test succeeds, the customer pays and must be happy. This model never worked, and will never work in the mobile sphere. Figure 3 illustrates why. Developers and testers have to test the mobile app, since the app must work. However, its success depends—for the business, not for IT—on other criteria, such as:

- What do users expect?
- How good are competing offers?
- How does the app boost (and not ruin) the reputation of the company?
- Does it achieve the aims of the business?

These questions go beyond an old-fashioned understanding of testing as defining test cases and executing them repetitively. However, they help to understand how users perceive mobile apps quality today, even if the ultimate consequence is a redefinition of the role of testers.

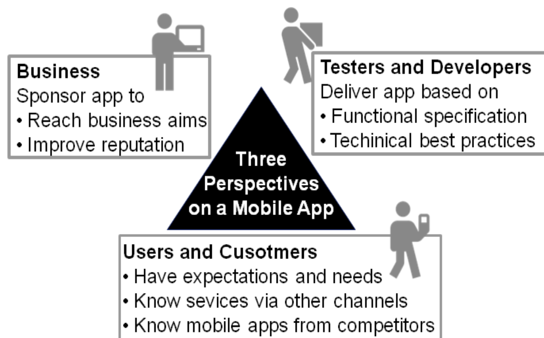

Figure 3: App Quality and Success in the Mobile World

The **business aims** of apps differ. They may be $65,000 in in-game sales or may be to convince 1,000 customers to switch to mobile banking. Although the business aims differ between apps, their effect on **reputation** can be measured (partially) the same way. Thousands of users download an app, rate an app in the app store, and write comments. So, if testers look only at the specifications, they overlook other important aspects, which our analysis identified.

We reviewed the comments of more than 1,000 app ratings related to 54 apps in two major and one small app stores.[2] The apps are diverse: Swisscom apps, apps of Swiss banks, games, and other Swiss and international apps. We analyzed the user comments from a qualitative perspective. In other words, we focused on the types of comments rather than statistics. Figure 4 structures the various types of user complaints, which the following paragraphs discuss in greater detail.

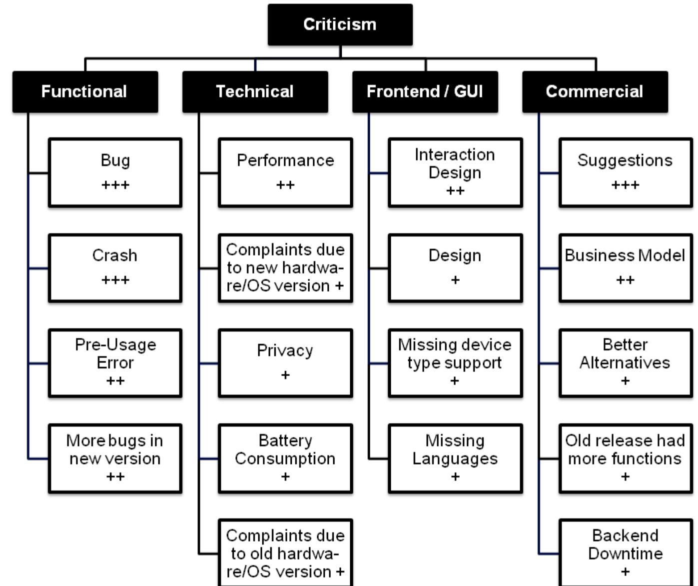

Figure 4: Identified Complaints in >1,000 User Ratings for Apps (mentioned in + 0.1–4.9%, ++ 5.0–9.9%, +++ 10–20% of all ratings)

Four main groups of feedback exist. The first covers functional issues:

- *Bugs:* Users report that features cannot or do not work properly (for example, incorrect calculations). The focus is on an error that exists in the implementation that the user believes is worth reporting. The bug can be minor or severe.
- *Crash:* The application starts and can be used but later crashes or freezes. This can happen either frequently or infrequently.
- *Pre-Usage Errors:* Users do not get to the point they can work with the app. Typical reasons include download or installation problems or failures during the app start. Also, this covers cases when, for example, the app expects a Facebook account, but the user does not have or want one.
- *More bugs with a New Version:* The new version has more bugs than the old version.

On a more technical side, the following remarks were made:

- *Performance:* Users believe that the app is slow or the screen freezes for a short moment. Possible root causes include old hardware, bad coding and algorithms, large downloads, or network issues.
- *New Hardware/OS Complaints:* The app does not work with or is not optimized for new hardware or new operating system versions.

[2] The same app was counted twice if published for Android and iOS given different code bases.

- *Data Privacy*: The app requests permissions that the user thinks are not appropriate.
- *Battery Consumption:* The user complains that the app consumes too much battery.
- *Old Hardware/OS Complaints:* The app does not run properly on old hardware or operating systems.

The third group of feedbacks circles around the interaction between the user and the mobile app:

- *Interaction Design:* Use of the app is not intuitive or appears illogical to users, which also covers gameplay issues related to gaming apps.
- *Design:* The app does not look "nice"; for example, the text is too small to be read.
- *Missing Device Type Support:* An app is not available or not optimized for a device type (smartphone, tablet, for iOS: iPod). This issue may result from a vendor decision, but, for us, the effect on reputation (negative feedback in the app store) matters.
- *Missing Language:* Languages are a sensitive issue in countries with more than one spoken language. Users complain that their language is not available (German, French, Italian, or English).

Other complaints are the results of business decisions:

- *Suggestions:* Users have ideas for new features and suggest them. This is input for business analysis because the actual app does not match all user needs.
- *Business Model:* Users dislike how the app is managed. For example, they state that the app is priced too high, customer service does not respond, the app is outdated, user feedback is ignored, there are too many ads etc.
- *Better Alternatives:* Users know of better apps from competitors. Additionally, the company itself might offer better options through other channels, such as a web page for PC users or apps for competing mobile operating systems.
- *Removed Features:* An old version of the app has features that a new does not provide. Users miss these old features.
- *Backend Downtime:* The app works only when services run on a back-end server. Back-end server downtime hinders usage of the mobile app.

We conclude with the top five complaints and remarks, which are: improvement ideas, bugs, crashes, complaints about the business model, and more bugs with a new release. From this list, we derive two theses about the business of mobile apps today.

- **App stores are an easy way to engage in customer co-creation.** [3] Customers post what they like or dislike, which helps (at least) to implement incremental improvements—a good finding. At the same time, many customers complain about the business model. This must be analyzed on a per app basis and is a reputational risk.
- **Many projects do not apply basic testing techniques.** They seem not to engage in proper regression tests. Indicators are frequent crashes, not being able to install and start software, and buggier new versions. Device tests and (automated) functional tests could improve app quality.

4. MOBILE APPS AND TEST CASES

This section discusses the areas that require special attention in mobile testing. This helps testers when they design test cases. For the test case design itself, they apply the known methods for test case design, such as boundary value analysis or use-based testing [4].

4.1 Architecture

Architecture defines the components and their interplay, and addresses questions such as: Where is which part of the code running? Which components communicate, and how and when? When testers understand the architecture, they get ideas on where an app can break, and, thus, where to test. There are four main architectural patterns in the mobile world (Figure 5).

Native standalone apps have existed since the early, pre-smartphone period. A calculator and simple games fall into this category. These apps are downloaded (or pre-installed) and then run on the mobile device itself. The apps do not communicate with servers.

Mobile web apps are web pages optimized for presentation on mobile phones. The web server provides web pages, which the mobile device loads and presents. Certainly, the web page can contain code running in the browser of the mobile device.

Native client server (C/S) apps are native apps. Users download and install them on their mobile devices. When used, the app connects to a back-end server to invoke a service, such as retrieving data. Mobile banking apps are examples.

Launcher apps are in the middle between mobile web apps and native C/S apps. Users download and install the app on their device. Certain features are provided by the app. In other cases, the app redirects users to a mobile web app.

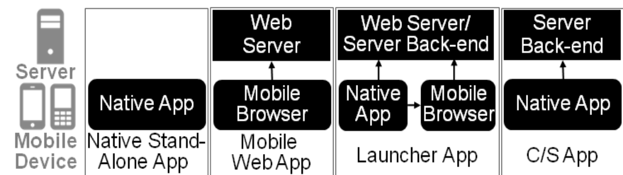

Figure 5: Architectural Patterns for Mobile Applications

4.2 Usage Context

We introduced "in the wild" testing in Section 2. The idea is to test an app in the same context in which users will later use the app. Doing so requires characterizing the usage contexts. The first step is Leland Rechis' classification of three mobile users groups [5]. **Bored now** users have time to kill, such as when commuting on a train or sitting in a café. **Urgent now** users need a quick answer for an immediate need, such as a train timetable. **Repetitive now** refers to users who do the same thing over and over, such as looking for stock prices, reading the news, or checking for new messages.

The three usage groups translate into three questions about the usage context of an app.

- What is the **app type**? *Interactive apps* such as games use various options for user interaction, including gestures. They present results in a visually appealing manner using nice graphics and videos. Phone directories or online shops are examples of *no-nonsense apps*. They have a straightforward design. Users must get results quickly or close a transaction within a few clicks.
- What is the **usage length?** Usage length can be a few seconds (phone book), minutes (maps), or even hours (games), and affects which and how many tests are needed to test network bandwidth fluctuations, network cell changes, or battery consumption.
- How frequent do **mobile devices and back-end servers interact**? An app for a magazine requires one data download per issue. Then, the app runs without the network. In contrast, intra-day trading apps require a stable server connection.

4.3 User Bases

Google Play and Apple iTunes offer apps for the **open market**. The user base is the entire world. Other apps serve company-internal users. For these apps, two user bases are relevant: limited device variance and device restricted. **Limited device variance** apps run only on defined device types and operating system versions. Users can choose between different, but heavily restricted options, e.g., Apple iPhone 4 8 GB

Apple iOS 6.1, Samsung Galaxy S3 Android 4.1.2, and Blackberry Q10. This allows users to express their individuality. At the same time, the number of test configurations is limited. Company-specific email clients for mobile devices are an example. **Device Restricted** apps go further and have only one (or few, very similar) device option(s). Examples of restricted devices apps are apps for train conductors or and apps for sales personnel.

The chosen user base—open market, limited device variance, device restricted—affects the test strategy (Table 1). An open market app requires device testing with many devices and variants of operating system versions. Moreover, they must proactively monitor the market for new devices and operating system versions. Company-internal apps require less device testing because they are tested only for their defined devices and operating system versions. Only if the company rolls out new hardware or software do these apps need to be tested with them.

Table 1. User base

Customers	Devices	OS Versions
Open Market	All	All
Limited Device Variance	Few to all	Few to all
Restricted Devices	One	Restricted

4.4 Characterization for Test Strategy

Table 2 guides test mangers through the process of characterizing an app and its usage, as discussed in the previous subsections. This characterization is the first step towards defining test case in mobile testing.

Table 2. Test Strategy Relevant Usage Dimensions

Dimension	Values
Architecture	☐ Standalone native app ☐ Mobile web page ☐ Integrated native app ☐ Launcher app
Category	☐ Bored Now ☐ Urgent Now ☐ Repetitive Now
App type	☐ Interactive ☐ No-nonsense
Usage length	Seconds to hours: ________________
Mobile device / server interaction	☐ Never ☐ Short ☐ Frequent ☐ Standing
User base / diversity	☐ Open Market ☐ Limited Device Variance ☐ Restricted Devices

5. TEST EXECUTION

5.1 Device Infrastructure

5.1.1 Device Types

The test strategy affects the devices used for the tests. Device options include emulators, local devices, a private device cloud, or a public device cloud. The choice can differ among test types, such as functional tests or device tests.

Emulators simulate a mobile device on a laptop or PC. One example is the emulator of the Android SDK. Testers can define the screen size, memory size, SD cards, and other factors. The emulator even simulates cameras and incoming text messages. [6]

A **local device** is a mobile device "owned" by a tester. It might or might not be attached locally to his or her PC. The tester can install apps on the device and test them manually. He can use gestures, eye tracking, and other motions. At the end, the tester writes a manual test report. Tool support is possible if the tester has recording software on his or her mobile device or if the device is connected to a PC. The latter allows remote control of the mobile device and automated tests, for example, using Experitest Manual and Experitest Automation [7].

A **private device cloud** is a company-internal, centralized device pool. A tester (or a test automation tool) selects a device, connects to it, and starts testing. He or she can use the phone in the cloud as a local device, such as to make outside calls. The only limitations are physical interactions such as gestures or location changes. Picture 1 shows the onsite private device cloud installation at Swisscom based on Perfecto Mobile [8] (see Section 6 for details). Test centers benefit from private device clouds because they are easy to integrate with other existing tools.

Public device clouds consist of a pool of devices, but service providers run them and offer access to them through the Internet. Samsung has an interesting offer, its Remote Test Lab [9], which provides many old and new Samsung devices that are free to use for up to five hours a day. In general, the challenge lies in smooth integration into an existing test infrastructure.

Picture 1: Swisscom Installation. One rack with device pools (left) and one slide-in module for a device (right)

5.1.2 Test Types and Device Types

Front-end tests and **front-end–back-end-integration tests** focus on functional correctness. Code coverage matters. Local devices, private device clouds, and emulators are good options. Testing physical interactions and gestures requires a local device. Testing network effects require a local device or a private device cloud. Emulators also work in all other cases. Public cloud devices work but have more overhead. Using a private cloud for functional tests has two benefits. First, testers simply switch to another device if one does not work. Second, testers can rely on a central device team that deals with all device issues.

Device tests validate whether a mobile app works with the relevant devices, operating systems (OS), and OS versions. In this context, emulators can cover GUI aspects such as screen sizes. All other tests require real devices. A private cloud eases the automation and running of test cases in parallel on many devices. This is a must for open market apps. Company-internal apps need to be tested on far fewer devices. A local device might be an option for tests but also burdens the testers

with device maintenance. **Monitoring OS upgrades and new devices** is similar. Local devices can work for company-internal apps. A private device cloud works for all apps, whereas public device clouds are attractive for tests with exotic devices. **Testing in the wild** requires local devices that must be carried on trains, trams, or to the tops of mountains.

The right choice for **bug analysis and retesting** depends on the type of bug. For *functional bugs*, testers can work with a local device or with a private device cloud. Even emulators work if the bug does not relate to network aspects or physical interactions. *Compatibility bugs* demand a private device cloud given the importance of quick tests on a device pool. For exotic devices, testers could use a public device cloud. Table 3 summarizes and compares the options.

Table 3. Device Supply Methods and Purpose

		Emu-lator	Local Device	Private Device Cloud	Public Device Cloud
Front end	*With physical interaction*		+		
	Without physical interaction	+	+	+	O
Back end		No mobile devices involved			
Front-end/Back-end integration		O	+	+	
Device testing		O	(+)	+	
Testing in the wild			+		
Monitoring OS upgrades / New devices			(+)	+	(+)
Bug analysis / retesting	*functional*	(+)	+		
	compatibility			+	+

5.1.3 Cost Factors

Costs also affect the device strategy. There are four cost types: device costs, device maintenance costs, automation costs, and process integration costs.

Device costs are the direct costs for buying or renting phones and tablets. Mobile testing requires an up-to-date pool of test devices. For example, Perfecto Mobile suggests a pool of ten devices for 50% market coverage. For 80% market coverage, approximately 30 devices are needed. Ten should be replaced each quarter [10]. Such a number of devise incurs costs regardless of whether the devices are bought or temporarily rented in a cloud.

Device maintenance costs include staff costs. Many small tasks create work and, thus, costs: devices hang and crash, OS upgrades fail, devices have to be jail broken, apps must be installed for the test automation tool, SIM cards have to be ordered. Costs are obvious for a central device team that maintains a private device cloud. Costs may also be hidden, such as when testers maintain their own devices. Then, they invest time in device issues instead of testing. If their device(s) do not work, they are blocked from testing.

Automation costs cover writing and maintaining test scripts, training employees, and licensing tools. The latter costs vary from zero to thousands of dollars for one tester workplace.

Process integration costs include the costs to integrate mobile testing tools into an existing infrastructure, such as building new interfaces. These costs also cover the manual extra work needed because of missing integration.

5.2 Test Automation

5.2.1 Reasons to Automate

There are two reasons to automate mobile test cases: to ensure minimal functional coverage and to achieve scalable test configuration coverage.

Minimal functional coverage is a safety net. The test scripts cover the basic features of the app and run, for example, each night. They cover front-end tests and front-end–back-end-integration tests. A basic test infrastructure consists of a PC running the test script, one local device, and an automation tool. Selenium Web Driver [11] and Robotium [12] are sample tools for mobile web apps and Android apps testing.

Scalable test configuration coverage checks whether the app runs problem-free on all relevant devices and OS versions. Test scripts must run in parallel on multiple devices, which requires a private device cloud with parallelization features. We describe our Swisscom solution in Section 6 as one sample solution that implements parallelism on top of Perfecto Mobile.

Although test automation can reduce costs, it is not always the first goal for mobile testing. Many test cases are executed many times, such as on twenty devices for device testing. No human tester can maintain focus for such a long period; thus, automation is required.

5.2.2 Implementing Mobile Test Automation

A test automation solution must cover four aspects: test scripts, a connection between the PC and the mobile device, a remote control mechanism for the device, and an interaction strategy for the mobile device GUI (Figure 6).

The chosen solution affects the **test script** language. For example, Perfecto Mobile has its own scripting language. It provides also an adapter to HP Quick Test Professional. Both, Selenium and Robotium, rely on JUnit test cases.

When scalable test configuration coverage is the main aim, the test scripts must run on multiple devices and potentially on various OS and OS versions. This requirement affects the **connection** between a test PC and the mobile device. First, a direct connection can exist from the PC to the device, such as through a USB cable. Second, an indirect connection can exist that acts as a switch between various PCs and a large device pool. Such a setup helps during device tests to run one script on various devices or when many testers are working in parallel.

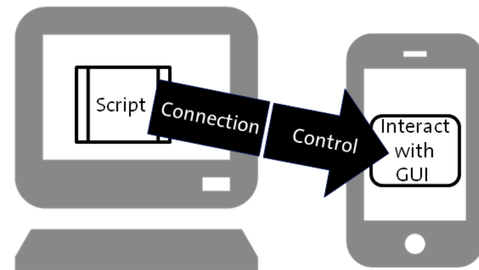

Figure 6: Test Automation Components

When a PC **remotely controls** a mobile device, the PC sends commands. The mobile device treats these commands as user input through a touchscreen. Four implementation options exist.

- Direct control, such as through USB. A cable connects the PC with the mobile device to send commands. This setup works for Android devices but requires a jailbreak for iOS devices.
- Installing a receiver / remote control app on an iOS device; the app receives commands from the PC and executes them locally via USB or WLAN.

- Remote-controlled web browser. A modified web browser on the device acts as a receiver for commands from the PC. Perfecto Mobile offers this option to enable web app testing without having to jailbreak the device.
- Instrumentation of apps. Code is added to the app before testing, allowing remote control of the app.

GUI interaction requires additional explanation. A PC can use five main commands to control an app on a mobile device:

1. Searching for text or pictures to check for or to click on them;
2. Typing text into a GUI element, such as an email address;
3. GUI element interactions, such as through select boxes, lists, or wheels to define numbers;
4. Events, such as incoming calls or text messages; and,
5. Physical interaction, for example, device rotation, motion sensors, gestures, and a camera.

Test automation tools support the first three or four aspects. For aspect five—physical interaction—automation tools support rotations from landscape to portrait and vice versa. For example, gestures are better tested with manual (exploratory) tests.

Test scripts must always reference the exact GUI element using an identifier. The identifier must fulfill four requirements to keep maintenance low [13]:

- Uniqueness;
- Screen or window size has no influence;
- Language-independent if the app supports more than one language; and,
- Stable, even if GUI changes.

Two preferred options exist for mobile apps. Option one is that test scripts refer to the IDs of the GUI elements, which is standard for testing normal web applications but is not always possible for mobile apps. Additionally, GUI IDs require the IDs to remain unchanged, which can be an issue for outsourced software development. Option two is OCR. Test scripts refer to text fragments, which the OCR searches for on the screen. This option is very stable and, thus, is the option we primarily use.

Two more options are suited for front-end tests and front-end–back-end-integration tests. Both options require always running tests on the same device. Layout and screen size must not change. Then, test scripts can search for pictures (for example, to click on them) or testers can click on fixed-screen coordinates (for example, click at <300,200>).

5.2.3 Limitations for Test Automation

Test automation has limits. First, developers must ensure that app design and code ensure testability. The identifiers used in the test scripts must not change. If one test script is used for an iOS and for an Android app, the interaction design must be the same. Both apps must use similar GUI elements. If not, the need for two tests will double the testing costs.

Additionally, automating tests differs for no-nonsense and interactive apps. Test scripts for no-nonsense apps consist of a few clicks and some text input, making them easy scripts. Interactive apps have nice pictures and animation. Sport simulation apps, such as a ski race app or other games, might rely on physical interaction, which is more difficult to put into test scripts. Thus, the test coverage achieved by automation is higher for no-nonsense apps than for interactive apps.

5.3 Test Environments

Larger IT departments segregate production servers from development and test servers. They are in different zones. Thus, network connections between production servers and test servers are difficult or even impossible. This impacts the testing of mobile apps that access back-end services. The apps run on a mobile device. That mobile device is in the Internet but should be in a test zone to be able to reach the test server.

Three possible solutions exist (Figure 7). Option one is to use an emulator. The mobile app runs on an emulator on a PC in the test zone. Option two is to set up a WLAN for the test zone. The mobile device connects to the test server via the WLAN. Both an emulator and the WLAN do not support "in the wild" tests. Testing on mountains or when commuting between the office and home is not possible. If such tests are important, access point names (APNs) can be utilized [14][15]. APNs allow a physical device to the company's testing zone. Certainly, this requires services from a telecom company.

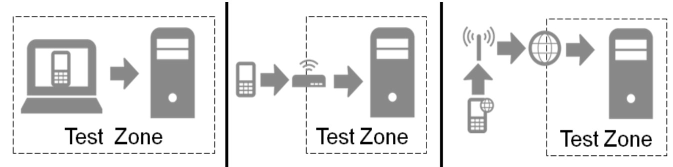

Figure 7: Mobile Testing and Test Zones: Emulator (left), WLAN (middle), access point solution (right)

6. MOBILE TESTING AT SWISSCOM

This section describes Swisscom IT Service's mobile testing framework and one project for which it was used. The project is a web shop for users with mobile devices. Table 4 shows the usage dimensions.

The aim of the test automation was to develop a safety net for:

- Regression tests with quick turnaround times;
- Finding functional flaws;
- Checking the layout for various screen sizes; and,
- Device tests, i.e., to run scripts on a large pool of mobile devices.

An additional need was to enable business users without test training to define simple, linear test scripts.

Table 4. Usage Dimensions for Sample Project

Dimension	Values
Architecture	☐ Standalone native app ☑ Mobile web page ☐ Integrated native app ☐ Launcher app
Category	☐ Bored Now ☑ Urgent Now ☐ Repetitive Now
App Type	☐ Interactive ☑ No-nonsense
Usage Length	Seconds to hours: *up to 30 minutes*
mobile device / server interaction	☐ Never ☐ Short ☑ Frequent ☐ Standing
User base / diversity	☑ Open Market ☐ Limited Device Variance ☐ Restricted Devices

Our solution (Figure 8) is based on a Perfecto Mobile private cloud, which has a rack for a large pool of mobile devices. It is located on Swisscom premises. Perfecto Mobile allows writing and executing test scripts and provides an interface through which to control the devices. It implements OCR-based search for text fragments on screens, screenshots, and reports etc. For better integration and cost-effective

test case specification and execution, we implemented the Swisscom Framework on top of Perfecto Mobile.

The Swisscom Framework implements keyword-driven testing [13] with pseudo–parallel test execution. The framework picks a step from the test script and sends it first to device one, then subsequently to device two and so on. Then, the framework picks the next step and, again, sends it to the first device, then to the second. This pseudo-parallelization scales nearly with the number of devices attached to the test environment.[3] The reason is simple. Many steps—calls to the server back end via mobile Internet or rendering web pages—need a few seconds, which is enough to send commands to the next devices.

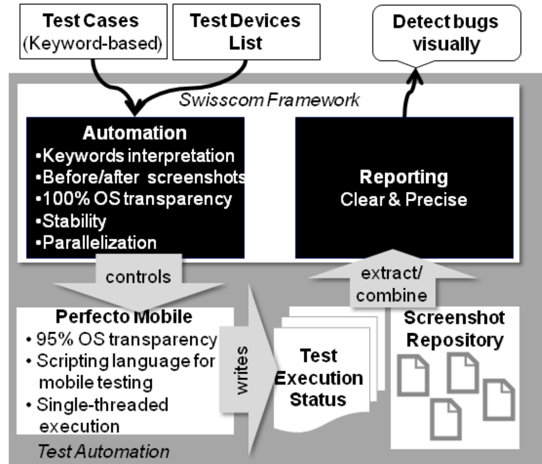

Figure 8: Swisscom IT Services Mobile Testing Framework

Keyword-driven testing means that a data table stores the various steps of the script. As an example, the data table could look like the following:

STEP	COMMAND	PARAMETER 1	PARAMETER 2
1	Search	Swisscom	-
2	Rotate left	-	-
3	Type In	Name	Klaus

Then, the framework first searches for the word "Swisscom" to determine whether, for example, a page has been loaded. Then, it rotates the device. Next, the script types in "Klaus" in the field "Name."

Our keyword-driven solution is based on very similar Perfecto Mobile commands that provide the functionality to search for names or for typing in text in certain fields etc. However, the various mobile operating systems require slightly different handling, such as for scrolling. Our framework hides this difference and, thus, scripts run without any modification e.g. on Android and on iOS.

After the test case definition, implementation, and execution, project managers and testers receive test reports. The report helps them quickly understand the tests that succeeded and those that failed. Again, Perfecto Mobile has a solution, but it did not work with our keyword-based test scripts layer. The test reports had more than 500 pages; thus, we had to implement our own solution. Our framework makes screenshots of the devices before and after each step. It matches them with the keywords respectively test case steps stored in the data table. The output consists of HTML pages as shown in Figure 9. Each step is one row. The pre- and post-screenshots of the devices are in the columns to the right, enabling a human to see within one second:

- Whether the test case runs on all devices; and,
- Whether the layout is correct on all devices.

Our HTML page reports turned out to be the most valuable for our test managers.

swisscom Test Report

Befehle	Parameter1	Parameter2	Step	Apple iPhone 4 Before and After	Samsung Galaxy S IV Before and After
openDevice			1		
home			2		
openurl	www.swisscom.ch		3		
navigation	Mobile		4		

Figure 9: Sample Test Report for a Mobile Web Shop Test Case. Pseudo-parallel execution on Apple iOS and a Samsung Android device, primarily the test case start sequence. This report allows for a quick comparison, which helps uncover layout issues.

7. RELATED WORK

Mobile testing received some interest in academia a while ago. Most of the research followed an engineering paradigm: finding a "better" technical solution than the existing ones.

For example, Zhifang and Bo developed a test tool for mobile app testing and used it in a sample case. The tool implements many ideas found in today's commercial tools: image comparison, OCR, or local agents on mobile devices for remote control [16] [17]. Very close to today's problem is also the paper by Delamaro et al. in 2006 on white-box testing of apps [18]. Based on instrumentation, they see the actual test traces and measure the test coverage. Their tool was one of the first to run on a real device rather than an emulator.

Mahmoud and Maamar [20] look at the complete development cycle of mobile apps. For testing, they list four main topics: implementation validation, usability testing, network performance testing, and server-side testing. For usability and network, they even provide concrete subareas that must be tested, for example, test readability or the masking of sensitive data. The work of Muccini [19] is similar, but he takes a more structural perspective. A key point is to distinguish between web applications, which are rewritten for mobile devices, and context-aware apps. The latter includes, for example, location or user information in their processing, which affects testing.

Jha [21] compiled an exhaustive risk catalog for mobile apps. The main areas are product elements (for example, code, user interfaces, and data), operational quality (for example, reliability, security, and scalability), development quality such as testability and maintenance, and project management.

[3] Obviously, this requires that test scripts are linear. This is sufficient for all cases we observed up to now.

The work of Satoh [22] is more on the pure research side. He presents a framework that allows testing mobile apps and simulating physical location changes with mobile agents.

In addition to work on the functional testing of mobile apps, which is the focus of this paper, a lot of work exists on mobile app security and usability, such as by Gilbert et al. [23] and Kaikkonen et al. [24].

The topic of mobile app testing has recently received more attention by testing professionals in the IT industry, as revealed by two indicators. First, the number of tools from commercial vendors such as Perfecto Mobile, Experitest, Jamo, Soasta, and Keynote DeviceAnywhere, is growing. Second, testing magazines for professional testers are publishing issues exclusively on mobile app testing [25]. Articles discuss whether physical devices or emulators are better [26]. They discuss best practices from their experience of testing concrete mobile apps [27][28]. Other articles take a strategy and management perspective [29]. All of these articles reflect practical experience and look at methodology questions. However, their aim is less on providing a holistic perspective on mobile testing (as this paper seeks to) and more on reporting their experiences.

8. SUMMARY

This paper answered two questions: what is special about mobile testing and which tests are needed when. In short, mobile testing has two new test types: tests in the wild and device tests. *Tests in the wild* are exploratory tests. Testers use real devices and board trains, hike mountains, or dance in clubs. They test the app in the same context in which users use them. *Device tests* focus on whether an app runs on all relevant devices and operating system versions. Device tests are part of the development process. After the release of an app, device tests continue because the app must work also with new devices and OS versions.

A test strategy for mobile testing defines the areas that must be tested. Testers must know the architecture (for example, a native client/server app), the usage context (for example, urgent now), and the user base (for example, open market). Since most projects are agile today, they require (some) test automation. This is the only way to test frequently against the latest build, such as every night. Since many mobile apps should run on many devices and operating system versions, device tests are needed. Device tests run one or a few test scripts against various devices. This demands a test infrastructure: a private cloud, parallelized test case execution, and—potentially—test cases running on various operating systems. We achieved these aims using our Swisscom Framework. It builds on a Perfecto Mobile private cloud.

The user feedback in app stores helps improve an app and meet better customer needs. At the same time, feedback represents a reputation risk if the app is buggy or fails to meet customer needs. We evaluated more than 1,000 single ratings and found that many apps do not achieve a basic level of quality. Users should be able to install and start an app without crashing. A new app release should be more stable and not buggier than an old release. Many apps fail on this basic quality level today. Thus, mobile testing needs many improvements, though, certainly, an app's market triumph requires more: understanding the user, design and usability, and creativity and innovation.

Acknowledgment. *The author thanks Hans-Joachim Lotzer for the valuable input during the past few months. Special thanks also to Robert Kälin and Cedric Gmür for their work on the Swisscom Framework and to Christoph Moser for his continuous support on the infrastructure side.*

9. REFERENCES

[1] http://en.wikipedia.org/wiki/Smartphone

[2] Klaus Haller, Konrad Schlude: *How Scrum Changes Test Centers*, Agile Record, August 2013

[3] C.K. Prahalad, V. Ramaswamy: *Co-opting Customer Competence, Harvard Business Review*, January 2000

[4] D. Graham et al.: *Foundations of Software Testing, Thomson Learning*, 2008, London, UK

[5] S. Wellmann: 2007. *Google Lays Out Its Mobile User Experience Strategy*. Information Week, April 11, http://www.informationweek.com/mobility/business/google-lays-out-its-mobile-user-experien/229216268, last retrieved April 9, 2003.

[6] http://developer.android.com/tools/devices/emulator.html

[7] http://www.experitest.com/

[8] http://www.perfectomobile.com/

[9] http://developer.samsung.com/remotetestlab

[10] Perfecto Mobile: *How to Set the Right Strategy for Selecting Devices for Your Enterprise's Mobile Testing*, White Paper, 2012

[11] Selenium Web Driver. http://docs.seleniumhq.org/projects/webdriver/

[12] Robotium. http://code.google.com/p/robotium/

[13] R. Seidl, M. Baumgartner, Th. Bucsics: *Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken*, Dpunkt Verlag, Heidelberg, Germany, 2011

[14] http://en.wikipedia.org/wiki/Access_Point_Name

[15] *Digi Connect® Application Guide Cellular IP Connections* (Uncovered), 2005

[16] L. Zhifang, L. Bin, G. Xiaopeng: *Test Automation on Mobile Device*, AST'10, May 3rd–4th, 2010, Cape Town, South Africa

[17] J. Bo, L. Xiang, G. Xiaopeng: *MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices*. 2nd International Workshop on Automation of Software Test (AST '07), Minneapolis, MN, 20–26 May 2007

[18] M. E. Delamaro, A. M. R. Vincenzi, J. C. Maldonado: *A Strategy to Perform Coverage Testing of Mobile Applications*, International Workshop on Automation of Software Test (AST '06), Shanghai, China, May 23–23, 2006

[19] H. Muccini, A. Di Francesco, P. Esposito: *Software Testing of Mobile Applications: Challenges and Future Research Directions*, 7th International Workshop on Automation of Software Test (AST '12), June 2–3, 2012, Zurich, Switzerland

[20] Q. H. Mahmoud, Z. Maamar: *Engineering Wireless Mobile Applications*, International Journal of Information Technology and Web Engineering, Volume 1, Issue 1, 2006

[21] A. K. Jha: *A Risk Catalog for Mobile Applications*, Master's thesis, Florida Institute of Technology, Melbourne, Florida, 2007

[22] I. Satoh: *A Testing Framework for Mobile Computing Software*, IEEE Transactions on Software Engineering, Vol. 29. No. 12, December 2013

[23] P. Gilbert et al.: *Vision: automated security validation of mobile apps at app market*, 2nd Int. Workshop on Mobile cloud computing and services, MCS '11, June 28, 2011, Bethesda, Maryland

[24] A. Kaikkonen et al.: *Usability Testing of Mobile Applications: A Comparison between Laboratory and Field Testing*, Journal of Usability Studies, Issue 1, Volume 1, November 2005, pp. 4–17

[25] Testing experience – *Special Issue on "Mobile App Testing,"* September 2012

[26] R. F. Alvarez: *Testing on Real Handset vs. Testing on a Simulator – the big battle*, testing experience, September 2012

[27] D. Knott: *Best Practices in Mobile App Testing*, testing experience, September 2012

[28] J. Jacob, M. Tharakan: *Roadblocks and their workaround while testing Mobile Apps*, testing experience, September 2012

[29] K. Rayachotti: *Mobile Test Strategy*, testing experience, September 2012