Testdroid: automated remote UI testing on Android

Jouko Kaasila
Bitbar
jouko.kaasila@bitbar.com

Denzil Ferreira
MediaTeam Oulu
University of Oulu
denzil.ferreira@ee.oulu.fi

Vassilis Kostakos
MediaTeam Oulu
University of Oulu
vassilis@ee.oulu.fi

Timo Ojala
MediaTeam Oulu
University of Oulu
timo.ojala@ee.oulu.fi

ABSTRACT
Open mobile platforms such as Android currently suffer from the existence of multiple versions, each with its own peculiarities. This makes the comprehensive testing of interactive applications challenging. In this paper we present Testdroid, an online platform for conducting scripted user interface tests on a variety of physical Android handsets. Testdroid allows developers and researchers to record test scripts, which along with their application are automatically executed on a variety of handsets in parallel. The platform reports the outcome of these tests, enabling developers and researchers to quickly identify platforms where their systems may crash or fail. At the same time the platform allows us to identify more broadly the various problems associated with each handset, as well as frequent programming mistakes.

Author Keywords
Remote testing, User Interface, Mobiles, Usability, Performance, Applications, Fragmentation.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Measurement, Performance, Reliability.

INTRODUCTION
Developing mobile software and conducting large-scale experimental studies with real users has become increasingly easy with the advent of app-stores that simplify the deployment of applications to large numbers of users. On closed platforms, like Apple's iOS, this process and the hardware on which the software runs are tightly controlled, with associated benefits and drawbacks. However, for open platforms such as Android the benefits of reaching easily a large number of users and manufacturers comes at a cost: it remains challenging to develop an interactive application that will run as intended on a variety of handsets from different manufacturers, as currently there are multiple Android handset manufacturers, each one with their own customized release of the operating system, and slightly different features and hardware availability.

Fragmentation is manifested in both software and hardware, where applications typically crash or behave differently across handsets, thus hampering usability and performance [10]. Testing of mobile interactive applications has thus become increasingly challenging since testing on a single handset is not enough: the app-store paradigm almost enforces that an application should run well on multiple types of handsets. Developers and researchers therefore struggle to provide support for their applications and deployments, as fragmentation increases the amount of time required to test APIs and devices. Comprehensive testing requires that an application is tested on multiple handsets, something which can be difficult, expensive, and time consuming.

In this paper we present Testdroid, an online platform for conducting automated scripted tests of interactive applications on physical mobile handsets running Android OS. Unlike existing systems that support such a diversified testing approach, Testdroid is able to fully automate the testing stage while conducting tests that tries to mimic human behavior and interaction with the applications.

RELATED WORK
In the early days of mobile development, testing was conducted in closed environments usually by device manufacturers [1]. The rise in popularity of application stores and software platform development kits unleashed a new set of opportunities for mobile application developers, but also new challenges. These include mobile devices' rapid evolution and use of various standards, protocols and network technologies, applications being deployed to various platforms and hardware, documentation that is often incomplete, and market trends that change frequently.

To overcome these challenges, we require tools and techniques to evaluate mobile applications in a more realistic and contextual manner. Researchers have proposed methods to collect the context of application usage directly from the users [3, 11], which in turn can drive application development and testing. Another alternative is the usage of logging software installed on the mobile devices to capture users' application and device usage [9]. Although useful, these techniques require time, resources and recruiting potential application users, hence motivating the need for tools that support automated usability evaluations and testing.

Automated testing of user interfaces is not novel; test suits exist for a variety of technology trends including desktop usability [13], desktop content adaptation [8], mobile content adaptation [14], and web interfaces [2]. Beyond the desktop, in mobile paradigms it has been suggested that automated tests are a cheaper and quicker alternative to field tests [12]. A fair amount of research has focused on facilitating UI testing on mobile platforms. Satoh's framework [17] emulated a mobile device at the application-level by implementing a mobile agent that simulates various network conditions. Kostiainen et al. [15] provide a framework for comparative usability testing, useful to test the optimal sequence to perform a certain task. Similarly, MobileTest [5] is a tool for automatic black box test of software on mobile devices. It adopts an event based testing approach to simplify the design of test cases and provides reusability of tests using software agents. These approaches, however, require source-code modification, as well as effort for managing the multitude of handsets required for testing.

On the other hand, Weiss & Zduniak [18] propose a capture-replay testing tool that allows researchers to remotely and interactively test their application on many handsets. A limitation of this approach is that it requires that the developer has multiple handsets available simultaneously, and also requires time to test different use cases manually. In an attempt to facilitate the automation of testing, the Handheld device User Interface Analysis [4] testing framework allows recording and editing user actions using XML, comparison between predefined use case and a real user usage, and hotspot analysis to identify most used sections of an interface. However, in this approach it is tedious to construct and edit UI tests.

Industry has also been active in this domain. For example, two US patents (US 6.526.526 B1 from IBM; US 6.662.217 B1 from Microsoft) have been filed on remotely distributed and automated usability testing. TestQuest [6] is a commercial automated testing solution that provides support for several electronic devices by simulating in software the presence of a user. A main drawback of this system is that it only performs image and text analysis on the output. It does not provide support for capturing why did a test fail in the first place in. A similar system, Digia QualityKit [7], allows software developers to record and run tests with critical mobile device constrains, but does not support remote execution. Conversely, the PerfectoMobile.com platform allows for scripting and automated execution, but relies on optical character recognition using cameras pointed at the handsets to control a device, which can be problematic and resource intensive.

In summary, a number of projects have looked at automating UI testing for mobile devices, but still a number of limitations exist in these approaches. In some cases it is not possible to automate testing on multiple handsets, and when this is possible it is typically hard to define the tests in the first place, or the developer/researcher needs to maintain a large number of handsets for testing purposes.

TESTDROID DESCRIPTION

The Testdroid platform allows developers and researchers to test their software on a variety of handsets, and obtain data about the performance of their software on these handsets. The tests are intended to mirror human input timing, and therefore are tailored for interactive applications. Conducting tests is a three-stage process: i) definition of the UI tests, ii) execution of the UI tests, and iii) obtaining the results.

Definition of the UI test

To facilitate the definition and execution of tests, we have developed a "test recorder" for the Android platform. Researchers can download the test recorder software on their development machines, and use it to record test cases for their software. The test recorder utilizes Robotium [16] library for defining the tests to be executed. The main advantage this approach is that the source code is not necessary in order to define UI tests. A developer can simply use the APK file of a project and create automated Robotium tests, while interacting with their own device.

To define the tests, the developer is able to manually navigate the various menus in the application, and the system keeps track of which menus or objects were interacted with. The recorder accesses Android's view hierarchy to inspect the user inputs, and provides three ways to capture the UI events: by considering the (x,y) coordinates of a click event, the text label of the object (e.g. "OK" button), or the unique id of the object. Furthermore, the developer is able to interject special events into the test script. For example, the developer is able to define a moment in time when a screenshot of the application should be recorded. This will ensure that as part of the test results screenshots are provided. In addition, our system gives the option to capture the delay time between user actions during recording and then repeat the same user actions with exactly same time intervals during playback. In this sense our system tries to mimic human use and allows for more realistic testing.

Execution of UI test

To execute the recorded tests, the test script is translated into automated Robotium tests, and for example can be executed as JUnit tests within Eclipse. A developer can physically plug in a phone to the development machine, and observe the tests running on the phone itself, with the JUnit results available on the development machine. In the absence of a physical device, the developer can run the tests within a phone emulator. However, to allow developers to diversify their testing, we have built an online system that allows developers to upload their programs, along with the test scripts, to be executed on a palette of physical devices that we make available for testing. Our system currently has over 100 different handset models from 9 different brands, 16 different Android versions and 9 different screen resolutions (Figure 1).

Once a developer uploads their software and tests, they can run their tests on all handsets or a subset of devices. The developer can restrict the devices where the test will be executed by the following criteria: Android version; API level (minimum supported Application Programming Interfaces); CPU (speed and number of cores); Device groups (phone or tablet); Internal Storage; RAM; Screen Resolution; Bluetooth; Camera (resolution); Manufacturer; UI overlay (*i.e.,*HTC Sense, Touchwiz);

Each physical handset has an associated queue of tests, which varies depending on the number of active users and the tests submitted. This means that tests are not immediately executed but they remain in queue until the desired handset becomes available. Once a particular test begins execution on a handset, our system stores the OS-level execution log, as well as the various defined screenshots by capturing the display *screenbuffer* from the operating system. The test script can use one of the three defined techniques for re-creating the UI interaction. These have different benefits and drawbacks. For example, relying on (x,y) coordinates is appropriate for graphic-intensive games: our system scales the (x,y) coordinates so that the tests can be run on various screen resolutions, a functionality which we call *ScaleClicks*. Relying on the object ID or the object text makes is more appropriate for running tests of form-based applications on a variety of handsets with different form factors and screen sizes. Driven by the test script, our system emulates the appropriate object events, as it emulates a button click or a screen touch with the designated object ID.

Obtaining the UI test results

Developers can create online accounts and associate their software and tests with an account. This developers can batch submit tests, and return after a few hours to check on the progress of these tests. Typically running a single test script for one application on all the available types of handsets takes a few minutes, but due to the popularity of our system the waiting time in a queue can become more than an hour. As tests on particular handsets are completed, the developer's account page becomes populated with results (Figure 2). For each test script we show the progress that was made, e.g. if the whole test executed, or whether the software crashed before completing the whole test. All execution and exception traces are shown, as well as the set of screenshots captured during the test.

DISCUSSION

We made our platform public in November 2011, and in the first two months it has attracted over 1000 developers and has conducted over 20000 tests. Considering the totality of the tests that have run on all our handsets, we have found that tests are more likely to fail on low-end devices, such as the Samsung Galaxy Gio and Samsung Galaxy Mini, while the same tests are more likely to succeed on other handsets.

The most common failure across all tests is that the application just fails to install on certain handsets. There are several reasons for this: the application may be using APIs that are not supported by a certain Android version, they may use direct screen access on a resolution that is not supported by the target device or simply trying to reserve more memory than the device has. Our observation is consistent with the comments on many applications in Android Market: most comments concern failed installations. Common mistakes include erroneous Android Manifest XML files, such as missing the minimum required Android version, which quite often the reason for failed installations. Another frequent cause is sloppy memory management, since many developers assume that Android garbage collection takes care of all memory related issues.

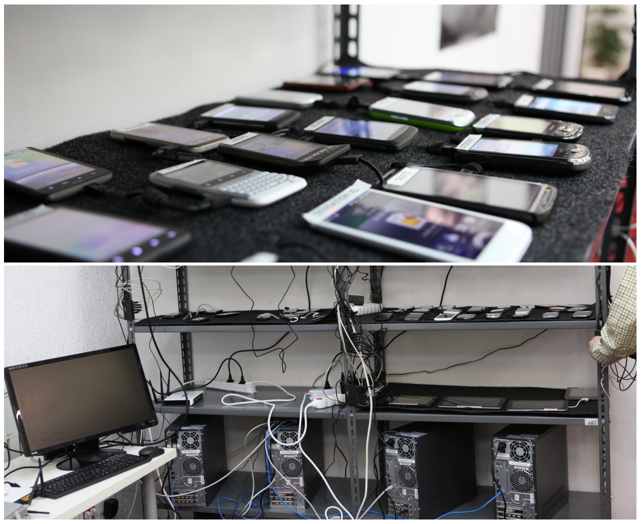
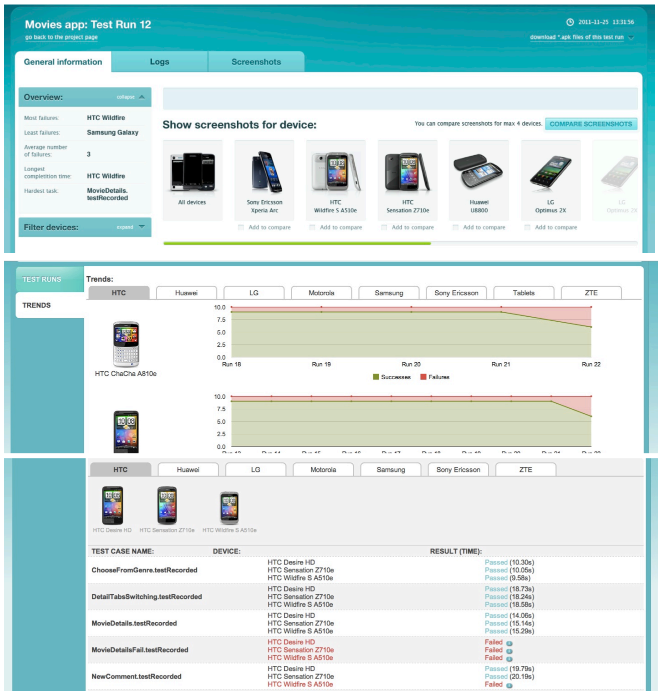


Figure 1. Top: snapshot of the physical handsets that execute the tests submitted online. Bottom: the handsets are physically connected to online servers that manage the test queue for each physical handset.



Most failures:	HTC Wildfire
Least failures:	Samsung Galaxy
Average number of failures:	3
Longest completition time:	HTC Wildfire
Hardest task:	MovieDetails. testRecorded

TEST CASE NAME:	DEVICE:	RESULT (TIME):
ChooseFromGenre.testRecorded	HTC Desire HD HTC Sensation Z710e HTC Wildfire S A510e	Passed (10.30s) Passed (10.05s) Passed (9.58s)
DetailTabsSwitching.testRecorded	HTC Desire HD HTC Sensation Z710e HTC Wildfire S A510e	Passed (18.73s) Passed (18.24s) Passed (18.58s)
MovieDetails.testRecorded	HTC Desire HD HTC Sensation Z710e HTC Wildfire S A510e	Passed (14.06s) Passed (15.14s) Passed (15.29s)
MovieDetailsFail.testRecorded	HTC Desire HD HTC Sensation Z710e HTC Wildfire S A510e	Failed Failed Failed
NewComment.testRecorded	HTC Desire HD HTC Sensation Z710e HTC Wildfire S A510e	Passed (19.79s) Passed (20.19s) Failed

Figure 2. Testdroid online interface. Top: the available devices for testing are presented. Middle: an overview of the success/ failure rate for batch tests across multiple handsets. Bottom: detailed results for the success/failure of tasks within tests.

In terms of the UI, Android's Density Pixel Independent interface implementation can cause some UIs to become unreadable if the form factor of the screen changes drastically, even causing buttons to disappear from the screen on certain occasions. Another recurring problem is the developers' assumption of network connectivity on the device, causing applications not to run or crash frequently.

Delays on the network are often not taken into account. For example, Android's BroadcastReceivers have a limited task execution time, thus if a request to a remote server takes longer than 10 seconds, the application is considered as unresponsive and is automatically closed by the operating system. Also allocating buffers as big as 4MB (*e.g.* for displaying photos) causes problems when the allocated and available heap-size memory is only 2MB on memory crippled (128MB of internal memory or less) Android devices.

Testdroid is by no means a "silver bullet" for automated application testing and has limitations of its own. For example, it is limited when testing applications that are depend of voice, gesture or movement input. Hence the context of use cannot be simulated, for example, replicating a noisy environment for voice input or a limited space for movement input, or clothing for gesture input. Also limiting is realistic network usage, as the testing hub is connected to a wireless connection (WLAN) for internet connectivity. The delays inherent in network data providers can indeed affect application performance, and are not reported accordingly. Mobility can lead to network degradation, but mobility is a testing factor which Testdroid cannot evaluate.

Our users online feedback suggest that device compatibility is one of the biggest challenges in development at the moment. Further, they request even further automating the way tests are executed. Surprisingly, the feedback we have obtained suggests that many test engineers are not very adept in programming. For example, our testing framework can have complex assert statements, or multitouch events, but requires technical skills to construct such tests. However most users rely on our test recorder, and in fact requested an even simpler way to test the application fully automatically. For this reason we have built an "auto-pilot" testing feature which only requires the application to be uploaded and the auto-pilot simply attempts to click on all buttons and objects in the UI reporting back the results.

CONCLUSION AND ONGOING WORK

This paper has presented Testdroid, an online platform for automating UI testing on a variety of mobile handsets. The main purpose of our system is to enable developers and researchers to easily construct test cases with no need for programming, as well as to provide a cost-effective way of having access to a variety of handsets without the overheads of maintaining those handsets. One of our future objectives is the ability to incorporate multitouch events in the tests. While our instrumentation has this technical capability, it is currently not possible to record these events during the test recording phase. In light of our users' comments about simplifying the test generation process, we believe it is important to make it possible for non-technical testers to record such tests. Finally, another future objective is to extend our testing to include a richer physical context, such as device orientation.

REFERENCES

1. Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkola, M., Koskela, J., Kyllönen, P., Salo, O. 2004. Mobile-D: An Agile Approach for Mobile Application Development. OOPSLA'04, pp. 174-175.
2. Atterer, R., Schmidt, A. 2007. Tracking the Interaction of Users with AJAX Applications for Usability Testing. CHI'2007, pp. 1347-1350.
3. Arhippainen, L., Tähti, M. 2003. Empirical Evaluation of User Experience in Two Adaptive Mobile Application Prototypes. MUM'03. pp. 27-34.
4. Au, R.W.F., Baker, S., Warren, I., Dobbie, G. 2008. Automated Usability Testing Framework. AUICpp. 55-64.
5. Bo, J., Xiang, L., Xiaopeng, G. 2007. MobileTest: A Tool Supporting Automatic Black Box Text for Softwre on Smart Mobile Devices. *AST'07,* pp. 8-14.
6. Bsquare, 2008. Bsquare offers industry leading automated testing tools. Retrieved on 19th January, 2012 from http://www.bsquare.com/bsquare-automated-testing-platform.aspx.
7. Digia QualityKit, 2007. Retrieved on 19th January, 2012 from http://www.digia.com.
8. Eisenstein, J., Puerta, A. 2000. Adaptation in Automated User-Interface Design. IUI 2000, pp. 74-81.
9. Ferreira, D., Dey, A., Kostakos, V. 2011. Understanding human-smartphone concerns: a study of battery life. Pervasive 2011, pp. 19-33.
10. Ham K.H., Park, Y.B. 2011. Mobile Application Compatibility Test System Design for Android Fragmentation. CCIS 257, pp. 314-320.
11. Kangas, E., Kinnunen, T. 2005. Applying User-Centered Design to Mobile Application Development. Communications of the ACM, Vol. 48, N.7, pp. 55-59.
12. Kaikkonen, A., Kallio, T., Kekäläinen, A., Kankainen, A., Cankar, M. 2005. Usability Testing of Mobile Applications: A Comparison between Laboratory and Field Testing. Journal of Usability Studies, 1, pp. 4-16.
13. Kasik, D.J., George, H.G. 1996. Towards Automatic Generation of Novice User Test Scripts. CHI'96, pp. 224-251.
14. Korpipää, P., Häkkilä, J., Ronkainen, S., Känsälä, I. 2004. Utilizing Context Ontology in Mobile Device Application Personalization. MUM'04, pp. 133-140.
15. Kostiainen, K., Uzun, E., Asokan, N., Ginzboorg, P. 2007. Framework for Comparative Usability Testing of Distributed Applications. Nokia NRC-TR-2007-005.
16. Robotium, 2010. It's like Selenium, but for Android. Retrieved on 19th January, 2012 from http://code.google.com/p/robotium/
17. Satoh, I. 2003. A Testing Framework for Mobile Computing Software. IEEE Transactions on Software Engineering, Vol. 29, N. 12, pp. 1112-1121.
18. Weiss, D., Zduniak, M. 2007. Automated Integration Tests for Mobile Applications in Java 2 Micro Edition. BIS 2007, pp. 478-487.