MobiTest:
A Cross-Platform Tool for Testing Mobile Applications

Ian Bayley, Derek Flood, Rachel Harrison, Clare Martin
Oxford Brookes University,
[ibayley, derek.flood, rachel.harrison, cemartin]@brookes.ac.uk

***Abstract*— Testing is an essential part of the software development lifecycle. However, it can cost a lot of time and money to perform. For mobile applications, this problem is further exacerbated by the need to develop apps in a short time-span and for multiple platforms. This paper proposes MobiTest, a cross-platform automated testing tool for mobile applications, which uses a domain-specific language for mobile interfaces. With it, developers can define a single suite of tests that can then be run for the same application on multiple platforms simultaneously, with considerable savings in time and money.**

Keywords – Mobile Application; Testing; MobiTest.

I. INTRODUCTION

The increasing prevalence of mobile applications (hereafter, apps) continues as the use of mobile phones becomes ubiquitous. By the end of 2010, there were an estimated 5.3 billion mobile subscriptions worldwide. In developed countries there are on average 116 subscriptions for every 100 inhabitants [1].

The applications that facilitate this, known as apps, are typically developed in a relatively short time span and on low budgets, often because the unit price of the app is very small or zero. This appears to greatly diminish the usability of many of the apps that are sold to users. This is unfortunate because a recent survey [2] has identified usability as being one of the most important factors when selecting a mobile app.

The annual cost of an inadequate infrastructure for testing in the US is estimated to range from $22.2 billion to $59.5 billion [3]. This cost is partly borne by users in the form of strategies to avoid and mitigate the consequences of errors. The remainder is absorbed by the software developers themselves, who have to compensate for inadequate tools and methods. The absorbed cost is even higher when one takes into account the damage that low software quality can bring to the reputation of the producer.

The problems noted above are further exacerbated by the need to target multiple platforms at once. In particular, a test suite for one platform must be rewritten for any other platform for which it is required. This problem has been addressed in the desktop domain through the use of the USer Interface eXtensible Markup Language (USIXML) [12], which allows developers to create a user interface using a common language that can then be translated to any platform.

This paper proposes a multi-platform testing tool that takes a description of the tests to be performed on an app and generates a test suite for every platform on which the app is to be tested. Consequently, the tests will only need to be specified once. They are described in a simple language, specialised to the domain of mobile devices. Here, we concentrate on GUI testing; but, the ideas expressed here could be extended to other forms of testing at a later date.

The rest of this paper is structured as follows. Section II details the related work of this research. Section III outlines our research objectives. Section IV provides an overview of the MobiTest tool. Section V highlights some of the challenges for implementation. In Section VI, the plan for progression is detailed and Section VII concludes this paper.

II. RELATED WORK

A. Software Testing

In *The Mythical Man Month*, Brooks [4] says that he assigns half of his development time for testing. This includes both component testing (of individual elements of the system) and system testing (of the complete system). His advice highlights the importance of testing; since, if it is not done adequately, the results can be very serious or even (in safety critical systems) fatal.

The waterfall model [5], one of the first software development methodologies, proposed that the testing phase should happen after the implementation phase has been completed. In contrast, Beck [6] proposes that the two phases be more tightly coupled, advocating the use of Test Driven Development (TDD).

TDD involves the writing the tests before writing the code, then executing the tests, and then fixing the code if the test has failed. This enables the developer to know exactly where the failing code is (as code is written in small increments). It also forces the developer to think continually about the design of the system. The collection of tests thereby accumulated can be run automatically whenever retesting is required.

George and Williams [7] found that TDD produced software that passed 18% more black box tests than software built using the waterfall model. However, this higher percentage comes at the cost of development time, which is longer by 16%.

Whichever approach is adopted, the use of automation reduces the time taken for testing. The alternative of manual testing is not only time-consuming, but also error prone.

 ISBN: 978-1-61208-230-1

B. Mobile applications

Mobile applications are different from traditional applications in several ways. They are adversely affected by the limitations of mobile devices, some of which have been highlighted by Zhang and Adipat [8] as follows:

- **Mobile Context**: When considering mobile applications the user is not tied to a single environment. The environment will also include interaction with nearby people, objects and other elements which may distract a user's attention.
- **Connectivity**: With mobile devices connectivity is often slow and unreliable and therefore will impact the performance of mobile applications which utilise these features.
- **Small Screen Size & Different Display Resolution**: In order to provide portability mobile devices contain very limited screen size and so the amount of information that can be displayed is drastically reduced.
- **Limited Processing Capability and Power**: In order to provide portability, mobile devices often contain less processing capability and power. This has the effect of limiting the functionality of applications for mobile devices.
- **Data Entry Methods**: The input methods available for mobile devices are restricted and require a certain level of proficiency. This problem increases the likelihood of erroneous input and decreases the rate of data entry.

Thus, mobile applications typically contain less functionality than traditional desktop applications. This is mainly due to the limitations of the platform, but is also affected by the context in which these applications are used. Mobile applications are designed to be used while on the move, and, as such, complex interactions are undesirable as this negatively affects usability.

In addition to this, mobile applications tend to be developed in a short period of time. This has been facilitated by the availability of better source libraries and development tools for creating mobile apps.

III. RESEARCH AIM

The aim of this research is to develop a mobile application testing tool that can be applied to all mobile platforms. As each mobile platform contains different components, it is necessary to first understand the components on each platform and how these relate to one another. In order to do this, the following two research questions (RQ) have been defined:

- RQ1: What components are available on each mobile platform?
- RQ2: Which of these components are common across the platforms?

To answer RQ1, a thorough examination of each of the mobile platforms will be performed. During this examination each of the components together with their associated events and attributes will be identified. These will then be compiled into a comprehensive profile of the platform.

Using the platform profile produced by RQ1, RQ2 will be answered through a comparison of these profiles. This RQ will aim to identify how the components on one platform relate to those on the other platforms. For example, the TextView component on Android [9] is equivalent to the Label component on the iOS platform [11].

Additionally, a third research question has been defined to investigate how these components can be combined into a platform independent testing tool.

- RQ3: How should these components be modelled in a platform independent testing tool?

By investigating the third research question, we will bring together all components from all platforms into a single platform independent representation. This is in contrast to USIXML as RQ3 incorporates all components not just a subset of them. The common components identified in RQ2 will have a single representation with a mapping to the concrete components used by the underlying platforms. Using this representation it will then be possible to construct the platform-independent testing tool, which we call MobiTest.

IV. MOBITEST

The MobiTest tool is designed to address some of the difficulties associated with the automated testing of mobile applications, by using a single set of unit tests to test the application on multiple platforms.

The initial version focuses on testing through the interface as this should be similar (although not exactly the same) on all platforms. In this way, the need for platform-specific code can be minimised. In this section we present a sample app on which MobiTest can be used, and then outline the proposed system architecture.

A. Sample Application

The Log In screen illustrated in Figure 1, which could be used on a number of applications, such as apps for logging medical data, invites the user to enter a username and password, which is checked against a database, and displays a message indicating whether these credentials have been accepted or not.

Assuming that ("ian", "brookes") is a valid (username, password) pair, a possible test of this app is as follows:

1. click in the username text field
2. press the keys 'i', 'a', 'n'
3. click in the password text field
4. press the keys 'b', 'r', 'o', 'o', 'k', 'e', 's'
5. click the OK button
6. assert the text component of the lower label is "password accepted"

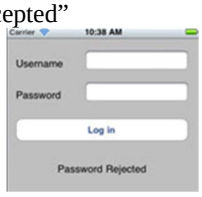

Figure 1. Sample Log In screen to be tested through MobiTest

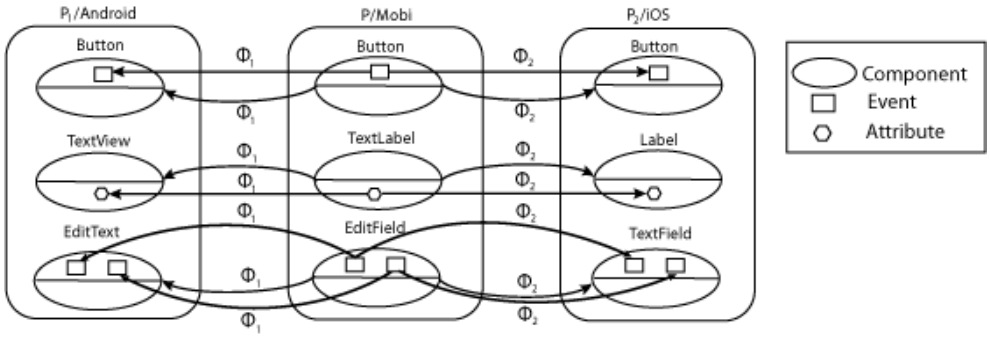

Figure 2. MobiTest System Architecture

When an assertion is false for any platform, this is reported to the user of the tool so that they can take action to correct the apparent error in the program. This is just one test that may be run on this application. In practice many more tests will be required. The benefit of MobiTest is that tests only need to be specified once. The test suites will be generated automatically for each platform, be it iOS, Android or Blackberry.

B. System Architecture

To generate automated tests for concrete platforms, such as Android or iOS, a virtual platform (called Mobi) will be defined. For Mobi, a number of GUI components will be defined through the answer to RQ3. For each such component, a list of valid attributes and events will be defined. Each concrete platform will have its components defined in a similar way. The available GUI components vary from platform to platform and even where the same component is available, the name may be different. Let P_i for i in 1..n denote the n different concrete platforms. To account for the naming differences, a function Φi can be defined that maps each Mobi component to its realisation in Pi. A similar function Φi maps each component attribute and event to its realisation in P_i. In the diagram below, n=2, P_1 represents Android and P_2 represents iOS.

Similarly, let p_i denote the set of actual components in the app written for platform P_i. Each version has six such components, as indicated in Figure 3.

Once a set of common components p is identified, a mapping φ_i from the components of p to those of p_i can thereby be deduced.

The tool MobiTest will operate as follows:

- **i.** from the layout XML files of each version of the app, MobiTest will deduce the mappings φ_i and insert them into an empty XML file `tests.xml`
- **ii.** the user will then add test cases in the form event$^+$ assert$^+$ where event and assert are given as tags with attributes and values in the normal manner and X^+ signifies one or more occurrences of X. It is anticipated that given the restricted nature of the language, GUI support can make this process exceptionally straightforward. This will be a major advantage, as it focuses attention on the interface which is unusual for conventional unit testing.
- **iii.** MobiTest will produce a test class for each platform. In it, for each test case, MobiTest will translate each event into a piece of code that triggers that event and, similarly, each assert into a piece of code that tests that assert. This will be done using the definitions of φ_i from the tests.xml file to identify the components and using Φ_i to determine the events and attributes
- **iv.** MobiTest will then run each test class on its associated platform, and present the test report to the user.

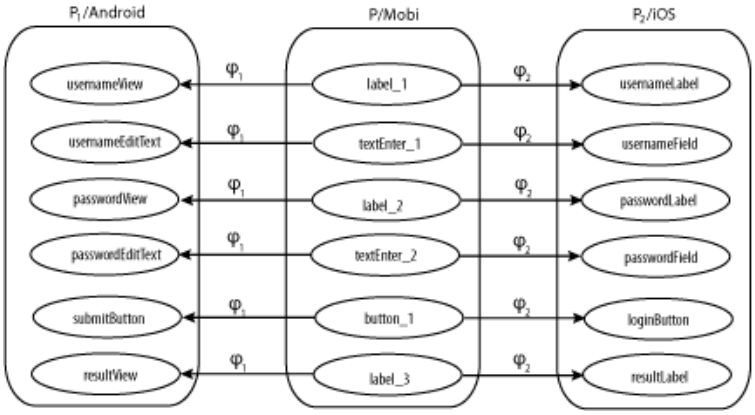

Figure 3. MobiTest view of the Log In screen

 ISBN: 978-1-61208-230-1

V. CHALLENGES

The creation of this tool presents a number of challenges:

- **Use of XML to specify components:** although iOS and Android can specify their layouts with XML, it may be that some platforms do not. In that case, it will be necessary to parse the code to obtain a list of the components used.
- **Incorrect assumptions about layout:** for example, suppose there are iOS and Android layout XML files that both specify two buttons. Based on the order in which components are specified in the file, it will be assumed that the first button from one file corresponds to the first button from the other. This assumption may not be valid.
- **Conflicting guidelines:** both iOS and Android provide a set of guidelines for user interfaces. These guidelines are not always compatible. Furthermore, adherence to such guidelines is often a requirement for the app to be distributed through the app store.
- **Platform-specific features**: each mobile platform has a number of components unique to that platform. For example, the "back" button, to return users to the previous screen, is physical for Android devices but it is a GUI component on iOS. The MobiTest tool will need to be able to identify these features and allow users to access and test them.
- **Inconsistent number of screens:** a single screen on one platform may correspond to multiple screens on another. MobiTest will therefore need to allow mappings between components on an application level, rather than at screen level.

VI. PLAN FOR FUTURE WORK

1. Determine components, events and attributes for a number of platforms by creating a compatibility matrix which identifies which components are available on which platform (RQ1) and how they correspond to components on other platforms (RQ2). For example, a Picker in iOS has no equivalent in Android but the ListView provides similar functionality.

2. Define the virtual platform Mobi and the functions Φi (RQ3). To help with this, an on-going study of existing multi-platform applications will be used to provide insight into how existing applications are created for cross-platform use and to help identify common conventions that are used in this context.

3. Write MobiTest for one platform, Android. To do this, a comprehensive examination of existing unit testing tools will be performed. This examination will focus mainly on the Android testing API [9], a specialisation of JUnit [10], and Logic Unit Tests for testing iOS applications [11]. Once this has been done, MobiTest will be generalised to multiple platforms.

VII. SUMMARY

This paper has proposed MobiTest, a testing tool for cross-platform mobile application development, which uses a domain-specific language for mobile interfaces. Short development cycles and the wide range of platforms mean that time available for testing is limited when developing applications for mobile devices. MobiTest will address this issue by allowing developers to specify a single set of tests for applications that can then be used with each platform on which the application is developed.

Conflicting guidelines and platform specific features are just some of the challenges when developing such a platform. If these challenges can be addressed, testing of mobile applications can be simplified and performed more easily leading to higher quality mobile applications and a much more enjoyable, satisfying and effective user experience.

Although this approach may not solve all of the issues associated with automated testing of mobile applications, we believe that it will help to address issues specifically relating to application development across multiple platforms.

VIII. ACKNOWLEDGEMENTS

This research is supported by Oxford Brookes University and the Science Foundation Ireland (SFI) Stokes Lectureship Programme, grant number 07/SK/I1299, the SFI Principal Investigator Programme, grant number 08/IN.1/I2030 (the funding of this project was awarded by Science Foundation Ireland under a co-funding initiative by the Irish Government and European Regional Development Fund), and supported in part by Lero - the Irish Software Engineering Research Centre (http://www.lero.ie) grant 10/CE/I1855.

IX. REFERENCES

[1] ITU, "The world in 2010 ICT Facts and Figures," ITU, 2010.

[2] D. Flood, R. Harrison, D. Duce, and C. Iacob "Using Mobile Apps: Investigating the usability of mobile apps from the users perspective," International Journal of Mobile HCI (In Press) 2012.

[3] G. Tassey, "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology 2002.

[4] F. P. Brooks, *the mythical man-month*: Addison-Wesley Publishing Company, 1982.

[5] W. Royce, "Managing the Development of Large Software Systems," in *WESCO*, 1970.

[6] K. Beck, *Extreme Programming Explained: Embrace Change*: Addison-Wesley, Pearson Education, 2000.

[7] B. George and L. Williams, "An initial Investigation of Test Driven Development in Industry," in *ACM Symposium on Applied Computing*, Melbourne, FL, 2003.

[8] D. Zhang and B. Adipat, "Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications," *International Journal of Human-Computer Interaction*, vol. 18, pp. 293 - 308, 2005.

[9] A. D. Guide, "http://developer.android.com/." vol. 2011, 2011. (accessed 25/09/2012)

[10] JUnit, "http://www.junit.org/" 2011. (accessed 25/09/2012)

[11] i. D. Library, "http://developer.apple.com/." vol. 2011, (accessed 25/09/2012)

[12] http://www.usixml.org (accessed 25/09/12)

 ISBN: 978-1-61208-230-1