Test Patterns for Android Mobile Applications

INÊS COIMBRA MORGADO, DEI, Faculty of Engineering of University of Porto, INESC TEC
and ANA C. R. PAIVA, DEI, Faculty of Engineering of University of Porto, INESC TEC

Mobile applications are a rapidly increasing part of our daily life, featuring more than one million applications and fifty billions downloads in the two major markets. Thus, it is important to ensure their functional correctness. The Pattern-Based GUI Testing (PBGT) project presented an approach for systematising and automating the GUI testing of web applications by modelling testing goals with User Interface Test Patterns(UITPs), *i.e.*, test strategies for recurring behaviour of the UI. This paper extends the set of UITPs used by the PBGT project with three UITPs specific to the testing of mobile applications: Side Drawer, Orientation and Resources Dependency.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability/ Verification*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing Tools*; D.2.13 [**Software Engineering**]: Reusable Software—*Reuse models*

General Terms: Pattern-Based GUI Testing

Additional Key Words and Phrases: GUI Modelling, UI Patterns, Test Patterns, Mobile applications

1. INTRODUCTION

Since the release of the iPhone in 2007 [Apple 2007] and of the first Android smart phone in 2008 [Android 2008; Wilson 2008], smart phones have started to greatly increase their mobile sales. In fact, in 2013 both Android's Google Play and Apple's App Store surpassed the one million available applications and fifty billion downloads threshold [Ingraham 2013]. This market dimension makes it extremely important to ensure the quality of an application as this generates a high level of competitiveness and, thus, for one to get popular it must be as flawless as possible. Furthermore, there has also been an increase of business critical mobile applications, such as mobile banking applications, which makes it even more important to ensure its functional correctness. However, due to particularities of the mobile world, such as new interaction gestures, small memory and new development concepts like activities, the mobile application testing process is a challenging activity [Amalfitano et al. 2012]. According to the World Quality Report 2014-15 [Capgemini et al. 2014], the number of organisations performing mobile testing is growing from 31% in 2012 to 55% in

This work is financed by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness), POPH-QREN Programme (operational programme for human potential) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within the project FCOMP-01-0124-FEDER-020554 and the PhD scholarship SFRH/BD/81075/2011.

Author's address: Inês Coimbra Morgado, Universidade do Porto Faculdade de Engenharia, Rua Dr. Roberto Frias, 4200-465, Porto, Portugal, email: pro11016@fe.up.pt; Ana C. R. Paiva, Universidade do Porto Faculdade de Engenharia, Rua Dr. Roberto Frias, 4200-465, Porto, Portugal, email: apaiva@fe.up.pt

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YY ACM 0000-0000/YY/01-ART1 $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000

2013 and near 87% in 2014. The same report mentions that the greatest challenge for mobile testing is the lack of the right testing processes and methods, followed by insufficient time to test and absence of in-house mobile test environments.

When automating tests, one can automate the execution of the test cases (Robotium[1], Selendroid[2]) or their generation. The first has several problems that need to be tackled, such as the variety of devices, which makes the execution of the test scripts a challenge, and the variety of platforms, which hardens the maintenance of the test scripts. Regarding the automation of test cases generation, Model Based Testing (MBT) [Utting and Legeard 2006] is the most popular technique [Kervinen et al. 2006; Memon 2001; Xie 2006; Vieira et al. 2006; Arlt et al. 2011].

The main issue of MBT is the necessity for an input model of the application, whose manual construction is a time consuming and error prone process. A recent project, Pattern-Based GUI Testing (PBGT) [Moreira and Paiva 2014a], diminishes the effort required in building this model by presenting an easy to use modelling framework [Monteiro and Paiva 2013] and by providing a Domain Specific Language (DSL), PARADIGM [Moreira and Paiva 2014b], that increases the abstraction level of the models describing test goals instead of the system functionality. This DSL is built on top of User Interface Test Patterns (UITPs) that provide test strategies for testing common behaviour, the so called UI Patterns. For instance, the Login UITP encompasses the test strategies necessary to verify if a login UI Pattern [Toxboe 2012] is correctly implemented.

Furthermore, in the context of the PBGT project an experiment with mobile applications was conducted in order to assess if the same approach could also be applied [Costa et al. 2014]. The success of the experiment proved the necessity of developing test strategies (UITP) specific for the mobile world. However, when defining test strategies for mobile it is important to be aware of the possible differences between the existing operating systems as they have followed distinct design conventions [Neil 2014].

The remaining of this paper is structured as follows. Section 2 presents the PBGT project. Section 3 presents three test patterns specific for mobile applications. Section 4 presents the drawn conclusions.

2. OVERVIEW ON PATTERN-BASED GUI TESTING

The PBGT project [Moreira and Paiva 2014a] is based on the assumption that GUIs similar in design, *i.e.*, based on the same UI patterns, should share the same test strategy [Moreira et al. 2013]. As such, this project developed the notion of UI Test Pattern, which is the association of a set of test strategies to a UI Pattern. For instance, the Login UI Test Pattern defines a test strategy to test the authentication process, which is very common in software applications. However, the implementation of the authentication process can differ in the different software applications, *e.g.*, when the authentication fails a pop-up message may optionally appear. So the Login UI Test Pattern may be configured to describe the slightly different implementations and check if the test passed or failed.

In the context of the PBGT project a tool [Moreira and Paiva 2014a] was developed on top of the Eclipse Modelling Framework[3]. It is divided in the following components:

a) a DSL called PARADIGM to build test models based on UITPs [Moreira and Paiva 2014b];
b) a Modelling environment to build and configure GUI models[Monteiro and Paiva 2013];
c) a reverse engineering process to automatically generate the model of a web application [Sacramento and Paiva 2014];
d) a test case generator based on PARADIGM models [Vilela and Paiva 2014].

[1]http://robotium.googlecode.com/
[2]http://selendroid.io/
[3]http://eclipse.org/modeling/emf/

Even though the initial goal of the PBGT project was to test web applications, an experiment was performed to assess if the same approach could be used to test mobile applications, which produced encouraging results [Costa et al. 2014]. However, it was also concluded that there is the need to develop test strategies specific for the mobile world as it presents characteristics that web applications do not, *e.g.*, context events. The goal of this research work is to extend the PBGT approach with some UITPs specific for testing mobile applications.

3. THE TEST PATTERNS

As stated in Section 2, a UI Test Pattern is a set of test strategies to test a recurring behaviour. In order to make these patterns reusable and to facilitate the definition of more patterns, this paper presents the formal definition of the test pattern used in the context of the PBGT project.

3.1 Test Pattern Formal Definition

A test pattern is the set of the associated test strategies and consists on: <Goal, V, A, C, P> as defined in [Moreira et al. 2013] in which:

Goal. is the ID of the test;

V. is a set of pairs variable, value relating test input data with the variables involved in the test;

A. is the sequence of actions to perform during test case execution;

C. is the set of checks to perform during the test case execution;

P. is the precondition (boolean expression) defining the conditions in which it is possible to run the test.

As such, the *Goal* IDentifies the test with a name, V and A represent What to do and How to do it, C represents Why the test is run, *i.e.*, the final purpose of the test, and P represents When the test strategy is to be executed.

These patterns are formally defined as:

$$G[configuration] : P- > A[V]- > C \tag{1}$$

,*i.e.*, for each goal configuration (G), if the pre-conditions (P) are verified, then a sequence of actions (A) is executed with the corresponding input values (V). In the end, a set of checks (C) is performed.

3.2 Test Patterns

The description of the test patterns in this Section is based on [Meszaros and Doble 1997]:

Pattern Name. unique identifier to shortly refer the pattern;

Context. situation where the problem occurs;

Problem. description of the problem addressed by the pattern;

Forces. reflections to consider when choosing a solution to the problem;

Solution. description of the proposed solution for the pattern;

Consequences. positive and negative consequences that arise from the solution;

Application Candidates. real conditions where the (UI Test) patterns can be applied.

3.2.1 *Side Drawer Test Pattern.*

Context
The Android OS provides several forms of navigation through its different screens and hierarchy. One of these is the *Side Drawer* (or Navigation Drawer) UI Pattern, *i.e.*, a transient menu that opens when the

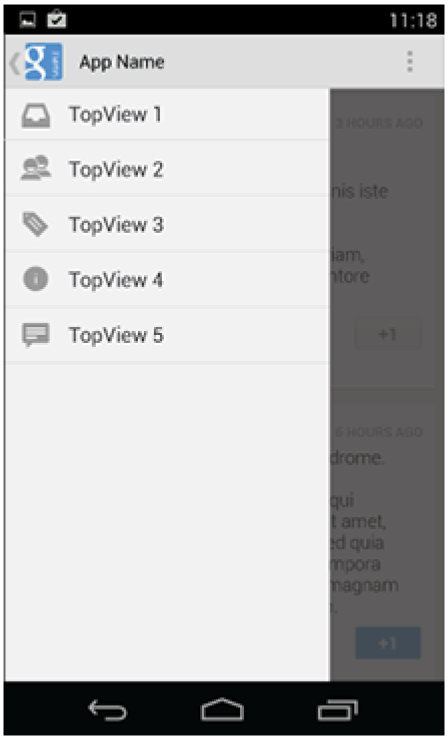

Fig. 1. Example of a *Side Drawer* UI Pattern [Android 2015b]

user swipes the screen from the left edge to the centre or clicks on the application icon on the left of the application's *Action Bar*. Figure 1 depicts an example of this UI pattern.

According to Android's guidelines [Android 2015b], when this menu is open it should not cover the *Action Bar* and it should be possible to close it by clicking the applications icon button.

Problem
How to detect situations in which the Android Guidelines for the Side Drawer menu are not followed?

Forces

—It may not be easy to identify when a side drawer is available to be open

—No pre-configuration is required

Solution

1) Test Goal: "Side Drawer position on screen"
2) Set of Variables V: {}
3) Sequence of Actions A: [open side drawer]
4) Set of Checks C: "side drawer does not cover action bar and it is possible to interact with the app icon"
5) Set of Preconditions P: "side drawer available"

Consequences

—Assurance that the application follows the guidelines for the Side Drawer design pattern

Application Candidates

—Side Drawer [Neil 2014; Android 2015b] (*e.g.* Vueling[4], Booking[5])

It is interesting to note that several Google applications for Android do not follow their own guideline and when the side drawer is open it covers the Action Bar. Some examples are Gmail[6], Hangouts[7], Google Keep[8], Youtube[9] and Google Docs[10] applications in the most recent Android version, 5.1.

3.2.2 *Orientation Test Pattern.*

Context
Android devices have two possible orientations: portrait and landscape. When rotating the device, the screen of the application also rotates and its layout is updated. However, according to Androids Guidelines for testing [Android 2015a] there are two main aspects the developers should test: custom UI code can handle the changes and no user input data should be lost.

Problem
What is the best strategy to test if the application can correctly handle the rotation of the screen?

Forces

—The option of changing the orientation upon the devices rotation must be enabled on the device
—It may not be trivial to compare different status of the screen

Solution

1) Test Goal: "Data unchanged when screen rotates"
2) Set of Variables V: {}
3) Sequence of Actions A: [rotate screen]
4) Set of Checks C: "user entered data was not lost"
5) Set of Preconditions P: "orientation change possible and data inserted"

and

1) Test Goal: "UI elements available when screen rotates"
2) Set of Variables V: {}
3) Sequence of Actions A: [rotate screen]
4) Set of Checks C: "elements available before rotation are available after rotation"
5) Set of Preconditions P: "orientation change possible and a screen change detected"

Consequences

[4]https://play.google.com/store/apps/details?id=com.mo2o.vueling&hl=en
[5]https://play.google.com/store/apps/details?id=com.booking&hl=en
[6]https://play.google.com/store/apps/details?id=com.google.android.gm
[7]https://play.google.com/store/apps/details?id=com.google.android.talk
[8]https://play.google.com/store/apps/details?id=com.google.android.keep
[9]https://play.google.com/store/apps/details?id=com.google.android.youtube
[10]https://play.google.com/store/apps/details?id=com.google.android.apps.docs.editors.docs

—Assurance that all the screens handle the change in orientation correctly

Application Candidates

—Writing an email (*e.g.*, Gmail[11])
—Filling a search field (*e.g.*, Youtube[12])

3.2.3 *Resources Dependency Test Pattern.*

Context
Several applications use external resources, such as GPS or Wifi. Moreover, several of these are dependent on the availability of those resources. As such, it is important to verify if the application does not crash when the resource is suddenly made unavailable [Android 2015a].

Problem
What is the best strategy to test whether the application still behaves correctly when a resource it is using is made unavailable?

Forces

—It is necessary to be able make the device's resources unavailable
—It is necessary to detect when an application is using a certain resource

Solution

1) Test Goal: "No crash when stopping service"
2) Set of Variables V: {resource_name}
3) Sequence of Actions A: [stop resource]
4) Set of Checks C: "application did not crash"
5) Set of Preconditions P: "service running and service being used by the app"

Consequences

—Assurance that the application handles resource unavailability

Application Candidates

—Mobile application using Internet connection (*e.g.*, Facebook[13])
—Mobile application using GPS signal (*e.g.*, Google Maps[14])

4. CONCLUSIONS

This paper presents three UI Test Patterns to test mobile applications that extend previous work developed in the context of the Pattern-Based GUI Testing project: the Side Drawer Test Pattern, the Orientation Test Pattern, and the Resources Dependency Test Pattern.

The success of the PBGT project has proven the usefulness of defining test strategies (UI Test Patterns) for testing recurring behaviour (UI Patterns) on web applications. Moreover, the experiment conducted on

[11]https://play.google.com/store/apps/details?id=com.google.android.gm&hl=en
[12]https://play.google.com/store/apps/details?id=com.google.android.youtube&hl=en
[13]https://play.google.com/store/apps/details?id=com.facebook.katana&hl=en
[14]https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=en

Android has proven that even though the same approach can be applied to mobile applications, it is necessary to specify UI test patterns that are mobile specific. This happens because mobile applications have additional behaviour that is not present in web applications, such as changing the orientation of the screen.

In the future, we intend to develop more UI Test Patterns specific to the mobile world, integrate these three UITPs within the PARADIGM language (the DSL used in the context of the PBGT project) in order to enable the modelling of additional test goals specific for the mobile world and implement the test strategies for these UITPs in order to allow its automatic execution.

REFERENCES

Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. 2012. Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. ACM Press, New York, New York, USA, 258–261.

Google Android. 2008. Announcing the Android 1.0 SDK, release 1. (Sept. 2008). http://android-developers.blogspot.pt/2008/09/announcing-android-10-sdk-release-1.html

Google Android. 2015a. Android - What To Test. (2015). http://goo.gl/AL22tJ

Google Android. 2015b. Navigation Drawer. (2015). https://developer.android.com/design/patterns/navigation-drawer.html

Apple. 2007. Apple Reinvents the Phone with iPhone. (2007). https://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

Stephan Arlt, Cristiano Bertolini, and Martin Schäf. 2011. Behind the Scenes: An Approach to Incorporate Context in GUI Test Case Generation. In *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2011)*. Washington, DC, USA, 222–231.

Capgemini, HP, and Sogeti. 2014. World Quality Report 2014-15. (2014). http://www.capgemini.com/thought-leadership/world-quality-report-2014-15

Pedro Costa, Ana C.R. Paiva, and Miguel Nabuco. 2014. Pattern Based GUI Testing for Mobile Applications. In *9th International Conference on the Quality of Information and Communications Technology (QUATIC 2014)*. IEEE, Guimarães, Portugal, 66–74.

Nathan Ingraham. 2013. Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio. (Oct. 2013). http://goo.gl/z3RprB

Antti Kervinen, Mika Maunumaa, Tuula Pääkkönen, Mika Katara, Wolfgang Grieskamp, and Carsten Weise. 2006. Model-Based Testing Through a GUI. In *5th International Workshop on Formal Approaches to Testing of Software (FATES 2005) (Lecture Notes in Computer Science)*, Wolfgang Grieskamp and Carsten Weise (Eds.), Vol. 3997. Springer Berlin Heidelberg, Berlin, Heidelberg, 16–31.

Atif Muhammed Memon. 2001. *A comprehensive framework for testing graphical user interfaces*. Ph.D. Dissertation.

Gerard Meszaros and Jim Doble. 1997. A pattern language for pattern writing. In *Pattern languages of program design 3*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 529–574.

Tiago Monteiro and Ana C. R. Paiva. 2013. Pattern Based GUI Testing Modeling Environment. In *Sixth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013)*. IEEE, Luxembourg, Luxembourg, 140–143.

RodrigoM. L. M. Moreira and Ana C. R. Paiva. 2014a. PBGT tool: an integrated modeling and testing environment for pattern-based GUI testing. In *29th ACM/IEEE international conference on Automated software engineering (ASE 2014)*. ACM Press, New York, New York, USA, 863–866.

Rodrigo M. L. M. Moreira and Ana C. R. Paiva. 2014b. A GUI Modeling DSL for Pattern-Based GUI Testing PARADIGM. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'2014)*. Lisbon, Portugal.

Rodrigo M. L. M. Moreira, Ana C. R. Paiva, and Atif Memon. 2013. A pattern-based approach for GUI modeling and testing. In *24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)*. IEEE, Passadena, CA, 288–297.

Theresa Neil. 2014. *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps* (2nd ed.). O'Reilly Media, Inc., Sebastopol, Canada.

Clara Sacramento and Ana C. R. Paiva. 2014. Web Application Model Generation through Reverse Engineering and UI Pattern Inferring. In *9th International Conference on the Quality of Information and Communications Technology (QUATIC 2014)*. IEEE, Guimarães, Portugal, 105–115.

Anders Toxboe. 2012. UI-Patterns. (2012). http://ui-patterns.com/

Mark Utting and Bruno Legeard. 2006. *Practical Model-Based Testing: A Tools Approach* (1 ed.). Morgan Kaufmann Publishers, San Francisco, CA, USA.

Marlon Vieira, Johanne Leduc, Bill Hasling, Rajesh Subramanyan, and Juergen Kazmeier. 2006. Automation of GUI testing using a model-driven approach. In *2006 international workshop on Automation of software test (AST 2006)*. ACM Press, New York, New York, USA, 9–14.

Liliana Vilela and Ana C. R. Paiva. 2014. PARADIGM-COV - A Multidimensional Test Coverage Analysis Tool. In *9th Iberian Conference on Information Systems and Technologies (CISTI2014)*. IEEE, Barcelona, Spain, 1–7.

Mark Wilson. 2008. T-Mobile G1: Full Details of the HTC Dream Android Phone. (Sept. 2008). http://goo.gl/6vqI4E

Qing Xie. 2006. Developing cost-effective model-based techniques for GUI testing. In *28th international conference on Software engineering - (ICSE 2006)*. ACM Press, New York, New York, USA, 997–1000.