# Test Patterns for Android Mobile Applications

INÊS COIMBRA MORGADO, pro11016@fe.up.pt, DEI, Faculty of Engineering of University of Porto, INESC TEC
ANA C. R. PAIVA, apaiva@fe.up.pt, 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

**ACM Reference Format:**

Inês Coimbra Morgado, Ana C. R. Paiva, C. 2015.Test Patterns for Android Mobile Applications. jn V, N, Article 1 (January YY), 7 pages.

## 1. INTRODUCTION

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 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. Moreover, the process of updating a mobile application is more laborious than the one updating a web application for instance and, thus, early testing is important.

When automating tests, one can automate the execution of the test cases (Espresso[1], UiAutomator[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

[1] http://developer.android.com/intl/es/training/testing/ui-testing/espresso-testing.html
[2] http://developer.android.com/intl/es/training/testing/ui-testing/uiautomator-testing.html


EuroPLoP '15, July 08 - 12, 2015, Kaufbeuren, Germany

ACM 978-1-4503-3847-9/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2855321.2855354

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 under test, whose manual construction is a time consuming and error prone process. A recent project, Pattern-Based GUI Testing (PBGT) [Moreira and Paiva 2014b], 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 2014a], 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 UI 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 2014b] is based on the assumption that GUIs similar in design, *i.e.*, based on the same UI patterns, should share the same UI 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 2014b] 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 2014a];
b) a Modelling environment to build and configure GUI models[Monteiro and Paiva 2013];
c) a reverse engineering process to automatically extract and generate the model of a web application [Sacramento and Paiva 2014];
d) a test case generator based on PARADIGM models [Vilela and Paiva 2014].

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, such as different development concepts like activities, new interaction gestures and limited memory. The goal of this research work is to extend the PBGT approach with some UITPs specific for testing mobile applications.

## 3. THE UI 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 UI Test Pattern used in the context of the PBGT project.

[3] http://eclipse.org/modeling/emf/

### 3.1 UI Test Pattern Formal Definition

A UI 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.* the ID of the test;

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

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

*C.* the set of checks to perform during the test case execution, *i.e.* the indication of whether the pattern is correctly implemented or not;

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

As such, *Goal* identifies the pattern, *P* defines when the UI Test Pattern should be applied, *A* defines the sequence of actions to perform in order to execute the test and *C* indicates if the test passed or failed. *V* contains pre-configurable information necessary for the pattern (*e.g.* the correct user/password pair in a login/password pattern).

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 UI Test Patterns

The description of the UI Test Patterns in this Section is based on the template proposed by [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.

The following subs-sections present three patterns identified in Android mobile applications.

#### 3.2.1 *Side Drawer UI 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 user swipes the screen from the left edge to the centre of the screen or clicks on the application icon on the left of the application's *Action Bar*. Figure 1 depicts an example of this UI pattern, presenting an example of an application before and after opening the side drawer.

According to Android's guidelines [Android 2015c], when this menu is open it should occupy the full height of the screen.

*Problem*

A good Android developer should follow the Android guidelines in order to provide the best application possible to the end user. One of those guidelines refers to the position and size of the side drawer menu.

*Forces*

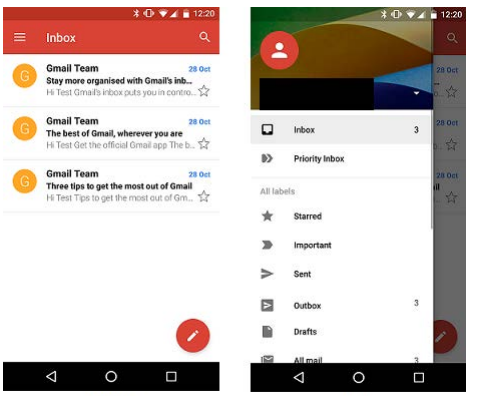


Fig. 1. Example of a *Side Drawer* UI Pattern (before and after being opened)

—It may not be easy to identify when a side drawer is available to be open;
—It is not trivial to identify the side drawer element;

*Solution*
The UI Test Pattern for this type of behaviour is only applied when the side drawer exists and consists in opening the side drawer and checking if it takes up all the screen height

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 takes up all the screen's height"
5) Set of Preconditions P: "side drawer available"

*Consequences*

—Assurance that the application follows the guidelines for the Side Drawer design pattern
—In case of a false negative, it may provide the user a false sense of correctness to the user

*Application Candidates*

—Side Drawer [Neil 2014; Android 2015b] (*e.g.* Vueling[4], Booking[5], Google Calendar[6], Google Keep[7])

### 3.2.2 *Orientation UI 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 AndroidâĂŹs 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*
When the orientation of an Android mobile device changes, the application should adapt to this change without

[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.calendar&hl=en
[7]https://play.google.com/store/apps/details?id=com.google.android.keep&hl=en

loosing any information previously introduced by the user.
*Forces*

—The option of changing the orientation upon the deviceâĂŹs rotation must be enabled on the device

—It may not be trivial to compare different states of the screen because the items' position may change and they may not be visible without scrolling the screen

*Solution*
In order to check the behaviour of the screen rotation, it is necessary to check if the screen elements and the previously input inserted data were not lost after rotating the screen.

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*

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

—In case of a false negative, it may provide the user a false sense of correctness to the user

*Application Candidates*

—Writing an email (*e.g.*, Gmail[8])

—Filling a search field (*e.g.*, Youtube[9])

### 3.2.3 *Resources Dependency UI 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*
The application should not crash when a resource that is being used 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

—It is necessary to be able to identify a crash

[8] https://play.google.com/store/apps/details?id=com.google.android.gm&hl=en
[9] https://play.google.com/store/apps/details?id=com.google.android.youtube&hl=en

*Solution*
In order to apply this pattern, the resource being used (available in V) is stopped and it is verified if the application did not crash.

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
—In case of a false negative, it may provide the user a false sense of correctness to the user

*Application Candidates*

—Mobile application using Internet connection (*e.g.*, Facebook[10])
—Mobile application using GPS signal (*e.g.*, Google Maps[11])

## 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 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.

## Acknowledgements

A special thank you to our shepherd, Rohini Sulatycki, for all her guidance in improving this paper.
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

## REFERENCES

AMALFITANO, D., FASOLINO, A. R., TRAMONTANA, P., DE CARMINE, S., AND MEMON, A. M. 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.

ANDROID, G. 2015a. Android - What To Test.

[10] https://play.google.com/store/apps/details?id=com.facebook.katana&hl=en
[11] https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=en

ANDROID, G. 2015b. Navigation Drawer.

ANDROID, G. 2015c. Up and running with material design.

ARLT, S., BERTOLINI, C., AND SCHÄF, M. 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.

COSTA, P., PAIVA, A. C. R., AND NABUCO, M. 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.

KERVINEN, A., MAUNUMAA, M., PÄÄKKÖNEN, T., KATARA, M., GRIESKAMP, W., AND WEISE, C. 2006. Model-Based Testing Through a GUI. In *5th International Workshop on Formal Approaches to Testing of Software (FATES 2005)*, W. Grieskamp and C. Weise, Eds. Lecture Notes in Computer Science Series, vol. 3997. Springer Berlin Heidelberg, Berlin, Heidelberg, 16–31.

MEMON, A. M. 2001. A comprehensive framework for testing graphical user interfaces. Ph.D. thesis.

MESZAROS, G. AND DOBLE, J. 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.

MONTEIRO, T. AND PAIVA, A. C. R. 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.

MOREIRA, R. M. L. M. AND PAIVA, A. C. R. 2014a. 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.

MOREIRA, R. M. L. M. AND PAIVA, A. C. R. 2014b. 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.

MOREIRA, R. M. L. M., PAIVA, A. C. R., AND MEMON, A. 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.

NEIL, T. 2014. *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps* 2nd Ed. O'Reilly Media, Inc., Sebastopol, Canada.

SACRAMENTO, C. AND PAIVA, A. C. R. 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.

TOXBOE, A. 2012. UI-Patterns.

UTTING, M. AND LEGEARD, B. 2006. *Practical Model-Based Testing: A Tools Approach* 1 Ed. Morgan Kaufmann Publishers, San Francisco, CA, USA.

VIEIRA, M., LEDUC, J., HASLING, B., SUBRAMANYAN, R., AND KAZMEIER, J. 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.

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

XIE, Q. 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.



EuroPLoP '15, July 08 - 12, 2015, Kaufbeuren, Germany



ACM 978-1-4503-3847-9/15/07...$15.00

DOI: http://dx.doi.org/10.1145/2855321.2855354