Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 314 (2015) 3–21

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Evaluating the Model-Based Testing Approach in the Context of Mobile Applications

Guilherme de Cleva Farto[1,2]

Universidade Tecnológica Federal do Paraná - UTFPR
Cornélio Procópio, Brasil

Fundação Educacional do Município de Assis - FEMA
Instituto Municipal de Ensino Superior de Assis - IMESA
Assis, Brasil

TOTVS Agroindústria
Assis, Brasil

Andre Takeshi Endo[3]

Universidade Tecnológica Federal do Paraná - UTFPR
Cornélio Procópio, Brasil

Abstract

The popularity of portable devices has grown rapidly in recent years. Due to the high number and diversity of users, new testing approaches are necessary to reduce the occurrence of faults and ensure better quality in mobile applications. The major objective of this paper is to evaluate the use of Model-Based Testing (MBT) in the construction and implementation of automated tests to verify and validate mobility solutions developed in the Google Android platform. The research proposal is guided by three questions: (Q1) – "Can the concepts of MBT be used in its current state to verify and validate functional requirements in mobile applications?"; (Q2) – "What are the results and challenges identified from adoption of MBT in mobile applications?"; and (Q3) – "How effective were the models and test cases generated, implemented and executed in the mobile application evaluated?". The results obtained from an experimental evaluation are discussed and related to questions of this research.

Keywords: automated testing, mobile applications, model-based testing, Android.

[1] The authors would like to thank TOTVS Agribusiness and its mobility cell by having collaborated with the development of this project. Special thanks to the Universidade Tecnológica Federal do Paraná (UTFPR – in Cornélio Procópio), Fundação Educacional do Município de Assis (FEMA), and Instituto Municipal de Ensino Superior de Assis (IMESA) for their support.

[2] Email: guilherme.farto@gmail.com

[3] Email: andreendo@utfpr.edu.br

http://dx.doi.org/10.1016/j.entcs.2015.05.002

1 Introduction

Currently, there has been a rapid growth in popularity of mobile devices, such as tablets, smartphones, and e-readers. According to a Gartner survey on sales in 2013 [18], the Android [3] platform had an increase of 127% over the previous year with approximately 121 million units sold. Thus, Android took the lead of the current market in the list of systems and applications developed for mobile environments with 62% ahead of Apple iOS [4] and Microsoft Windows Phone [39].

Due to the expansion in the number and diversity of mobile device users, the study of new testing approaches is essential to reduce the occurrence of faults and thereby ensure a better quality of mobile applications. According to Muccini et al. [27], the mobile context has characteristics that directly influence the testing activity, such as connectivity, limited resources, autonomy, user interface, context awareness, adaptation, new operating systems and programming languages, diversity of settings, and touch screen.

A special attention should be directed to the testing phase in mobile applications in order to improve the design and generation of test cases, as well as evaluating methods and tools available for verification and validation. Moreover, it must always be considered points of attention as cost, fault detection effectiveness, and the ability to automate test cases.

In this context, one of the techniques that can be applied to the testing activity to ensure software quality is Model-Based Testing (MBT). According to Utting and Legeard [37], MBT allows the automatic generation of test cases through a model built based on the expected behavior of software under test (SUT). MBT is an approach that has several advantages reported in the literature, such as the automatic test case generation, fault detection effectiveness, and reduction in time and cost for testing [37][7][19].

This paper aims to evaluate the use of the MBT concepts in the design and execution of automated tests in mobile applications. In particular, the research was focused on mobility solutions developed in the Google Android platform.

The following research questions have been defined: *Q1 – "Can the concepts of MBT be used in its current state to verify and validate functional requirements in mobile applications?"*; *Q2 – "What are the results and challenges identified from adoption of MBT in mobile applications?"*; and *Q3 – "How effective were the models and test cases generated, implemented, and executed in the mobile application evaluated?"*.

This paper is organized as follows: Section 2 presents the literature review on MBT and modeling technique called Event Sequence Graph (ESG). Section 3 discusses the testing of mobile applications and contextualizes the main related work. Section 4 describes the study configuration. Section 5 analyzes and discusses the results obtained from the experimental study. Section 6 discusses the threats to validity. Finally, Section 7 presents the conclusion and sketches future work.

2 Model Based-Testing

The software testing area is rich in techniques that can be applied during the development process in order to reveal faults in software [28][5]. Among them, approaches have been defined for automatic generation of test cases using a behavioral or structural model, also called the test model, of the SUT. This approach is known as Model-Based Testing (MBT) [34]. The adoption of models has been motivated by the observation that many tests tends to be unstructured, not repeatable, without documentation, and depend on the tester's creativity [38]. MBT can be divided into four main steps: (i) modeling, (ii) test generation, (iii) concretization, and (iv) test execution [30][14][9].

In modeling, the tester uses her/his understanding of the SUT to create a test model. The requirements are sources of information that make possible a better understanding of the software being tested functionality. In addition, the software product is contained in an environment that presents several factors such as operating systems, other computing solutions, different types of libraries, and other features. Thus, the tester needs to understand both the SUT and the test execution environment. Although it is recommended to create test models based on the requirements to maximize the independence between the model and the SUT [38], artifacts of analysis and design can be also used to understand and construct the test model.

The test generation algorithm depends on the technique adopted to describe the test model. According to El-Far and Whittaker [14], modeling techniques must possess properties that make test generation less expensive and automation easier. In this step, a tool is essential to support the automatic generation of test cases. The test model is submitted as input as well as the test selection criterion and the tool generates a set of test cases. At that moment, the generated test cases are still abstract and not executable because they are in a level of abstraction different from the SUT.

The concretization involves the transformation of the test cases from abstract ones into executable in the SUT. In this paper, we assume that this step will be performed automatically [37], so the abstract test cases are interpreted and executed in the SUT by using adapters [30]. In MBT, an adapter is a software component that transforms the inputs and outputs between two levels of abstraction model and SUT, making it possible to execute an abstract test case.

Finally, test execution is the application of the test cases in the SUT, brought after the transformation performed in the previous step. Following the execution, results are analyzed and corrective actions taken. An automatic check may be performed if the test model specifies both input and output values.

Event Sequence Graph (ESG). To build the test model, some modeling techniques can be used to express clearly the requirements and functionality of the SUT. It is expected that the modeling technique adopted for the MBT is formal, i.e., syntactically and semantically well defined [20]. There are several modeling techniques used in MBT such as Finite State Machines [24], Labeled Transition

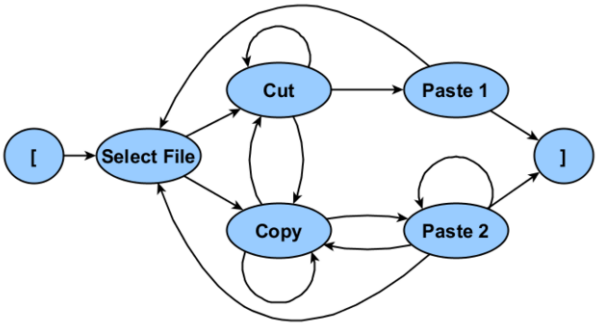


Fig. 1. ESG for a procedure to cut, copy, and paste files (adapted from [15]).

Systems [36], and UML [9]. In this paper, the expected behavior of the SUT is modeled by Event Sequence Graph (ESGs). For this step, we adopted the ESG technique because of its easiness to represent interactions between events and the simplicity to understand the requirements and functionality of the SUT.

An ESG is a directed graph used to model interactions between the software events and consists of nodes that represent events while the edges are valid sequences of these events [6][41]. Figure 1 illustrates an ESG model that defines the events and sequences in a procedure to cut, copy, and paste files. Elements "[" and "]" of the graph represent respectively the start and end of the event sequences.

Two possible full event sequences extracted from ESG in Figure 1 are listed as follows:

```
Cut: [, Select File, Cut, Copy, Cut, Cut, Paste 1, Select File,
Cut, Paste 1, ]
```

```
Copy: [, Select File, Copy, Cut, Copy, Copy, Paste 2, Copy, Paste
2, Paste 2, Copy, Select File, Copy, Paste 2, ]
```

3 Related Work

Despite research in software testing, the rise of technologies and platforms fosters the creation of new research topics. Among them, there is an exponential increase in mobile devices used not only for entertainment but also in critical areas, such as health, financial, and industrial [27]. Muccini et al. [27] mention the need for specialized approaches to create tests for mobile applications, taking into account characteristics that directly influence the testing activity highlighting connectivity, limited resources, autonomy, user interface, context awareness, adaptation, new operating systems and languages programming, diversity of settings, and touch screen devices. As a result, the testing of mobile applications introduces new challenges that must be overcome and have been addressed in scientific and technical research in the areas of software engineering and mobility [27][12][8][23][21][29][40][22][26][2][10][13]. These works can be divided into two lines.

In the first line, traditional testing techniques are adapted to mobile applications. Delamaro et al. [12] describe a strategy to support the structural testing of mobile applications enabling test execution through emulators as well as physical devices. Their approach targets the Java Micro Edition platform (J2ME). Yet in structural testing, Jensen et al. [22] propose a technique that aims to automatically find event sequences to execute a specific line of code. The authors evaluated the technique in Android applications. In functional testing, Bo et al. [8] develop a tool called MobileTest to automate the black box testing from an event-based approach to simplify and improve the design of test cases.

In the second line, researchers investigate faults characteristic of mobile applications and new testing strategies are proposed based on this knowledge. Maji et al. [23] evaluate the reported failures in Symbian and Android platforms resulting in a detailed analysis of found faults, corrections made ,and a comparison between the two operating systems.

Neamtiu and Hu [21] describe an approach to test applications on Google Android and emphasize the user interface faults. The authors adopted random testing techniques, instrumentation, and analysis of virtual machine logs.

Pathak et al. [29] investigate software faults related to excessive energy consumption on smartphones with Android and provide an automatic solution to detect these problems by using a data flow analysis algorithm.

Yang et al. [40] propose a testing technique to identify and quantify causes to faults related to excessive waiting time for certain events in Android applications. To do so, the authors relied on artificial insertion of delay instructions in typical problematic operations.

Liu et al. [26] characterized a series of performance faults commonly identified in Android mobile applications. The authors also present a static analysis tool that is able to detect patterns of faults.

Amalfitano et al. [2] state that the mobile application test-driven user interface has been increasingly explored and automation tools have been proposed. However, they conclude that the use of these supporting tools is not appropriate for developers and testers with little experience. The paper presents an approach based on MBT and Finite State Machines called MobiGUITAR which provides resources for the generation of test cases from the analysis of the graphical interface of an Android application. The MobiGUITAR tool was used in four open source applications and resulted in the generation and execution of 7,711 test cases and ten new faults detected [2].

Figueiredo et al. [10] present a technique for generating test cases based on interactions and behaviors defined in specifications of use cases for mobile applications. In later studies, Nascimento and Machado [13] investigated the use of exploratory testing using Labeled Transition Systems.

Many researchers have investigated testing techniques for mobile applications, emphasizing the Google Android platform. However, few efforts have been expended to address and evaluate methods of test automation for mobile applications. Therefore, this study aims to evaluate MBT and ESG in mobile applications developed

for the Android platform.

4 Study Configuration

The aim of this paper is to evaluate the use of the MBT concepts in the construction and later in the execution of automated tests in mobile applications. In particular, those developed in the Google Android platform. We hope that the proposed study provides some results in order to answer the following research questions:

- (Q1) "Can the concepts of MBT be used in its current state to verify and validate functional requirements in mobile applications?"
- (Q2) "What are the results and challenges identified from adoption of MBT in mobile applications?"
- (Q3) "How effective were the models and test cases generated, implemented and executed in the mobile application evaluated?"

The experimental study in this research involved a group of 15 people, consisting of five professionals with experience in developing mobility solutions with Android and ten undergraduate students of the fourth year of Computer Science. The experiment was divided into steps presented in the following subsections.

4.1 Selecting a mobile application developed in the Google Android platform

In the first stage of the study, we selected a solution presented by Deitel et al. [11]. The project called "AddressBook App" is an application for managing contacts and it uses various resources, such as listing of records with ListActivity, custom visual components with AdapterViews and Adapters, multiple calling of Activities or screens, persistence with SQLite database, styles and themes of interface, and menus with MenuInflater. One of the reasons that contributed to the selection of AddressBook App is its features, which are constantly adopted in Android applications. Other factor is the clear and diverse composition of events such as "display screen", "push button", "trigger menu", and others.

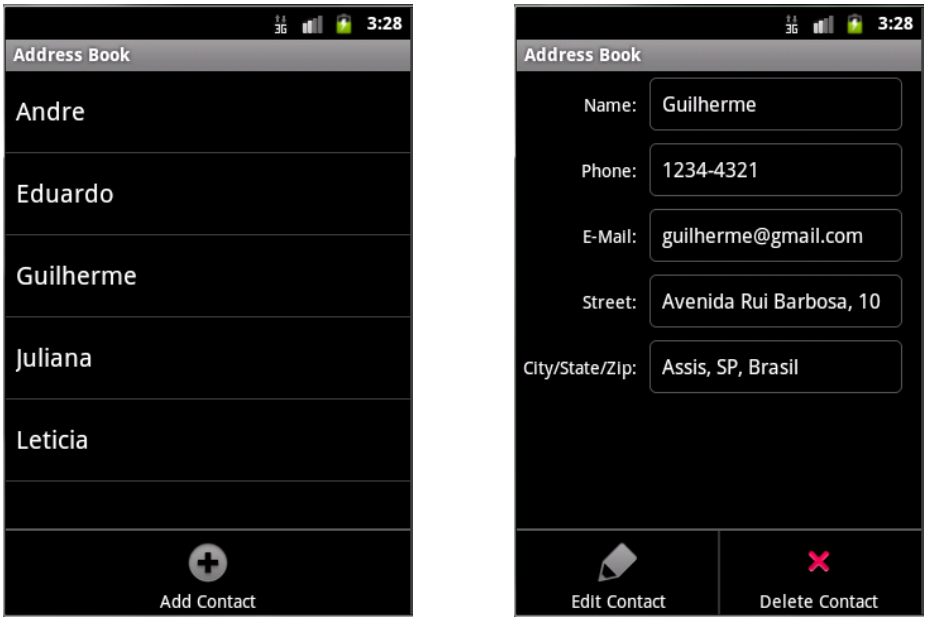
The development of this application was performed collaboratively since participants with more knowledge helped beginners. Furthermore, all the people in the group could understand the requirements and the application's events.

The activity of creating the design, installation, implementation, and manual testing were performed in a period of approximately 50 minutes with access to books and to the application's source code.

Figure 2 illustrates two screens of AddressBook application. Interaction with the application occurs mainly through screens lists of records, record selection, application menus, dialogs, input fields, and action buttons.

4.2 Presentation of MBT concepts and the modeling technique ESG

The next activity held was to train the participants in MBT concepts and the ESG modeling technique. A researcher provided the training to explain the main settings,



(a) List of Contacts (b) View for Contact Update

Fig. 2. AddressBook App screens application: (a) ListActivity with registered contacts and (b) Activity in contact update mode.

the research conducted, advantages, and application of MBT as well as ESGs. The training was held in approximately 30 minutes.

4.3 Creation of ESG test models

Based on the knowledge of MBT and ESG acquired previously, the people' became able to develop a test model in ESG that represents possible events and flows.

At this stage the 15 people involved were divided into three groups with five members each. Thus, we obtained three test models with different perspectives. Groups 1 and 2 were composed by the undergraduate students and Group 3 was composed by developers.

The activity was held in the sequence of the MBT and ESG training. The modeling time ranged from 30 to 40 minutes. Figures 3, 4, and 5 illustrate the three test models.

Figure 3 illustrates the ESG model developed by Group 1. This model encompasses the events of the application, but nodes were not defined for two scenarios: (i) ability to display and to inform the contact data when a record is selected and (ii) an event to display a confirmation dialog when a deletion is requested for a selected contact.

Figure 4 illustrates the ESG model developed by Group 2. This model was the most complete since it includes events to inform and display contact data. However, the group also did not consider an event to confirm the deletion of an existing contact by using a dialog message.

Figure 5 illustrates the model developed by Group 3. This group did not include

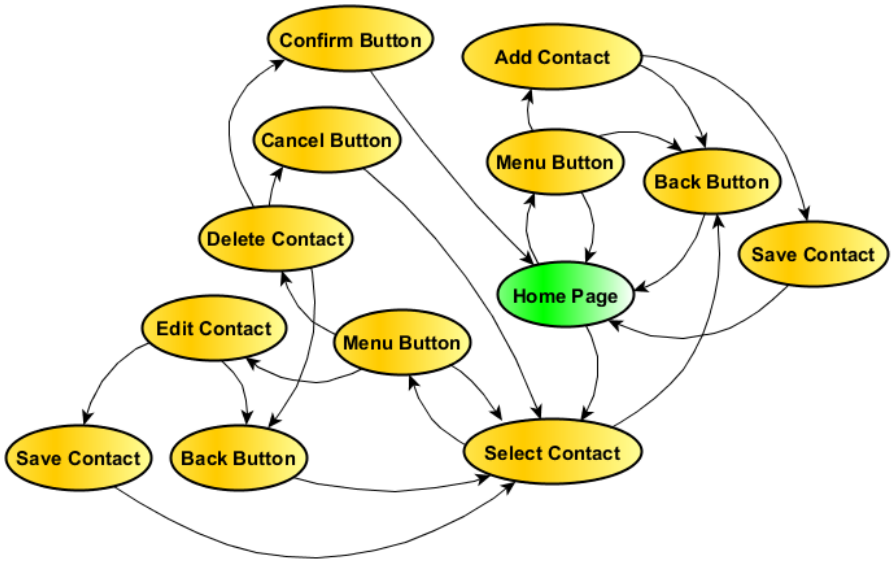


Fig. 3. ESG model for AddressBook application developed by Group 1 (five students from CC).

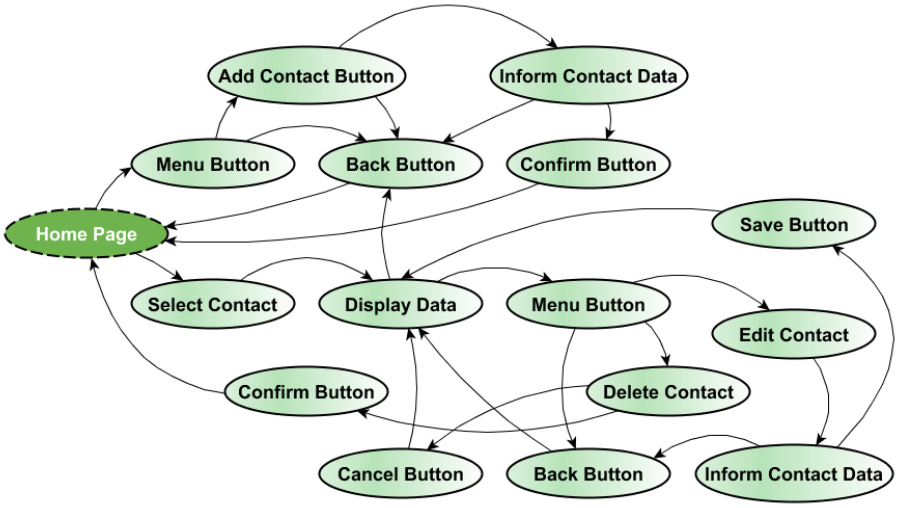


Fig. 4. ESG model for AddressBook application developed by Group 2 (five students from CC).

the problems identified in Groups 1's and 2's models. However, they dismissed that the behavior is different when the back button is (i) pressed and the application is the inclusion mode (return to the main screen) and (ii) pressed for an update (return to the display contact screen.

The three models were analyzed and a derived model was designed. In this way, the event sequences were more comprehensively represented. This was performed since the three ESGs would not validate the SUT correctly. Therefore, the validation of the test model is essential because the quality and effectiveness of the MBT approach are directly related to the model's quality. Figure 6 illustrates a unified

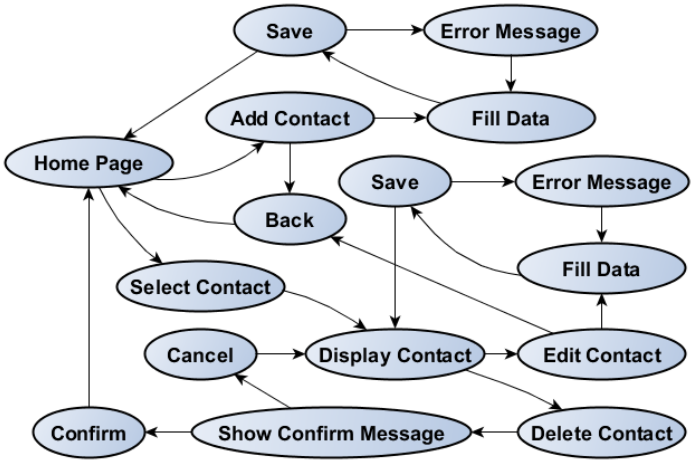


Fig. 5. ESG model for AddressBook application developed by Group 3 (five students from CC).

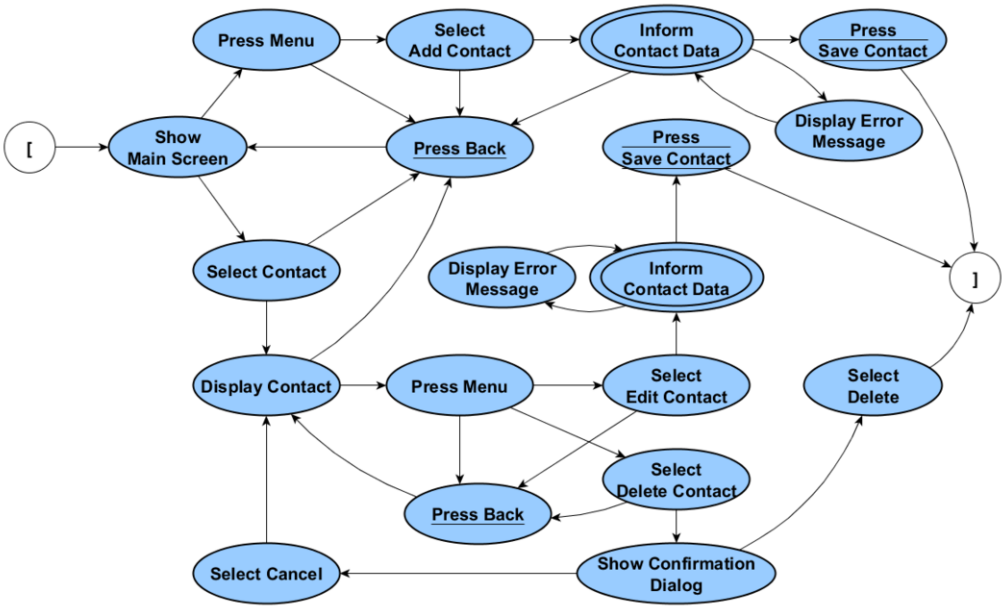


Fig. 6. Unified ESG model based on the requirements, features, and events in AddressBook application.

test model designed by the researchers based on the three ESGs.

The double-circled nodes are related to a decision table; there are two events named “Inform contact data” in Figure 6. Decision tables are resources that can be used in the testing activity [28] and incorporated into the ESG to better represent constraints and effects on input data in a given event. A decision table is defined as a set of rules that determines which flow of application events must be assumed from a test input. [15]. For the nodes “Inform contact data”, the decision table assumes that the contact will not be persisted if any failure exists. In addition, an error notification message will be displayed. Two possible failures are (i) an

incorrect value informed or (ii) a lack of filling a field.

4.4 Generating and implementation of test cases from the ESG model

This phase was conducted by researchers due to limited time to learn the technologies involved. We adopted a supporting tool called TSD4WSC. It provides a graphical interface for ESG modeling and for generating Complete Event Sequences (CESs). A CES is a linear sequence generated from an ESG model and helps the developers and testers to obtain the flow of events expected in the SUT [25]. In MBT, CESs are test cases providing input events and analyzing which are the expected outputs. In the generation of test cases, the test model becomes the input artifact for TSD4WSC tool while the output is a set of CESs. During generation, CESs are generated to cover all the edges in the test model. For this, the Chinese Postman Problem (CPP) [1] algorithm is applied to minimize the testing cost.

In this context, the ESG test model of AddressBook App (Figure 6) was used as input artifact for TSD4WSC. The output of the test cases generation was three CESs with 22, 7, and 20 events to be executed. These three test cases are listed as follows:

```
22 events: [, Show_Main_Screen, Press_Menu, Press_Back,
Show_Main_Screen, Select_addContact, Inform_contactDataR1,
Press_saveContact, Show_Main_Screen, Select_contact, Press_Back,
Show_Main_Screen, Select_contact, Display_contact, Press_Menu,
Press_Back, Display_contact, Press_Menu, Select_editContact,
Inform_contactDataR2, Show_errorMessage, Inform_contactDataR1,
Press_saveContact, ]

7 events: [, Show_Main_Screen, Press_Menu, Select_addContact,
Inform_contactDataR2, Show_errorMessage, Inform_contactDataR1,
Press_saveContact, ]

20 events: [, Show_Main_Screen, Select_contact, Display_contact,
Press_Menu, Select_deleteContact, Press_Back, Display_contact,
Press_Menu, Select_editContact, Press_Back, Display_contact,
Press_Menu, Select_deleteContact, Show_confirmationDialog,
Select_cancel, Display_contact, Press_Menu, Select_deleteContact,
Show_confirmationDialog, Select_delete, ]
```

These test cases are abstract, i.e., they cannot be applied directly in the SUT. Thus, the events need to be implemented. To develop automated tests on the Android platform we can use native features like Instrumentation [17], UIAutomator [35], and MonkeyRunner [17] or adopt the integrated development environment, such as Espresso [16], Robolectric [31], and Selendroid [33]. In this step, the Robotium framework [32] was selected. Currently, Robotium is widely used for

mobile applications testing by providing support to write automated tests. Some advantages are: construction of test cases with minimal knowledge of the SUT, short period of time needed to implement more sophisticated test cases, and faster test execution and control [17][32].

Table 1 brings a brief description of classes and methods (of Robotium tool [17][32]) used in the concretization of abstract test cases.

Table 1
Descriptions of classes and methods employed in test case concretization with Robotium

Instruction (class or method)	**Description**
ActivityInstrumentationTestCase2	Class that provides native functionality for Android testing
com.robotium.solo.Solo	Class that provides methods used in interaction and interface between design and test project to be tested
solo.sendKey(int)	Method that sends or simulating a key event
solo.goBack()	Method that simulates the event of "Back" button
solo.clickInList(int)	Method that simulates the click event on a component list (ListView)
solo.getString(int)	Method which returns a String from the constant that represents
solo.clickOnMenuItem(String)	Method that simulates the selection of a menu item
solo.clickOnButton(String)	Method that simulates the click of a button
solo.clearEditText(int)	Method that clears the contents of a text field
solo.enterText(String)	Method that assigns a String to a text field
solo.finishOpenedActivities()	Method that ends the screens (Activities) displayed

Figure 7 illustrates a fragment of the AddressBookCaseTest class that defines the implemented methods based on generated CESs. The methods `Select_cancel()` and `Select_delete()` have been omitted since they are implemented in the same way as `Press_saveContact()`. The same occurs between the implementations of the methods `Inform_contactDataR1()` and `Inform_contactDataR2()`, changing only the input data. The method `Select_deleteContact()` have been omitted since it is implemented in a similar manner to `Select_addContact()` and `Select_editContact()`. The methods `Show_Main_Screen()`, `Display_contact()`, `Show_confirmationDialog()`, and `Show_errorMessage()` were also omitted because they represent states that were initiated from previous events. For instance, the data of a registered contact is always displayed after his selection in the list of contacts.

Figure 8 illustrates the `TestCaseUtil` class which features a generic method to execute the CESs. The `executeTestCase()` method receives as parameters an instance of the `AddressBookCaseTest` adapter and a String object containing the CES. At runtime, the implemented methods (representing abstract events of the test model) in `AddressBookCaseTest` class are dynamically invoked using the Java Reflection API and the event sequence is performed. The `executeTestCase()` method throws an exception if some fault is identified, failing the test case.

The Android project with the implemented Robotium test cases to verify and validate the mobile application AddressBook App is available at:

```
1  public class AddressBookCaseTest extends ActivityInstrumentationTestCase2 {
2
3      private Solo solo;
4
5      @SuppressWarnings("unchecked")
6      public AddressBookCaseTest() {
7          super(AddressBook.class);
8      }
9
10     @Override
11     public void setUp() throws Exception {
12         solo = new Solo(getInstrumentation(), getActivity());
13     }
14
15     public void testCesOne() throws Exception {
16         String cesOne = "[, Show_Main_Screen, Press_Menu, Press_Back, Show_Main_Screen,
               Select_addContact, Inform_contactDataR1, Press_saveContact, Show_Main_Screen,
               Select_contact, Press_Back, Show_Main_Screen, Select_contact, Display_contact,
               Press_Menu, Press_Back, Display_contact, Press_Menu, Select_editContact,
               Inform_contactDataR2, Show_errorMessage, Inform_contactDataR1,
               Press_saveContact, ]";
17
18         TestCaseUtil.executeTestCase(this, cesOne);
19     }
20
21     public void testCesTwo() throws Exception { // The full CES have been omitted
22         String cesTwo = "[, Show_Main_Screen, Press_Menu, ... , ]"
23
24         TestCaseUtil.executeTestCase(this, cesTwo);
25     }
26
27     public void testCesThree() throws Exception { // The full CES have been omitted
28         String cesThree = "[, Show_Main_Screen, Select_contact, Display_contact, ... , ]"
29
30         TestCaseUtil.executeTestCase(this, cesThree);
31     }
32
33     public void Press_Menu() throws Exception {
34         solo.sendKey(KeyEvent.KEYCODE_MENU);
35     }
36
37     public void Press_Back() throws Exception {
38         solo.goBack();
39     }
40
41     public void Select_contact() throws Exception {
42         solo.clickInList(0);
43     }
44
45     public void Select_addContact() throws Exception {
46         solo.clickOnMenuItem(solo.getString(R.string.menuitem_add_contact));
47     }
48
49     public void Select_editContact() throws Exception {
50         solo.clickOnMenuItem(solo.getString(R.string.menuitem_edit_contact));
51     }
52
53     public void Press_saveContact() throws Exception {
54         solo.clickOnButton(solo.getString(R.string.button_save_contact));
55     }
56
57     public void Inform_contactDataR1() throws Exception {
58         for (int i = 0; i < 5; i++) { solo.clearEditText(i); }
59
60         solo.enterText(0, "Guilherme");
61         solo.enterText(1, "1234");
62         solo.enterText(2, "guilherme@gmail.com");
63         solo.enterText(3, "Av. Faria Lima, 100");
64         solo.enterText(4, "Sao Paulo, SP, Brasil");
65     }
66
67     @Override
68     public void tearDown() throws Exception {
69         solo.finishOpenedActivities();
70     }
71 ....
72 }
```

Fig. 7. Fragment of AddressBookCaseTest code class

```
public class TestCaseUtil {
....
    public static void executeTestCase(AddressBookCaseTest classCaseTest, String cesOne)
         throws Exception {
        String[] events = cesOne.split(",");

        for (int index = 1; index < events.length - 1; index++) {
            String event = events[index].trim();

            Method method = AddressBookCaseTest.class.getDeclaredMethod(event);

            if (method != null) {
                method.invoke(classCaseTest);
            }
        }
    }
....
}
```

Fig. 8. Fragment of TestCaseUtil code class

https://github.com/guilhermefarto/AddressBookTest

4.5 Execution and data collection of implemented test cases with Robotium

We can apply the test cases implemented with Robotium platform directly in the SUT after the concretization phase. Therefore, an Android Virtual Device (AVD) was set up to simulate the environment in which the mobile application is installed and tested. An AVD is an emulator of the Android operating system and the main tool used in mobile application testing to provide a rich set of features also found in physical devices. It also enables combinations of settings as screens with different sizes and qualities, memory size, and storage [3]. An AVD was created with version 2.3.3 (API Level 10) of Android with a screen of 3.2", 512 MB of RAM, and 50 MB memory card.

When the environment is initiated, the test cases are executed in the sequence defined by the CESs while the emulator visually displays the steps. Data of the test cases as execution time and identified faults have been collected and analyzed after the completed execution.

5 Analysis of Results

It is possible to answer the research question Q1 ("Can the concepts of MBT be used in its current state to verify and validate functional requirements in mobile applications?") after the experimental study and the collected results by stating that the MBT testing technique along with the ESG modeling technique can be used as a valid approach for modeling and generating test cases for mobile applications developed on the Google Android platform.

To complement the statement given to answer Q1 we notice that the use of Robotium platform is an interesting choice for concretization and testing since it enables a rapid implementation of test cases generated from of a given ESG model as well as the execution of automated tests that can be visually monitored by the Android emulator. The ability to re-execute tests for different device configurations

by simply configuring the AVD is another important advantage.

The results obtained from the experimental study can be related to the research question Q2 ("What are the results and challenges identified from adoption of MBT in mobile applications?") by emphasizing:

- Automatic generation of test cases: the process of creating test cases through automatic generation with TSD4WSC tool was simplified from ESG unified model. With this, all events were executed ensuring a complete test of the application
- Capacity to detect faults: faults in AddressBook application were located and described in this section proving the ability of the MBT approach for fault detection in mobility solutions after execution of implemented test cases
- Improvement in the test quality: human interference is reduced because the generation and execution of tests are performed automatically. Thus all event sequences in the model are checked and some human factors will not compromise the tests such as forgetting a possible alternative flow, validate only the correct path, or the most common path
- Reduction in time and cost for testing: despite measuring indicators such as time and cost have not been performed in the study, we suggest a possible reduction because the use of same CESs can be applied in several configurations of mobile devices running Android, avoiding manual tests to be performed repeatedly
- Evolution of test models: it is common for test models to be updated to better represent events in a mobile application as well as to meet changing requirements. This evolution occurs in a simple way and the maintenance cost of testing is reduced, once an updated version of the test cases can be regenerated after some change

The experimental study also identified some challenges in the use of MBT in mobile applications that are related to the research question Q2:

- Difficulties in test modeling: the design of the test model can be a complex activity because it requires knowledge about modeling events through diagrams and requires an complete understanding of requirements and flows of the SUT
- Particularities in concretization of test cases in mobility context: we need to take into account some particularities of the context of mobile applications like physical buttons on the device as back button, search button, and return button, transitions between screens, and event-driven screens. There are other features, which are out of the scope of this paper, that could be considered, such as connectivity, data networking, telephony, use of sensors like GPS for geolocation, and integration with Web applications
- Knowledge in Robotium: the tester responsible for the concretization should have knowledge in the Robotium platform and its API

To create the test model that identifies the features and flows of the given mobile application (Figure 6), 23 nodes (events) were used and 33 edges.

Using the TSD4WSC tool, we obtained three CESs, covering all edges in the

ESG model for the AddressBook application. The length (number of events) of each sequence is 22, 7, and 20 events, respectively.

With the specified configuration, the execution times were 163.7s (1st CES with 22 events), 56.8s (2nd CES 7 events), and 79.3s (3rd CES with 20 events). In total, the test suite run in 299.8s (approximately 5 minutes). We noticed that the execution time can be high because the CESs implemented with Robotium is visually shown in the emulator. However, such functionality has its advantages like the possibility to monitor the entire flow of events performed automatically.

Table 2 shows the Lines of Code (LoC) and the McCabe Cyclomatic Complexity for the SUT (AddressBook App) and for the AddressBook App testing application (AddressBookTest). As noted, the average, standard deviation, and maximum cyclomatic complexity values remained close between the AddressBook App e AddressBookTest projects. Such proximity of values is justified by the implementation of the generic method `executeTestCase()` from the `TestCaseUtil` class. This method is responsible for dynamic execution of generated CESs. However, it is emphasized that the generic method can be used in more complex tests, i.e., sequences with an extensive number of events and more comprehensive test cases. However, it is only necessary the generation and concretization of CESs for the Android applications that will be tested.

Table 2
Data Consolidation

Mobile Application	**Concretization Data Analysis**			
	LoC	**McCabe Cyclomatic Complexity**		
		Average	**SD**	**Max.**
AddressBook App	416	1,22	0,497	3
AddressBookTest	124	1,23	0,516	3

Faults related to data validation were found. This fact was an initial evidence for the research question Q3 ("How effective were the models and test cases generated, implemented and executed in the mobile application evaluated?"). The four faults are described as follows:

- (1) It allows inserting and updating contacts only informing the contact's name and disregarding important data like phone number, e-mail, and address
- (2) It allows inserting and updating contacts by informing numerical values (e.g., "1234") for the contact name. Fault #1 is still active in this situation
- (3) It enables to include contacts with redundant data. For instance, there may exist contacts with for the same name or e-mail
- (4) It lacks an error message indicating the some invalid data (for Faults #1, #2, and #3)

We emphasize that the evaluated application is simple and has learning purposes. Thus, such faults may not be considered in the original project shown in [11]. Yet, the test model and generated test cases were able to reveal some faults, though

more experimental studies are needed.

6 Threats to Validity

The study presented in this paper was conducted to evaluate the applicability of MBT in the context of mobility. The results provided relevant evidence to the outlined issues and can be used to motivate future work. However, the study presents threats discussed as follows that make it impossible to generalize the results to other scenarios.

One issue was the work of researchers in the stages of generation and concretization of the tests through the use of TSD4WSC and Robotium tools. We opted for this configuration as the emphasis of the study was to evaluate the use of an MBT approach and not the tools. Other point is that the study was set up with only a mobile application AddressBook [11]. Although it is developed by third parties and is characterized as a typical application, the domain, and the features of Android applications are much broader.

The choice of MBT techniques and tools is also a possible threat. There is no consensus in the literature on a single technique or tool that characterizes the MBT approach. So a question is whether the choices of technique and tools are representative or not. However, the technique and the tools have been applied in other contexts [6][15][25] and is not complicated to realize that the testing process adopted follows the steps described in the MBT literature [37][30].

Finally, it is important to note that those involved in the experiment were responsible for the implementation of the SUT and subsequently ESG modeling. Consequently, the definition of the test model may have been influenced by knowledge of the functionality and flows of the mobile application.

In this section, we discuss the major threats of the study. Other experimental studies are needed to reduce the limitations identified. However, this paper presents results that motivate further research on the topic. The main motivational factors are: the potential of automation, cost reduction, and the ability to detect faults as well as the good acceptance by groups of professionals and students.

7 Conclusion

This paper presented an experimental study to evaluate the use of Model-Based Testing (MBT) in the modeling, generation, concretization, and execution of automated tests in mobile applications. We adopted the technique Event Sequence Graph (ESG) to design the test model and to express the mobile application requirements and features to be tested. The research was focused on mobility solutions developed in the Google Android platform and, therefore, test cases were implemented using the Robotium framework.

The analyzed results provided valid information for research questions exposed by this paper and it was identified the possibility of applying MBT as a recommended practice for testing Android applications. The use of MBT provides some

advantages, such as automatic generation of test cases, fault detection capability, improved test quality, reduced time and cost for testing, and evolution of test models. Some challenges were also noted with emphasis on the test modeling difficulties, tests concretization particularities in the context of mobility, and the need for expertise in specific tools. The experimental study provides some initial evidences that MBT along with ESG modeling technique can be used as a systematic approach to test Android applications.

Further research can be conducted in the emerging field of automated tests in mobile applications. An initiative is to simplify and provide a partial or complete generation of concrete test cases. In future work, other experimental studies could be performed to analyze the fault detection capabilities, as well as time and cost when compared with manual testing. Specifically, we plan to evaluate the number of detected faults, the time to design and run tests, the effort to learn a supporting tool (like Robotium), and so on.

References

[1] Aho, A. V., A. T. Dahbura, D. Lee and M. U. Uyar, *An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours*, in: R. J. Linn and M. U. Uyar, editors, *Conformance Testing Methodologies and Architectures for OSI Protocols*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1995 pp. 427–438.
URL http://dl.acm.org/citation.cfm?id=202035.202073

[2] Amalfitano, D., A. R. Fasolino, P. Tramontana, B. Ta and A. Memon, *Mobiguitar - a tool for automated model-based testing of mobile apps*, IEEE Software **99** (2014), p. 1.

[3] Android.com, *Android developers* (2014).
URL http://developer.android.com

[4] Apple.com, *Apple - ios 7* (2014).
URL https://www.apple.com/ios/

[5] Beizer, B., "Software Testing Techniques," Van Nostrand Reinhold Co., New York, NY, USA, 1990, 2nd edition.

[6] Belli, F., C. J. Budnik and L. White, *Event-based modelling, analysis and testing of user interactions: approach and case study*, Software Testing, Verification and Reliability (STVR) **16** (2006), pp. 3–32.
URL http://dx.doi.org/10.1002/stvr.335

[7] Blackburn, M., R. Busser and A. Nauman, *Why model-based test automation is different and what you should know to get started*, in: *International Conference on Practical Software Quality and Testing, Software Productivity Consortium*, 2004, pp. 212–232.

[8] Bo, J., L. Xiang and G. Xiaopeng, *Mobiletest: A tool supporting automatic black box test for software on smart mobile devices*, in: *Proceedings of the 2nd International Workshop on Automation of Software Test (AST)*, 2007, pp. 8–8.

[9] Bouquet, F., S. Debricon, B. Legeard and J.-D. Nicolet, *Extending the unified process with model-based testing* (2006).

[10] de Figueiredo, A. L. L., W. L. Andrade and P. D. L. Machado, *Generating interaction test cases for mobile phone systems from use case specifications*, ACM SIGSOFT Software Engineering Notes (SEN) **31** (2006), pp. 1–10.
URL http://doi.acm.org/10.1145/1218776.1218788

[11] Deitel, P. J., H. M. Deitel, A. Deitel and M. Morgano, "Android for Programmers: An App-Driven Approach," Prentice Hall Press, Upper Saddle River, NJ, USA, 2012, 1st edition.

[12] Delamaro, M. E., A. M. R. Vincenzi and J. C. Maldonado, *A strategy to perform coverage testing of mobile applications*, in: *Proceedings of the 2006 International Workshop on Automation of Software Test (AST)*, AST '06 (2006), pp. 118–124.
URL http://doi.acm.org/10.1145/1138929.1138952

[13] do Nascimento, L. H. O. and P. D. L. Machado, *An experimental evaluation of approaches to feature testing in the mobile phone applications domain*, in: *Proceedings of the Domain-Specific Approaches to Software Test Automation: In Conjunction with the 6th ESEC/FSE Joint Meeting*, DOSTA '07 (2007), pp. 27–33.
URL http://doi.acm.org/10.1145/1294921.1294926

[14] El-Far, I. K. and J. A. Whittaker, *Model-based software testing*, in: *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc., 2002 pp. 825–837.
URL http://dx.doi.org/10.1002/0471028959.sof207

[15] Endo, A. T., "Model-based testing of service oriented applications," Ph.d. dissertation, ICMC, USP, São Carlos, SP (2013).

[16] Espresso, *Espresso - android test kit* (2014).
URL https://code.google.com/p/android-test-kit/wiki/Espresso

[17] Fundamentals, A. T., *Android testing framework* (2014).
URL http://developer.android.com/tools/testing/testing_android.html

[18] Gartner, I., *Gartner says worldwide tablet sales grew 68 percent in 2013, with android capturing 62 percent of the market* (2014).
URL http://www.gartner.com/newsroom/id/2674215

[19] Grieskamp, W., N. Kicillof, K. Stobie and V. Braberman, *Model-based quality assurance of protocol documentation: Tools and methodology*, Software Testing, Verification and Reliability (STVR) **21** (2011), pp. 55–71.
URL http://dx.doi.org/10.1002/stvr.427

[20] Hierons, R. M., K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward and H. Zedan, *Using formal specifications to support testing*, ACM Computing Surveys (CSUR) **41** (2009), pp. 1–76.
URL http://doi.acm.org/10.1145/1459352.1459354

[21] Hu, C. and I. Neamtiu, *Automating gui testing for android applications*, in: *Proceedings of the 6th International Workshop on Automation of Software Test (AST)*, AST '11 (2011), pp. 77–83.
URL http://doi.acm.org/10.1145/1982595.1982612

[22] Jensen, C. S., M. R. Prasad and A. Møller, *Automated testing with targeted event sequence generation*, in: *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA)*, ISSTA 2013 (2013), pp. 67–77.
URL http://doi.acm.org/10.1145/2483760.2483777

[23] Kumar Maji, A., K. Hao, S. Sultana and S. Bagchi, *Characterizing failures in mobile oses: A case study with android and symbian*, in: *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, 2010, pp. 249–258.

[24] Lee, D. and M. Yannakakis, *Principles and methods of testing finite state machines - a survey*, Proceedings of the IEEE **84** (1996), pp. 1090–1123.

[25] Linschulte, M., "On the Role of Test Sequence Length, Model Refinement, and Test Coverage for Reliability," Ph.d. dissertation, Universität Paderborn, Paderborn (2013).

[26] Liu, Y., C. Xu and S.-C. Cheung, *Characterizing and detecting performance bugs for smartphone applications*, in: *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014 (2014), pp. 1013–1024.
URL http://doi.acm.org/10.1145/2568225.2568229

[27] Muccini, H., A. Di Francesco and P. Esposito, *Software testing of mobile applications: Challenges and future research directions*, in: *Proceedings of the 7th International Workshop on Automation of Software Test (AST)*, 2012, pp. 29–35.

[28] Myers, G. J., C. Sandler and T. Badgett, "The Art of Software Testing," Wiley Publishing, 2011, 3rd edition.

[29] Pathak, A., A. Jindal, Y. C. Hu and S. P. Midkiff, *What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps*, in: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, MobiSys '12 (2012), pp. 267–280.
URL http://doi.acm.org/10.1145/2307636.2307661

[30] Pretschner, A. and J. Philipps, *10 methodological issues in model-based testing*, in: M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker and A. Pretschner, editors, *Model-Based Testing of Reactive Systems*, Lecture Notes in Computer Science **3472**, Springer Berlin Heidelberg, 2005 pp. 281–291.
URL http://dx.doi.org/10.1007/11498490_13

[31] Robolectric, *Robolectric - test-drive your android code* (2014).
URL http://robolectric.org

[32] Robotium, *Robotium - the world's leading android test automation framework* (2014).
URL https://code.google.com/p/robotium

[33] Selendroid, *Selendroid - selenium for android* (2014).
URL http://selendroid.io

[34] Sinha, A. and C. Smidts, *Hottest: A model-based test design technique for enhanced testing of domain-specific applications*, ACM Transactions on Software Engineering and Methodology (TOSEM) **15** (2006), pp. 242–278.
URL http://doi.acm.org/10.1145/1151695.1151697

[35] Testing, A. U., *Ui testing* (2014).
URL http://developer.android.com/tools/testing/testing_ui.html

[36] Tretmans, J., *Conformance testing with labelled transition systems: Implementation relations and test generation*, Computer Networks and ISDN Systems **29** (1996), pp. 49–79.
URL http://dx.doi.org/10.1016/S0169-7552(96)00017-7

[37] Utting, M. and B. Legeard, "Practical Model-Based Testing: A Tools Approach," Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[38] Utting, M., A. Pretschner and B. Legeard, *A taxonomy of model-based testing approaches*, Software Testing, Verification and Reliability (STVR) **22** (2012), pp. 297–312.
URL http://dx.doi.org/10.1002/stvr.456

[39] WindowsPhone.com, *Windows phone* (2014).
URL http://www.windowsphone.com

[40] Yang, S., D. Yan and A. Rountev, *Testing for poor responsiveness in android applications*, in: *1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS)*, 2013, pp. 1–6.

[41] Yuan, X., M. Cohen and A. Memon, *Gui interaction testing: Incorporating event context*, IEEE Transactions on Software Engineering **37** (2011), pp. 559–574.