Implementation of a Test Data Generator based on DSL Files

Paul POCATILU, Alin ZAMFIROIU
Bucharest University of Economic Studies
ppaul@ase.ro, zamfiroiu@ici.ro

In software testing process, test data generation represents an important step for high quality software, even for mobile devices. As proposed in previous works, a potential source for random data generation is represented by the UI layout files that are used for almost all mobile platforms (Android, iOS, Windows Phone/Mobile). This paper continues the previous work and presents a test data generation system based on Android layout files. The test data generator uses DSL files as input and generates test data that conform to several testing principles. The generated test data could be stored in XML files or any format required by the testing frameworks.
Keywords: *Mobile Applications, UI Layout Files, Software Testing, Test Data Generators, Software Quality, Unit Testing*

1 Introduction

Software testing represents an important step in software development [1], [2] and the testing process is thoroughly presented in books like [3] and [4]. Like other applications, mobile applications require testing in order to achieve a required level of quality. This can be done using similar tools and frameworks and also specific and dedicated tools, depending on the platform. Some of the mobile application testing types are shortly described in [5].

During the testing process, test data generation has its own role for testing success. Test data generation is made using different tools and techniques. The paper continues the research presented in [6], [7] and [17] and focuses on template generation for test data based on Android layout files. Test data templates are XML-based files written using DSL (Data Specification Language). The generated test data can be used by own testing tools or frameworks or can be used as inputs for existing testing frameworks and tools.

The paper is structured as follows. The section *Android testing frameworks and tools* presents the most important aspects related to Android applications testing. It also make a short presentation of the Android testing instruments. *Data Specification Language (DSL)* section describes the proposed system for test data generator based on Android layout files and details the XML-based language used for test data specification. The proposed parser for Android layout files is presented in the last section, *Android layout files parser*. The section *Random test data generator based on pragmatic testing (RIGHT-BICEP)* presents a test data generator, based on DSL files, that allows to generate data that conform to selected principles. The paper ends with conclusion and future work.

2 Android Testing Frameworks and Tools

Android applications being developed using Java programming language, *JUnit* testing framework is suitable for the automated testing of functional issues. JUnit is a testing framework for regressive unit testing of Java programs [8]. The main Java classes used by the framework are associated to test cases and suites.

The Android platform includes several tools and frameworks. Also, third party developers have built such tools and framework for Android applications testing. In [9] are presented the fundamentals of Android applications testing.

Table 1 summarizes the most used Android testing tools and frameworks. Many of these are based on JUnit.

DOI: 10.12948/issn14531305/20.2.2016.03

Table 1. Android testing tools and frameworks

Framework/Tool	Included in Android API	Testing level	Testing approach	Type
Espresso	Yes	UI testing	Black box	Framework
Android Instrumentation API	Yes	Unit testing	White box	Framework
Monkey	Yes	UI testing	Black box	Tool
monkeyrunner	Yes	UI testing	Black box	Tool
Robotium	No	UI testing	Black box	Framework
Robolectric	No	Unit testing	White box	Framework
UI Automator	Yes	UI testing	Black box	Framework

Espresso is a testing framework used for UI testing. It is based on JUnit and it is included in Android Support Repository.

Android Instrumentation framework is developed by Google for testing Android applications. It allows to control the life cycle of the Android applications and components during the tests.

Monkey is a testing tool that runs within the emulator or on the mobile device. It is used to send random events to the Android device (user or system).

monkeyrunner provides an API that can be used to control an Android device by installing applications, sending commands and taking and saving screenshots.

Robotium is a testing framework for Android and hybrid application. It is based on Android Instrumentation and it is used to automate UI testing.

Robolectric is framework that allows testing on a JVM running on a computer. This will speed-up the testing process.

UI Automator framework provides an API that allows to control user and system applications for UI testing.

In [10] there is a short presentation of several other Android testing frameworks and tools:

- *Mockito* – a framework for testing Java and Android applications; it allows creation of mock objects for testing and it is used in unit testing;
- *EasyMock* – a testing framework used in unit testing; it uses mock objects;
- *PowerMock* – a framework based on Mockito and EasyMock;
- *Inifinitest* – a testing plugin for Eclipse and IntelliJ; it is a continuous test runner.

In order to automate the testing process, some of these tools can run using generated data by dedicated tools.

Specific Android testing approaches are presented in [11], [12] and [13].

3 Data Specification Language (DSL)

In order to generate test data, test data generators (TDG) can be based on random functions or can use inputs related to application under test (specifications, source files, data constraints, list of values, layout files etc.).

The test data will be used either for white-box testing (as in [14]) or for functional testing. White-box testing requires a deeper knowledge of source code and a previous analysis of it is required before [15].

Our proposed solution take as input an XML-based file that includes a description of each field for which will be generated test data.

Figure 1 depicts the architecture of the test data generator system. Android layout files are used as inputs for the parser. The parser generates a DSL file that is used as input for the test data generator. Finally, the test data generator will provide the test data.

DOI: 10.12948/issn14531305/20.2.2016.03

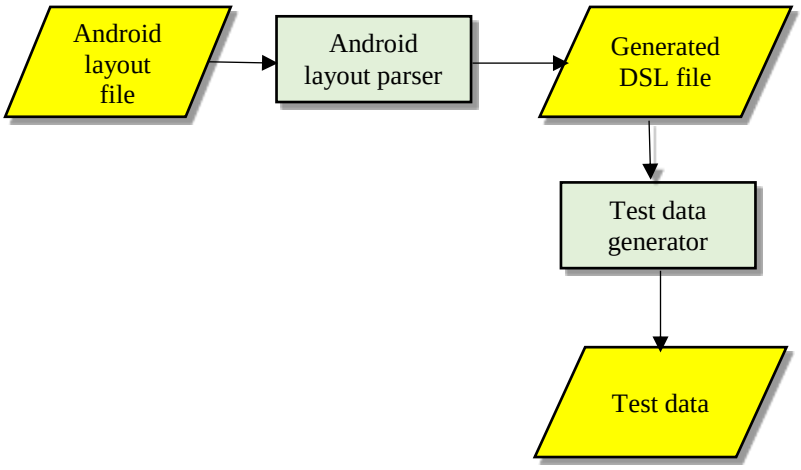

Fig. 1. Test data generator system

The DSL file provides required information to test data generator and allows to generate test data for the analyzed software under test (SUT). The generated test data could be stored in memory or in files (XML, binary or any other specific format).

In [6] and [7] was proposed an XML-based language used for test data generation. The current version include more nodes for a better control of data generation. The root node any DSL file is *dataset*. Each field for which data will be generated is represented by *field* node. The field node include tow attributes:

- *id*, associated to the identification attribute of the current field;
- *type*, representing the control type (EditText, Spinner etc.).

Each field includes the nodes:

- *type*, representing data type like string, number, Boolean etc.;
- *generation*, showing how data will be generated (random or a list of values);
- *maxLength*, with attribute *fixed*, used for the required length.

The fields that require values from a list of values will include the *lov* node with values used for selection.

The XSD schema of DSL files is presented in Listing 1.

Listing 1. DSL files XSD schema

```
<xs:schema attributeFormDefault="unqualified"
        elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="dataset">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="field" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="type"/>
              <xs:element type="xs:string" name="generation"/>
              <xs:element name="maxLength" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="fixed" use="optional"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="lov" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element
                       type="xs:string" name="item" maxOccurs="unbounded" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
```

DOI: 10.12948/issn14531305/20.2.2016.03

```
            </xs:sequence>
                <xs:attribute type="xs:string" name="id" use="required"/>
                 <xs:attribute type="xs:string" name="type" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

If the control identifier is not present, the system will generate one.

Data length will be deduced from the Android UI layout files, such as *android:maxLength* attribute.

Data type could be determined based on *android:inputType* and initial fields values. For input that include numbers and a specific format (like phone numbers, date etc.), it should be included the format also.

Also, for numeric fields, it could be added the nodes *minValue* and *maxValue* and their corresponding values obtained from the layout file or specifications or could be added later.

4 Android Layout Files Parser

The Android layout parser uses XML-based files available in *res/layout* folder of the Android project. Several sources, such as [16], present the content and structure of Android layout files.

In order to exemplify the DSL template generation, the XML layout from Listing 2 was used.

Listing 2. Android layout file used as example

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android= "http://schemas. android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <!-- Author label here-->
    <EditText
       android:id="@+id/editAutor"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:inputType="textCapWords"/>
    <!--Title label here -->
    <EditText
       android:id="@+id/editTitlu"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:inputType="text"/>
     <!--Date lable here -->
     <EditText
       android:id="@+id/data"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:inputType="date"/>
     <!-- Publisher label here -->
     <EditText
       android:id="@+id/editEditura"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:inputType="text"/>
     <!--ISBN label here-->
     <EditText
       android:id="@+id/editIsbn"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>
     <!--Price label here -->
     <EditText
        android:id="@+id/editPret"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="0"
```

DOI: 10.12948/issn14531305/20.2.2016.03

```
        android:inputType="number"/>
    <!-- -->
    <Spinner
      android:id="@+id/spinGen"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"/>
    <!--Status label here -->
    <CheckBox
      android:id="@+id/checkUzata"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
   <Button
       android:id="@+id/buttonSalveaza"
       android:layout_height="wrap_content"
       android:layout_width="wrap_content"
       android:layout_gravity="center"
       android:text="Salveaza" />
  </LinearLayout>
</ScrollView>
```

The layout includes eight controls for which test data need to be generated: six *EditText* controls, one *Spinner* control and one *CheckBox* control. Three *EditText* controls include *android:inputType* attributes with values: *textCapWords*, *text*, and *number*. Figure 2 presents the actual layout used as example running on a real device.

Fig. 2. Test data generator system

The generated test data could be used by existing tools and frameworks to fill the controls and to activate the submission button. Based on the layout from Listing 2, the generated DSL file is presented in Listing 3. Current version includes mostly random values generation and list of values (checked and unchecked) for *CheckBox* controls.

Listing 3. Generated DSL file

```
<dataset>
  <field type="EditText" id="editAutor">
    <type>string</type>
```

DOI: 10.12948/issn14531305/20.2.2016.03

```
    <generation>random</generation>
    < maxLength fixed="No" />
  </field>
  <field type="EditText" id=" editTitlu ">
    <type>string</type>
    <generation>random</generation>
    <maxLength fixed="No">40</length>
  </field>
  <field type="EditText" id=" data ">
    <type>string</type>
    <generation>random</generation>
    < maxLength fixed="No" />
  </field>
  <field type="EditText" id=" editEditura ">
    <type>string</type>
    <generation>random</generation>
    < maxLength fixed="No" />
  </field>
<field type="EditText" id=" editIsbn ">
    <type>string</type>
    <generation>random</generation>
    < maxLength fixed="No" />
  </field>
  <field type="EditText" id=" editPret ">
    <type>number</type>
    <generation>random</generation>
    < maxLength fixed="No" />
  </field>
  <field type="Spinner" id=" spinGen ">
    <type>string</type>
    <generation>lov</generation>
  </field>
  <field type="CheckBox" id=" checkUzata ">
    <type>boolean</type>
    <generation>lov</generation>
    <lov>
      <item>checked</item>
      <item>unchecked</item>
    </lov>
  </field>
</dataset>
```

This DSL file represents an input for the test data generator. In this stage, the DSL file does not fully automate test data generators. It could require a manual intervention or other additional parsers or editors that need to narrow data boundaries or add other constraints or will provide the list of values for list-based controls. For example, the for the *Spinner* control, the list of values has to be filled before data generation.

5 Random Test Data Generator Based On Pragmatic Testing (RIGHT-BICEP)

The DSL files will be used for generate the data set for testing. In order to generate these datasets it is necessary to know what will be tested with these datasets. The test data generator we propose will use, for unit testing, the Right-BICEP principle, Figure 3.

DOI: 10.12948/issn14531305/20.2.2016.03

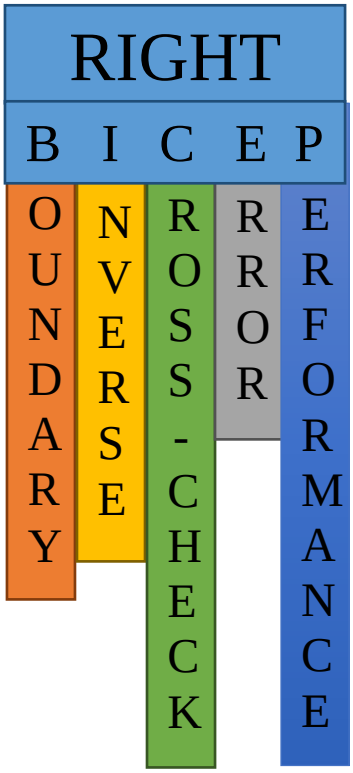

Fig. 3. Right-BICEP principle

According to this principle, for each control must be tested and verified if the input provided by the user comply [8]:

- **RIGHT** – if the provided control input is correctly; it should be verified if the input of the control is correct and is not changed;
- **Boundary** – all limits should be checked and if the input is correctly obtained for these limits; it should be verified if the control provide the correct input with minimum length or with maximum accepted length;
- **Inverse** – inverse relationship must be verified for each control; it should be verified by an inverse relationship if the control provide the correct input;
- **Cross-check** – it must be verified by a cross-check accuracy, using calculation methods similar tested and validated by a large community of programmers and compare results;
- **Error** – check control behavior when obtaining certain errors or introduce erroneous values by the user; the control should be forced to provide errors and to be analyzed the reaction of the control in these situations;
- **Performance** – verifying the optimal functioning of that control, otherwise it is strongly recommended another type of control.

Besides this principle exist another one named **CORRECT**, which should be considered in the process of testing and in the process of elaboration of dataset for testing.

Following the principles of testing and using the DSL file layouts obtained for Android was developed Random Test Data Generator (RTDG) application. The application generates an XML file with test data fields described in the file DSL.

For each test data file with the desired DSL file uploaded it is specified the number of tests required for each field and what type of tests to generate. The RTDG will generate test data for three of the six categories of Right-BICEP principle: Right, Boundary and Error. Figure 4 depicts the interface of the RTDG application.

DOI: 10.12948/issn14531305/20.2.2016.03

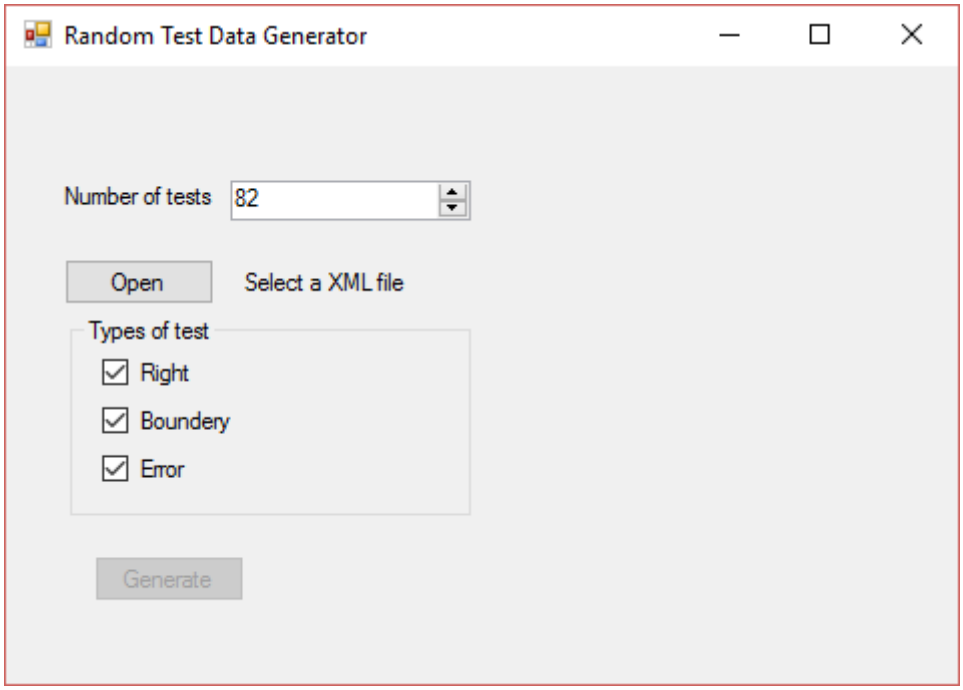

Fig. 4. Random Test Data Generator Application

Based on each identified field of DSL file it is determined the control type and what types of test data should be generated. For example, if the identified control type is *EditText*, then:

- the data type could be either *String* or *numeric*,
- the *id* attribute is taken from the control related attribute (android:id) or it is generated;
- the maximum length of characters accepted is determined, if it is specified. If the *maxLenght* tag is not specified it is considered a length of 20 characters. This length is used to build test data for Boundary principle.

After all the fields were identified, random test data will be generated. If the field is of type String it may also contain characters and numbers but also special signs. If that field is numeric, it contains only numbers.

Based on DSL file from Listing 3 it is generated the XML file from Listing 4. The file contains one set of test data for each *EditText* control, for each category (Right, Boundary, and Error).

Listing 4. Generated data test file

```
<?xml version="1.0" encoding="UTF-8"?>
<teste>
    <test control="editAutor" type="RIGHT">
        <input ERROR="false">t0l06,1/7*76</input>
        <expectedResult>t0l06,1/7*76</expectedResult>
    </test>
    <test control="editAutor" type="Boundary">
        <input ERROR="false">3o27984(9;0q73;mz8x3</input>
        <expectedResult>3o27984(9;0q73;mz8x3</expectedResult>
    </test>
    <test control="editAutor" type="Error">
        <input ERROR="true">j5gz8/!12<u<l:%)8634</input>
    </test>
    <test control="editTitlu" type="RIGHT">
        <input ERROR="false">mi6?=]8o*6bz</input>
        <expectedResult>mi6?=]8o*6bz</expectedResult>
    </test>
    <test control="editTitlu" type="Boundary">
        <input ERROR="false">mq_85&)0,s</z,<f1b3fac~_;&)o10-ts;56~09c</input>
        <expectedResult>mq_85&)0,s</z,<f1b3fac~_;&)o10-ts;56~09c</expectedResult>
    </test>
    <test control="editTitlu" type="Error">
        <input ERROR="true">c/_?67m08bbjl.z,84u3t/,-?abf+*2-+jf]vjj4&</input>
```

DOI: 10.12948/issn14531305/20.2.2016.03

```
    </test>
    <test control="data" type="RIGHT">
        <input ERROR="false">u:7o2q:2tk%8a@16</input>
        <expectedResult>u:7o2q:2tk%8a@16</expectedResult>
    </test>
        <test control="data" type="Boundary">
        <input ERROR="false">h;s;{mdm%o9xy5!m]ptq</input>
        <expectedResult>h;s;{mdm%o9xy5!m]ptq</expectedResult>
    </test>
    <test control="data" type="Error">
        <input ERROR="true">6bqwlx`3=bu_;94$6&:h</input>
    </test>
    <test control="editEditura" type="RIGHT">
        <input ERROR="false">81194y8>(e</input>
        <expectedResult>81194y8>(e</expectedResult>
    </test>
    <test control="editEditura" type="Boundary">
        <input ERROR="false">i=(dg*=~js/b^.lm!g#f</input>
        <expectedResult>i=(dg*=~js/b^.lm!g#f</expectedResult>
    </test>
    <test control="editEditura" type="Error">
        <input ERROR="true">8i`b;h42._1/5&1^>3}$;tfu`i84</input>
    </test>
    <test control="editIsbn" type="RIGHT">
        <input ERROR="false">x4]<k:ez68/g(09ig.9</input>
        <expectedResult>x4]<k:ez68/g(09ig.9</expectedResult>
    </test>
    <test control="editIsbn" type="Boundery">
        <input ERROR="false">c0.0:j${g2jtel>376z_</input>
        <expectedResult>c0.0:j${g2jtel>376z_</expectedResult>
    </test>
    <test control="editIsbn" type="Error">
        <input ERROR="true">o9y_32n:.-@->u6hiu:%07!b7ey/</input>
    </test>
    <test control="editPret" type="RIGHT">
        <input ERROR="false">z7c#y&#,2t7[</input>
        <expectedResult>z7c#y&#,2t7[</expectedResult>
    </test>
    <test control="editPret" type="Boundary">
        <input ERROR="false">d&#me,6x52u5g9$kv07b</input>
        <expectedResult>d&#me,6x52u5g9$kv07b</expectedResult>
    </test>
    <test control="editPret" type="Error">
        <input ERROR="true">0-k:h/46b68yq@n$5.~v0,55ln</input>
    </test>
</teste>
```

Within the XML file, test data is organized by controls. For each control, test data are generated in order to comply with each principle selected in the application. Each set contains the *test* tag, the control *id* for and the type of test that is generated. If the test is Right or Boundary, the tag *input* contains the attribute *ERROR* with the *false* value and the value represents the input data set to achieve for that test. If the selected test type is Boundary or Right, *expectedResult* tag exists and represent the expected result from the test. For the Error tests, the value of the attribute *ERROR* is true and the tag *expectedResult* doesn't exists, because the expected result is the error.

The proposed Random Test Data Generator application can be improved so as to generate other files with test data and not only XML. Also, the structure of the resulting file can be customized depending on the application framework used to automate the testing process based on test data provided by RTDG application.

6 Conclusions and Future Work

The proposed system can be integrated with many testing frameworks and tools available for Android platform. The presented format of DSL files is a preview and it will be improved during the future development. The proposed application for random test data generation demonstrates the functionality of the proposed system.

The next steps include the further development of the test data generator that generate test data based on DSL files: data output format, more testing criteria.

DOI: 10.12948/issn14531305/20.2.2016.03

Acknowledgment
Parts of this research have been published in the Proceedings of the 14th International Conference on Informatics in Economy, IE 2015 [17].

References

[1] S. Pressman, *Software Engineering: A Practitioner's Approach. 7th ed*., New York: McGraw-Hill, 2009
[2] I. Sommerville, *Software Engineering. 9th ed*., Boston: Addison-Wesley, 2011
[3] G. J. Myers, C. Sandler, T. Badgett, *The Art of Software Testing*, 3rd Edition, Wiley, 2011
[4] M. Roper, *Software Testing*, McGraw-Hill Book, 1994
[5] M. Kumar and M. Chauhan, "Best Practices in Mobile Application Testing (White Paper)," Infosys, Bangalore, 2013
[6] P. Pocatilu, F. Alecu and S. Capisizu, "A Test Data Generator for Mobile Applications," in Proc. of the IE 2014 International Conference, Bucharest, Romania, May 15-18, 2014, pp. 116-121
[7] P. Pocatilu and F. Alecu, "An UI Layout Files Analyzer for Test Data Generation," Informatica Economica, vol. 18, no. 2/2014, pp. 53-62
[8] J. Langr, A. Hunt and D. Thomas, *Pragmatic Unit Testing in Java 8 with JUnit*, The Pragmatic Programmers, 2015
[9] Testing Fundamentals | Android Developers, available at: http://developer.android.com/tools/testing/testing_android.html
[10] P. Pocatilu, I. Ivan et al, *Programarea aplicațiilor Android*, Bucharest: ASE Publishing House, 2015
[11] S. Yang, D. Yan and R. Rountev, "Testing for poor responsiveness in Android applications," in Proc. of the 1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS), 2013, pp. 1 – 6
[12] A. Gupta, *Learning Pentesting for Android Devices*, Packt Publishing, 2014
[13] W. Choi, G. Necula and K. Sen, "Guided GUI testing of Android apps with minimal restart and approximate learning," in Proc. of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (OOPSLA '13). ACM, New York, NY, USA, pp. 623-640
[14] S. Jiang, Y. Zhang and D. Yi, "Test Data Generation Approach for Basis Path Coverage," ACM SIGSOFT Software Engineering Notes, vol. 37, no. 3, pp. 1-7, 2012
[15] A. Zamfiroiu, "Source Code Quality Metrics Building for Mobile Applications," in proc. of the IE 2014 International Conference, Bucharest, pp. 136-140
[16] R. Meier, *Professional Android 4 Application Development*, Wiley, 2012
[17] P. Pocatilu, Capisizu, " A Test Data Generator based on Android Layout Files," in Proc. of the IE 2015 International Conference, Bucharest, Romania, 30 April – 3 May 2015, pp. 135-140

Paul POCATILU graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 1998. He achieved the PhD in Economics in 2003 with thesis on Software Testing Cost Assessment Models. He has published as author and co-author over 45 articles in journals and over 40 articles on national and international conferences. He is author and co-author of 10 books, (Mobile Devices Programming and Software Testing Costs are two of them). He is professor at the Department of Economic Informatics and Cybernetics within the Bucharest University of Economic Studies, Bucharest. He teaches courses, seminars and laboratories on Mobile Devices Programming, Economic Informatics, Computer Programming and Project Management to graduate and postgraduate students. His

DOI: 10.12948/issn14531305/20.2.2016.03

current research areas are software testing, software quality, project management, and mobile application development.

Alin ZAMFIROIU has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2009. In 2011 he has graduated the Economic Informatics Master program organized by the Bucharest University of Economic Studies and in 2014 he finished his PhD research in Economic Informatics at the Bucharest University of Economic Studies. Currently he works like a Senior Researcher at "National Institute for Research & Development in Informatics, Bucharest". He has published as author and co-author of journal articles and scientific presentations at conferences.

DOI: 10.12948/issn14531305/20.2.2016.03