Keyword-Driven Testing Framework For Android Applications

Wu Zhongqian, Liu Shu, Li Jinzhe, Liao Zengzeng
School of Software
Harbin Institute of Technology
Harbin, China
{imzhongqian, lijinze909, liaozengzeng} @163.com
sliu@hit.edu.cn

***Abstract*—Automated testing of the mobile applications is just getting started recently. Currently it is still in the discussion and research stage. However, from the growth in the number of mobile applications and logical complexity, and the crunch of the product release cycle, the automated testing is on demand. Compared with the rapid development of Android application, the traditional manual testing methods are lag behind. Due to the openness of Android and the variety of devices, the manual testing is limited by many factors. Especially in regression testing of iteration development the manual testing is just inadequate. In order to solve such problems as well as to improve the reusability of testing scripts, this paper proposes an Android-based keyword-driven automated testing framework. Based on Robotium and with keyword-driven this testing framework separates testing logic, testing scripts, and testing data in design. Tests can be done only through modifying the control files. Meanwhile, testing scripts are independent with testing examples. The test data and business logic are being integrated in the form of test data by designing reusable keyword library based on the Android GUI testing. Thus, the design of testing can be simplified to be the design of testing data table and minimize the manual operations. Further, it can free the test engineers from the tedious repetitive work, provide a more efficient and accurate test software products, and improve the competitiveness of products.**

Keywords-keyword-driven; automated testing; test framework; android GUI test

I. INTRODUCTION

The arrival of mobile internet era provides the opportunity for the smart phone applications to develop by leaps and bounds and make it the mainstream of today's mobile applications market. While the phones bring convenience to the users, software failures and problems also increase. The importance of software quality is gradually revealed. The losses will increase if software quality is not improved with the growth of the system size, complexity and importance. [1]

To improve the quality of mobile phone, the software testing plays an important role. Adequate testing of the software by the software developers and users to make sure that it works properly and meets the requirement specification. Statistics show that in a typical software project, testing effort often accounts for more than 40% of the total development workload. If the problem can be detected early and resolved before release, the production cost will be controlled more effectively. In fact, the cost of fixing bugs after the release is 200-300 times as much as it does in the testing phase [2]. So it is essential to improve the efficiency of the software testing process.

To improve the efficiency of the test, automated testing ideas and methods have been introduced. Practice has proved that the software automation test technology has helped improving the speed and efficiency of the software testing, saving the cost of software testing and shortening product release cycles. Meanwhile, automated testing technology has completed the work that manual testing cannot achieve. For example, the use of automated testing tools in the testing activities can reduce part of the overhead, at the same time, some testing activities are difficult to achieve and measured manually; automated testing framework can improve the efficiency of the test, quickly locate the functionality and performance defects in all versions of the tested software.

Up to now, the test automation frameworks have evolved into three generations. [3] Figure 1 shows evolution of Test Automation. In the beginning, there was record and playback script creation. In this, there were only stand-alone test scripts. And then, came the Functional Decomposition. It consists of reusable functional test modules. After that came the data-driven testing. In this, test data is taken out of the scripts. This makes the test data variation easy and similar test cases can be created quickly. [4]

Today, keyword-driven testing is getting more and more popular. The keyword-driven testing framework has better reusability than others. And with the increasing amount of testing and the accumulating of test code, its superiority is even more obvious. [5] It is a technique that separates much of the programming work from the actual test steps so that the test steps can be developed earlier and can be maintained with only minor updates. It consists of test scripts, keyword library and data.

However, there isn't a keyword-driven testing framework can be used directly to the android GUI testing now. Android GUI automation test framework has a vital role in safeguarding the rich and mature for Android application. Therefore, in order to narrow for the gap between theory and practice, the article uses Robotium[6] in Android to implement a practical, effective keyword-driven testing framework, and validate its practicability and superiority in the field of automated testing.

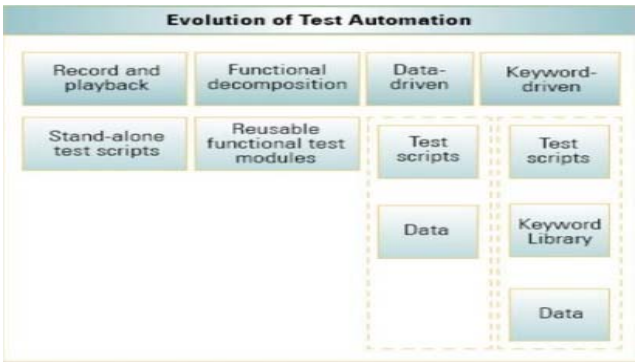


Figure 1. Evolution of Test Automation[7].

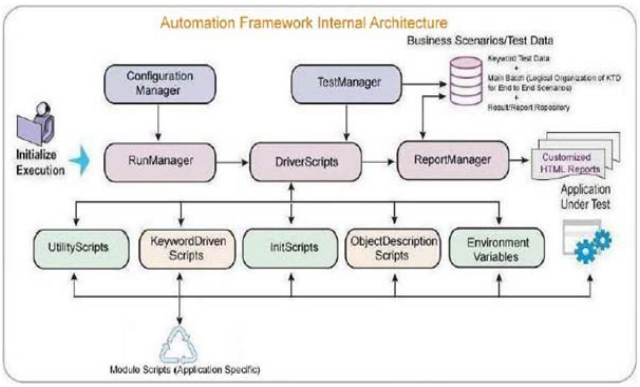


Figure 2. AKDT internal Architecture[9]

II. KEYWORD-DRIVEN TESTING

A. Overview

Keyword-driven testing is a part of the test automation discipline. It separates the programming work from the actual test script so that the test scripts can be developed without knowledge of programming. [8]

In keyword-driven testing, each keyword corresponds to an individual testing action like a mouse click, selection of a menu item, keystrokes, opening or closing a window or other actions. A keyword-driven test is a sequence of operations, in a keyword format, that simulate user actions on the tested application. Basically, to perform any testing actions, testers simply drag and drop the keyword that corresponds to the desired operation or they can just record their actions and the keyword-driven test is built for them.

While most automation solutions rely on dedicated automation engineers that convert manual tests to automated scripts and execute them, in Keyword-Driven Testing, test automation only focuses on building and maintaining the infrastructure. The same scripts, written by the testers and other domain knowledge experts, are used for both manual and automated tests. The testers themselves are also responsible for execution and analysis.

As a result the test scripts can be developed earlier and maintained with only minor updates, even when the application or testing requires significant changes.

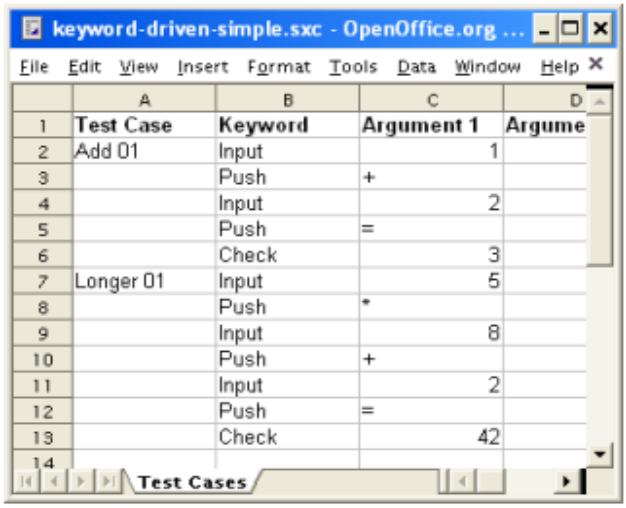
B. Keyword-Driven Architecture

Keyword-Driven framework (called AKDT as follow) like any other framework has been built using certain components to make it work the way it has been described. A graphical presentation of the internal architecture is given by Figure 2.

Keyword-Driven Testing can be divided into two main layers. The Infrastructure Layer is comprised of Utility Scripts and User Defined Functions. It is the engine that receives inputs (keywords) and performs operations on the tested application. The Logical Layer is used by testers and other subject matter experts to build and execute test automation scripts using pre-defined keywords.

Keyword-Driven Testing utilizes a dictionary that provides the entire organization with a language for building test automation scripts. Test automation is fully aligned with the business processes by using this approach.

The implementation of this methodology is framework dependent. This framework requires the development of data tables and keywords, independent of the test automation tool used to execute them and the test script code that "drives" the application-under-test and the data. A graphical presentation of the data tables is given by Figure 3.



	A	B	C	D
1	Test Case	Keyword	Argument 1	Argume
2	Add 01	Input	1	
3		Push	+	
4		Input	2	
5		Push	=	
6		Check	3	
7	Longer 01	Input	5	
8		Push	*	
9		Input	8	
10		Push	+	
11		Input	2	
12		Push	=	
13		Check	42	
14				

Figure 3. Test data file

In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test.

If we were to map out the actions we perform with the mouse when we test our Android Calculator functions by hand, we could create the following table. The "object" column contains the name of the activity where we're performing the mouse action. The "Keyword" column names the type of control the mouse is clicking. And the "Arguments" column names a specific control (1, 2, 3, 5, +, -, and so on).

C. Advantages of AKDT

1) Keyword-Driven Tests are Easy to Create: The testing engineer without knowledge of programming can write the detailed test plan having desired inputs and verification data in the form of simple & convenient spreadsheets. They can create keyword-driven tests visually by adding and deleting operations and edit them directly by changing an operation's parameters and position. However, writing scripts requires knowledge of scripting languages provided by the automated testing tool, whether it is VBScript, Jscript or any other scripting language and requires testers to know the application's internal objects.

2) Create Automated Test Batches With Logic From Keyword-Driven Tests: Keyword-driven tests allow you to introduce logic into the organization of your automated tests by building some simple decision into a keyword-driven test. So improving test coverage by letting the testers create the automated scripts, more tests are covered and misinterpretation of the manual tests is avoided.

3) Create Automated Tests Earlier with Keyword-Driven Testing: With keyword-driven testing, you can create simple functional tests in the earlier stages of development, testing the application, piece-by-piece, and improving your automated testing success rate and do this without having to learn a scripting language. In a word, it can save time by combining the test automation and test documentation into a single effort.

4) Easy to maintain: A single change in the application under test will require only a single change in the infrastructure. The advantages of automated tests are the reusability and therefore, ease of maintenance of tests that has been created at a high level of abstraction.

III. TECHNOLOGIES USED TO IMPLEMENT AKDT

A. Android Instrumentation testing framework

The Android core test environment is the Instrumentation framework [10]. Android supports the instrumentation, which extends the JUNIT TestCase class to provide functional test of Android Activities. In this framework, the test procedure can accurately control the application. By using the instrumentation, developers can create a simulation system object, control applications multiple life cycles, send UI events to the application during execution, and check the state of the program and so on, before the main program starts. Instrumentation framework implements these functions by running the tested app and testing program in the same process. So we can monitor the interaction between the testing system and the application easily. The working principle of android testing framework is shown in Figure 4.

However, the instrumentation framework is too complex. It requires the testers to know about java and android programming. Meanwhile, writing test cases will burn off more time and increase the cost of software testing. In addition, the high coupling let test code need to be amended once the application under test is modified or version updates.

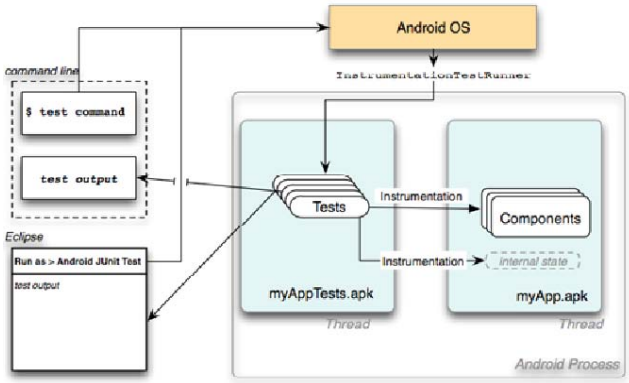


Figure 4. The working principle of instrumentation testing framework

B. Robotium testing frameworks

Robotium is an android open source testing framework and comes with a package of the android test class Instrumentation. It finishes the interactive testing by using Android's instrumentationTestRunner class. Robotium is mainly used to imitate the user scenario testing. By using Robotium, we can easily write robust and effective automated black box and White-box testing.

Robotium can span multiple activities to support functional testing, system testing and acceptance testing. In addition, it also supports Activities, Dialogs, Toasts, Menus, Context Menus control test.

The Open source Robotium framework is convenient to be modified and extend according to requirement. However, Robotium is just a package of instrumentation and more functional expansion is support. For example, it can't reach the full support of custom View.

IV. THE IMPLEMENTATION OF AKDT

A. AKDT's Architecture Design

From the above principle and the requirement, the keyword-driven automated test framework on the Android is shown in figure 5.

The framework mainly divided into five modules: data handle (including test configuration data, business test data), keyword handle, component manager, driver module, result analysis module.

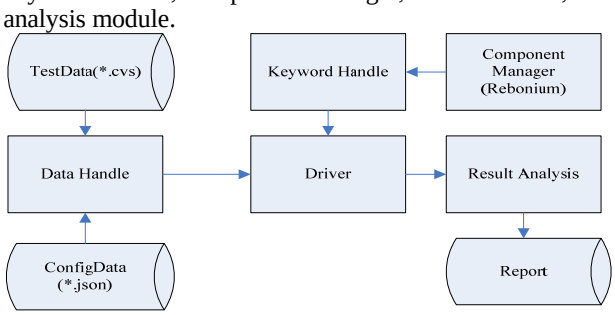


Figure 5. AKDT's Architecture

The configuration file is used to preserve the information they need for automatically test running. This file includes application running environment, tested application path, meta-data file path, testing results saved path, etc. Business test data file, including all kinds of business data, is needed in the test execution process, such as test cases coding, and corresponding operation and input/output data value, etc. Component object data is the collection of tested application control objects, such as a button, a list, a text input box or even a custom controls of Android application. Each object is unique and is identified by ID. Data is read by data handle module from the data files, and according to the need of next level modules, converted to the corresponding format for convenient reading.

The driver module is mainly used for processing different data file. It gets formative data from the data handle module and drives the keyword handle module to test components. Then driver module get test results from keyword handle module and deliver to result analysis module. The key word processor main processing specific operation keyword, including control operation (such as click, long press, drag, jump, etc.) and verify operation (such as verification whether is correct jumped and input correctly verified, etc.). It is the core of the test script code that decides the steps of each operation, the next called operation and so on. Each step needs two corresponding key data: input operation and output operation. The keyword handle module gets the needed information from the control processing module. This module realizes the keyword operation by using Robotium to call a series of control object, such as an operation ENTERTEXT of editing the TextView, which includes two types of operation (text box focus acquisition and text box input). Results analysis module uses the improved TestRunner to produce test analysis report.

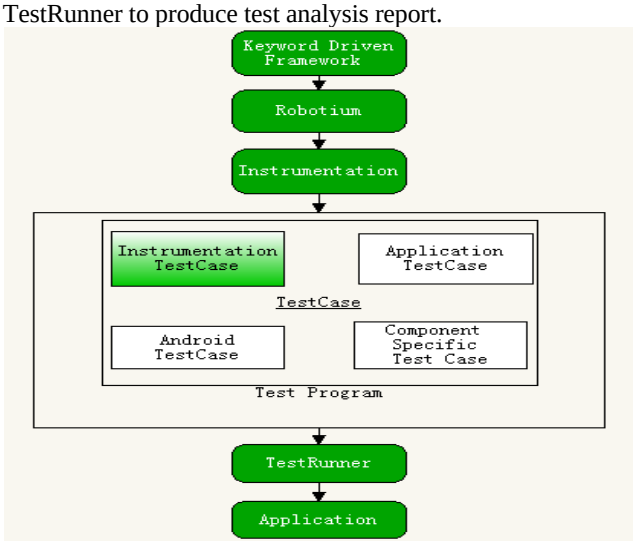


Figure 6. Android testing call framework

The relation among keyword-driven, Robotium and Instrumentation and their calling procedure are shown in figure 6. The TestRunner completes the interaction of android applications by calling the Android SDK original InstrumentationTestRunner. In testing applications, each Activity first will be initialized by Instrumentation, and then is loaded into the Android simulator or Dalvik virtual machine in the device to execute.

When the application is launching, the Instrumentation is in an open state and will be initialized before all application is running. Therefore, we can monitor the interactions between the testing system and the application by using Instrumentation. Implementation is described by the <instrumentation> tags in the project's AndroidManifest.xml file [11]. Only InstrumentationTestCase or its subclasses can use instrumentation method in the test cases.

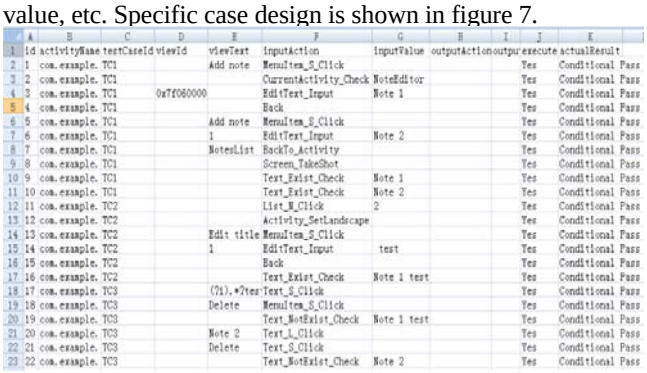
B. *Test Data storing and Processing*

This paper stores data in the style of file and mainly includes test configuration data, business test data and the test report. The test configuration data stores in a JSON file. Business test data will use the table structure way storage, saved to CSV file. Because the test report in addition to record test cases name, it also need to record test cases operation results, operation time, and other properties, so the test report in the way of tree structure can be saved to XML file.

JXL is a toolkit using java to operate Excel. It supports Excel's reading, writing, modification of cell's content and properties and so on. It can be used to generate Excel 2000 standard format files and is compatible with Excel 95, 98 version. In addition, it also supports some images and graphics operation, etc [12]. The most important thing is, JXL API using pure JAVA supporting way, which makes it not dependent on the Windows operating system. This means that even if operation in other operating system, it is also able to correctly deal with Excel form file. Therefore, this paper uses JXL to realize the CSV file processing in Android system.

C. *Business data design*

Business test data file contains all kinds of business data needed in the test execution process, such as test cases coding, and corresponding operation and input/output data value, etc. Specific case design is shown in figure 7.



	A	B	C	D	E	F	G	H	I	J	K
1	id	activityName	testCaseId	viewId	viewText	inputAction	inputValue	outputAction	outpu	execute	actualResult
2	1	com.example.	TC1		Add note	MenuItem_S_Click				Yes	Conditional Pass
3	2	com.example.	TC1			CurrentActivity_Check	NoteEditor			Yes	Conditional Pass
4	3	com.example.	TC1	0x7f060000		EditText_Input	Note 1			Yes	Conditional Pass
5	4	com.example.	TC1			Back				Yes	Conditional Pass
6	5	com.example.	TC1		Add note	MenuItem_S_Click				Yes	Conditional Pass
7	6	com.example.	TC1		1	EditText_Input	Note 2			Yes	Conditional Pass
8	7	com.example.	TC1		NotesList	BackTo_Activity				Yes	Conditional Pass
9	8	com.example.	TC1			Screen_TakeShot				Yes	Conditional Pass
10	9	com.example.	TC1			Text_Exist_Check	Note 1			Yes	Conditional Pass
11	10	com.example.	TC1			Text_Exist_Check	Note 2			Yes	Conditional Pass
12	11	com.example.	TC2			List_N_Click	2			Yes	Conditional Pass
13	12	com.example.	TC2			Activity_SetLandscape				Yes	Conditional Pass
14	13	com.example.	TC2		Edit title	MenuItem_S_Click				Yes	Conditional Pass
15	14	com.example.	TC2		1	EditText_Input	test			Yes	Conditional Pass
16	15	com.example.	TC2			Back				Yes	Conditional Pass
17	16	com.example.	TC2			Text_Exist_Check	Note 1 test			Yes	Conditional Pass
18	17	com.example.	TC3		(?i).*?tes	Text_S_Click				Yes	Conditional Pass
19	18	com.example.	TC3		Delete	MenuItem_S_Click				Yes	Conditional Pass
20	19	com.example.	TC3			Text_NotExist_Check	Note 1 test			Yes	Conditional Pass
21	20	com.example.	TC3		Note 2	Text_L_Click				Yes	Conditional Pass
22	21	com.example.	TC3		Delete	Text_S_Click				Yes	Conditional Pass
23	22	com.example.	TC3			Text_NotExist_Check	Note 2			Yes	Conditional Pass

Figure 7. Specific case design

The details of the definition of each field in the business test data file are shown as follow.

a) ID: Each test step has a unique ID number. If the test cases failed to perform, according to the ID number to trace the error steps.

b) ActitvityName: To save the name of the active Activity in the top of current task stack of being tested program.

c) Testcase: Used to record the ID of the test cases.

d) ViewID: To save control's ID number. Can search control name by using Hierarchyviewer tool in Android SDK .

e) ViewText: To save the displayed text in the control.

f) InputAction: Input operation of test steps. Such as: InputAction for Button_S_Click, is to click on the Button; If no input, this is null.

g) InputValue: To save the param of the input operation. If no input, this is null.

h) OutputAction: The expected output operation. OutputAction for JUMP, the expected output of this test step is Activity JUMP. If no output operation, this is null.

i) OutputValue: To save the value of the output operation. If no, this is null.

j) Execute: The execute mark of the step. The value is Yes or No. Yes means execute this step, No means pass this step.

k) ActualResult: The result. Pass means pass. It is initialed Null.

D. Keywords

1) test Cases: A Test Case is a sequence of steps that tests the behavior of a given functionality/feature in an application. Unlike traditional test approaches, test language uses pre-defined keywords to describe the steps and expected results (see example in Figure 7 above). Keywords are the basic functional sub-procedures for the test cases of the application under test. A test case is comprised of at least one keyword.

2) Keywords in Different Levels: One of the big decisions to make when designing a keyword-driven framework is the level of the keywords to be used. The low level (e.g. Input, Push) makes them suitable for detailed testing on the interface level. When testing higher level functionality like business logic, low level keywords tend to make test cases very long and higher level keywords (e.g. Add in calculator example) are much more usable. Sometimes it is possible to use only either low or high level keywords and build keyword handlers accordingly. Often both levels are needed and then it is a good idea to construct higher level keywords from low level keywords. For example low level keywords Input and Push could be used to create higher level keywords Add, Subtract, Multiply and Divide and similarly Equals could be created using Push and Check. A straightforward way to construct new high level keywords is letting framework developers implement handlers for them in the framework so that the new handlers use lower level handlers, as it is shown in Figure 8.

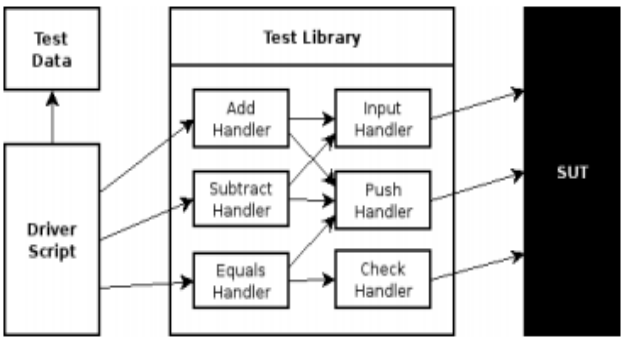


Figure 8. Framework with high level keyword [13]

3) Types of Keywords:

a) Item Operation (Item): Item Operation is divided into operation keyword and results check keyword. Operation keyword is an action that performs a specific operation on a given GUI component. For example button click, input value "CHANDER" in "user name" component etc. Results check keyword is used to judge whether the program implemented correctly and output expected results. For example verify that the value "3" appears in "result" field or judge whether the page jump is correct or not. When performing an operation on a GUI item, parameters should be specified: Name of GUI item, what operation to perform and the values.

b) Utility Functions (Function): a script that executes a certain functional operation that is hard \ non-effective to implement as a Sequence. For example: wait X seconds, take a screen shot etc[14].

c) Sequence: a set of keywords that produces a business process, such as "create customer". We recommend collecting frequently used functional processes such as login, addition of new records to the system as a sequence instead of implementing them as items in test cases[14].

4) Building the Keywords Library: The keyword-driven testing methodology divides test creation into two stages:

a) Planning Stage: Analyzing the application and determining which objects and operations are used by the set of business processes that need to be tested.Determining which operations require customized keywords to provide additional functionality, to achieve business-level clarity, and/or to maximize efficiency and maintainability.

b) Implementation Stage: Building a collection of references that uniquely identify objects, sometimes known as an "object repository", and ensuring that all such references have clear names that follow any predetermined naming conventions. (This is primarily for test automation.). Developing and documenting business-level keywords in function libraries. Creating function libraries involves developing customized functions for the application that needs to be tested.

Although this methodology requires more planning and a longer initial time-investment than going directly to the test

creation stage and recording your steps, it makes the test creation and test maintenance stages more efficient and keeps the structure of individual tests more readable and easier to modify.

V. AKDT FUNCTIONAL AUTHENTICATION

This section takes a function test by using an instance Notepad of the Android Developer's Guide to introduce in brief to the validity of the actual test. First, analyze the test target and create three test cases. They are adding, modifying and deleting Notes, as shown in Figure 7 and the related keywords instructions is shown as Table 1. Some screenshots of the AKDT Automated test process are shown in Figure 9.

Through the testing we found that if use other test approach, you need to write three test script including each step in the test cases which contains a total of 22 instruction. While use AKDT, testers who have no knowledge about test script language only need to write testing use cases in spreadsheets by using total 10 keywords call. If the keyword library already exists for that keyword, keywords call is almost negligible. So, using AKDT can reduce the large amount of work. With the expansion of the scale of the test software, the complexity of the script itself will increase and the role played by keyword becomes increasingly obvious.

Test Case	Test steps
Add test case (TC1)	MenuItem_S_Click: click the menu item " Add Note" CurrentActivity_check:check whether current Activity is " NoteEditor" EditText_Input:input a value "Note1" to a EditText Which ID=0x7f060000 Back: Equals to click a return key on a phone MenuItem_S_Click: click the menu item "Add Note" EditText_Input:input a value "Note2" to the first EditText BackTo_Activity:go back to the Activity of NoteList Screen_TakeShot: take a shot of screen and save to SD card Text_Exist_Check:check whether there is "Note1" text Text_Exist_Check: check whether there is "Note2" text
Modify test case (TC2)	List_N_Click: click No. N line of a ListView Activity_SetLandscape:set the Acitity horizontal MenuItem_S_Click: click the menu item " Edit tiltle" EditText_Input: inpu a value " test" to the first EditText Back: Equals to click a return key on a phone Text_Exist_Check :check whether there is "Note1 test" text
Delete test case (TC3)	Text_S_Click: short click on a text including "test" MenuItem_S_Click: click the menu item " Delete" Text_NotExist_Check: make sure there no exit a text "Note1 test" Text_L_Click: long click on a text "Note2" Text_S_Click: click the the item name "Delete" Text_NotExist_Check: make sure there no exit a text "Note2"

Table 1 *Test case explanation*

VI. CONCLUSIONS

Practice has fully shown that it is very difficult to complete GUI testing under android by relying on traditional automation tools. On one hand, there are many difficulties in capturing and maintaining the scripts. On another hand, the test scripts generated by recording have a very poor reusability due to the close coupling between test scripts and test data. The AKDT uses a different approach to detach the changed and unchanged elements. It makes a clearer separation of testing activities, and avoids the impact between them so that testers without knowledge about test script language can design and create test cases easily. It is important to understand that keywords are not magic, but they can serve well.

It is essential to do test design in a right and efficient way. The process of the test automation should be done but it should not dominate the process. Automation testing should flow with the overall strategy, methodology, and architecture.

Of course, there is still much work that needs to be explored and improved. For example, how to expand the keyword library conveniently and how to customize users' own keywords by using the existing keywords. Moreover, the existing tools available for this approach make use of the Xml, spreadsheets to maintain test cases in object repository which are not very scalable. And the test results should be visually displayed by using Ant and Hudson or other tools.

REFERENCES

[1] I.Burnstein,Practical Software Testing,Springer,2003

[2] Ron Patton, Software Testing, Person Education,2006

[3] Juha Rantanen,"Acceptance Test-Driven Development with Keyword-Driven Test Automation Framework in an Agile Software Project"Helsinki University of Technology, Department of Computer Science and Engineering, Software Business and Engineering Institute. 2007, pp.1-102

[4] Rashmi, Neha Bajpai, "A Keyword-Driven Framework for Testing Web Applications", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No. 3, 2012

[5] JIEHui , LANYu－qing , LUO Pei, "Keyword driven automated testing framework", Application Research of Computers, V01.26 No. 3 Mar.2009

[6] Robotium, http://code.google.com/p/robotium/, 2011-5-28

[7] Bharath Anand R., Harish Krishnankutty, kaushik Ramakrishnan, Venkatesh V.C.," Business Rules- Based Test Automation- A novel Approach for accelerated testing". 2007, pp. 1-12

[8] Faught, Danny R. (November 2004). "Keyword-Driven Testing", Sticky Minds, Software Quality Engineering, Retrieved September 12, 2012.

[9] The net of "test @ your choice"," Keyword-Driven Automation Test Framework",http://www.cromacampus.com/Innoavtion/TAFrameWork/KeyDriven.html

[10] Android SDK API, Android Testing and Instrumentation, 2012-04-02 http://developer.android.com/guide/topics/testing/testing android.html

[11] Reto Meier , Professional Android 2 Application Development, John Wiley &Sons, 2010

[12] Java Excel API, http://www.andykhan.com/jexcelapi/, 2009-10-24

[13] Pekka Laukkanen, "Data-Driven and Keyword-Driven Test Automation Frameworks", Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology. Espoo, February 24, 2006

[14] METHODS & TOOLS, Practical Knowledge for the software developer, tester and project manager Summer 2010(Volume 19-number1) www.methodsandtools.com

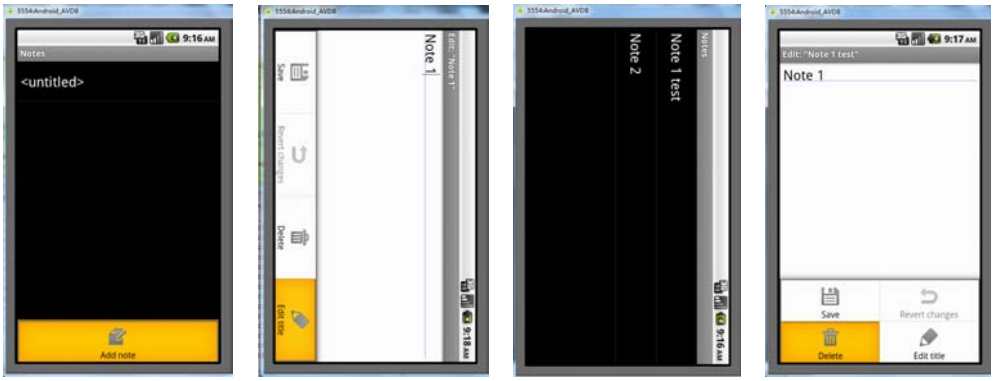


Figure 9. Automated test process screenshot