Executable Test Generation from UML Activity Diagram Using Genetic Algorithm

Anbunathan R
Test Manager and Research Scholar
Bharathiar University
Coimbatore, India
anbunathan.r@gmail.com

AnirbanBasu
Professor, Department of CSE
APS College of Engineering,
Bangalore, India
abasu@anirbanbasu.in

***Abstract*—UML Activity Diagram (AD) is used by testers to generate test cases. These test cases are generated with path coverage criteria, but they are abstract in nature. Converting these test cases to concrete is challenging as it involves identifyingexact Application Programming Interface (API)of the target language from AD elements. We propose a novel method to generate effective and executable test cases from AD automatically. AD elements such as edges and states are mapped with APIs of target language using Genetic Algorithm. The parameters for these APIs are identified by searching through the menu tree database of Application Under Test (AUT). Experiments are conducted to evaluate our approach with differentAndroid applications. It is observed that 83% of code coverage is achieved while running generated tests. Mutation analysis is performed and found that 86 out of 97 mutants are killed by generated tests. Our method is compared with other methods available in literature and found more effective**.

Keywords—Test automation,Genetic Algorithm, Executable test generation,Pairwise testing.

I. Introduction

System testing involves test case documentation and test script coding activities. Test script coding is laborious and costly activity. Whenever User Interface (UI) is getting changed, all scripts need to be revised.It is not only a time-consuming process but also maintenance of scripts is very tedious.

Generating code from UML Activity Diagram (AD) is a good proportion, as UI changes can be incorporated easily.In the case of Android mobile testing, the generated code comprises of Application Programming Interfaces (APIs), which consists of events and parameters.These APIs are provided by the target scripting language such as Robotium [21] etc. In [2], these events and parameters are mapped with AD elements through XML file manually.In our approach, these events are mapped using Genetic Algorithm (GA) automatically.The objective function of GA is based on string comparison algorithms such as Longest Common Subsequence (LCS) and Levenshtein distance algorithms.This allows AD to have more user-friendly names for AD edges and nodes.Also, parameter identification is a challenging task. A menutree database is extracted from target device [6]. This database is used to distinguish menu items from other parameters such as index and numerical values.

In this paper novel techniques have been proposed for event mapping technique based on GA and parameter identification from Menu Tree. The paper is organized as follows: related work from literature is summarized in Section 2, Section 3 describes the proposed approach. Section 4 discusses various experiments conducted on this approach, while Section 5 compares the proposed method with other methods available in the literature.

II. Related Work

Test automation [1] is most prevalent and versatile in literature. Nguyen et al. [2] proposed a method to generate Robotium scripts from State Diagram(SD). Edges represent Android events. These events are mapped with RobotiumAPIs through an XML file. Domain inputs such as menu names and indexes are updated in this XML file. A tool called Magic parses SD, takes XML based mapping file as input and generates Robotium scripts. In our approach, the XML-based mapping file is generated automatically using GA. Menu names are extracted from the menu tree database.

Mahmood R et al. [7][17] proposed an approach to generate Robotium scripts from the source code of Application Under Test (AUT). A tool namedEvodroid analyses the program and identifies code segments. An evolutionary algorithm searches through these segments and generates test cases in order to maximize code coverage. In our approach, GAevaluates user inputs and identifies exactRobotium API by leveraging LCS and Levenshtein algorithms.

Alsmadi I et al. [25] proposed a method to generate test cases using GA. Two different GAs are compared with respect to their resulting test coverage. GA which considers test case as a gene is found the better solution. Test cases are optimized by setting test coverage as exit criteria. In our approach, test cases are generated by recursion algorithm, but test scripts are generated using GA.A chromosome is an arbitrary string whose fitness value increases proportionally when its matching percentage with the target API name increases.

Shirole M et al. [26] proposed a method to generate test cases from SD. GA is used to identify the feasible path and test data. Transition with start node, end node, and a guard is encoded as a gene.A chromosome is considered as a test case which consists of many feasible transitions.

Shah etal. [27] proposed a method to generate test cases from Class and Sequence diagrams. These diagrams are exported into the corresponding XML files. A tool is developed to parse these XML files and test cases are generated automatically. In our approach, XMI file from AD is parsed by a tool and both test cases and test scripts are generated automatically.

In literature, several methods [3][4][5][14][15] are available to generate executable Java code from UML diagrams such as Sequence, Activity, Class and State diagrams.

Several methods [10][11][13][28][29] provide different approaches to generate test cases and test data from UML diagrams.

III. Proposed Method

In this section, some important definitions, overview, and architecture of the proposed test automation framework are discussed.

A. *Definitions*

Definition 1 (Basis path): Basis path (sometimes called independent path) through the program is any path from starting node to terminal node that introduces at least one new set of processing statements or a new condition.

Definition 2 (Path coverage): Path coverage is a white-box testing concept that considers the possible paths of the software under test.

Definition 3 (Code coverage): Code coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs.

Definition 4 (Mutation analysis): Mutation analysis is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in a small way. Each mutated version is called a mutant and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. This is called killing the mutant. Test suites are measured by the percentage of mutants that they kill.

Definition 5 (Defect Removal Efficiency): The defect removal efficiency (DRE) gives a measure of the software testing ability to find defects prior to release. It is calculated as a ratio of defects found to total number of defects found and defects leaked to customer.

B. *Overview of the proposed framework*

The proposed framework is based on UML Activity Diagram (AD) based testing. The major steps involved in this approach are illustrated in Figure 1. In this approach, AD is created to capture input scenarios. XMI file obtained from this AD is parsed to extract model information such as edges, their labels, and nodes. A Control Flow Graph (CFG) is derived from edges by sorting the edges using Breadth First Search (BFS) algorithm. A recursive algorithm is developed to obtain path test cases from CFG. These path test cases are extended so that the path traces till activity final node. To generate test scripts, Robotium APIs are identified from Edges and Robotium API database using GA. The parameters for these APIs are identified from Edges and menu tree database. Along with normal flow test cases, alternative flow test cases are generated using the combinatorial technique. Finally, test scripts are generated using String template.

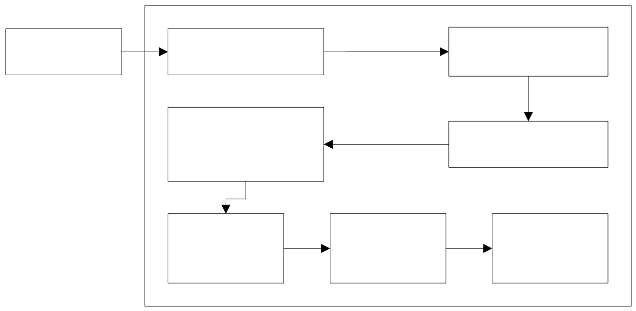
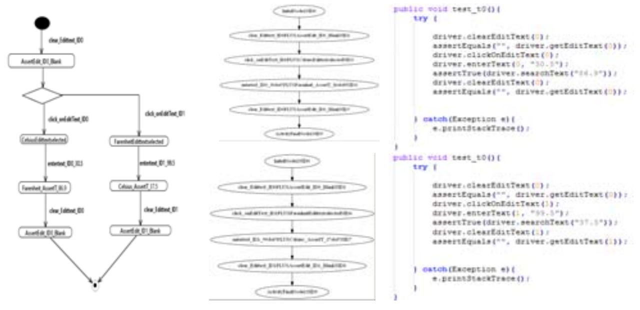


Figure 1. Block diagram of proposed automation framework.

1) *XMI parser*

A UML Activity Diagram as shown in Figure 2 is created using Papyrus tool [30]. This Activity diagram is represented in the form of XMI notation and saves as *.uml file. This XMI file is parsed using SAX parser and then different Activity Diagram components such as edges, labels and nodes are extracted.



```
public void test_t0(){
    try {

        driver.clearEditText(0);
        assertEquals("", driver.getEditText(0));
        driver.clickOnEditText(0);
        driver.enterText(0, "30.5");
        assertTrue(driver.searchText("86.9"));
        driver.clearEditText(0);
        assertEquals("", driver.getEditText(0));

    } catch(Exception e){
        e.printStackTrace();
    }
}
public void test_t0(){
    try {

        driver.clearEditText(0);
        assertEquals("", driver.getEditText(0));
        driver.clickOnEditText(1);
        driver.enterText(1, "99.5");
        assertTrue(driver.searchText("37.5"));
        driver.clearEditText(1);
        assertEquals("", driver.getEditText(1));

    } catch(Exception e){
        e.printStackTrace();
    }
}
```

Figure 2. Temperature converter Activity Diagram, corresponding Test Cases and Test Scripts.

2) *Precedence relation*

Activity Diagram is parsed using SAX parser and then all nodes and all edges are identified. The precedence relationship between nodes is found by sorting nodes from start node to end node using BFS algorithm as shown in Algorithm 1.

Algorithm 1. PrecedenceSortingBFS

This algorithm adds adjacent nodes in the queue, and then de-queues each node, marks as visited, repeatedly checks other adjacent nodes. A waiting queue is required to keep precedence relationship, when different paths are meeting in a common node, for example, Join node. After sorting all nodes with respect to precedence relationship, a Control Flow Graph (CFG) is generated.

Activity Diagram

est cases

3) *Basis Path test case*

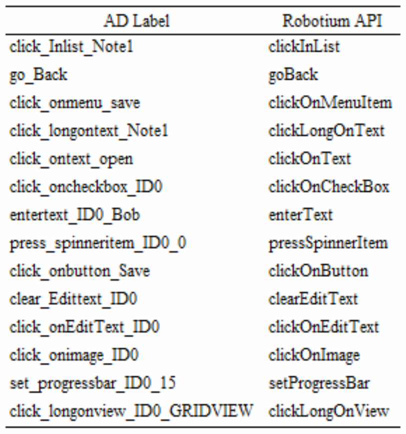
Basis pathsare obtained by traversing CFG from start node to end using DFS algorithm as shown in Algorithm 2. This algorithm is recursive, it marks all nodes traversed as visited. When an edge with the visited node is encountered, a new recursion is started to get new Basis path. The algorithm terminates when all edges are traversed. Basis paths ensure that all loops are traversed only one time, and all nodes and transitions are covered.

Algorithm 2. FindBasisPaths

Two Basis Path test cases are generated for temperature converter as shown in Figure 2.

4) *Identify Robotium APIs*

Robotium APIs are identified from edge labels. For example, from the label *click_onEditText_ID0,* Robotium API *driver.clcikOnEditText(0)*is identified. For this purpose, Genetic Algorithm is used to intuitively compare strings and identify matching strings. GA employs string matching algorithms such as Longest Common Substring (LCS) algorithm, Longest Common Subsequence and Levenshtein distance algorithms to identify Robotium API. Table 1 shows AD labels and their corresponding Robotium APIs, which are identified using GA.



AD Label	Robotium API
click_Inlist_Note1	clickInList
go_Back	goBack
click_onmenu_save	clickOnMenuItem
click_longontext_Note1	clickLongOnText
click_ontext_open	clickOnText
click_oncheckbox_ID0	clickOnCheckBox
entertext_ID0_Bob	enterText
press_spinneritem_ID0_0	pressSpinnerItem
click_onbutton_Save	clickOnButton
clear_Edittext_ID0	clearEditText
click_onEditText_ID0	clickOnEditText
click_onimage_ID0	clickOnImage
set_progressbar_ID0_15	setProgressBar
click_longonview_ID0_GRIDVIEW	clickLongOnView

TABLE I. AD LABELS AND THEIR CORRESPONDING ROBOTIUM APIS

AD label consists of two parts API name and parameter. API name is user-friendly, readable string connected with an underscore, whereas parameter often consists of keyword ‘ID’ and a number indicating index value. For example, in the case of *click_oncheckbox_ID0,* “click_oncheckbox” represents API name and ID0 represents index value 0 to be passed as an argument to the API. In the case of *set_progressbar_ID0_15*, a second parameter for setting progress bar value is given as '15'. In the case of *click_onbutton_Save*, the button name to be clicked is given as "Save". In the case of *go_back*, there is no argument involved.

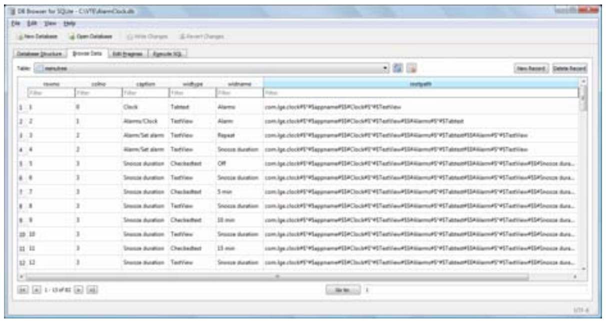
5) Identify parameter from Menu tree database

The parameter in AD label represents menu item to be handled by the phone. For example, in the case of click_onmenu_save, the menu item to be clicked is given as "save". But actually, in the phone, it is displayed as "Save". In this case capital letter "S" is missing in the label. To handle this situation, the parameter is compared with strings present in Menu tree database and the correct string is obtained.

Menu tree algorithm identifies layouts such as linear layout, relative layout, frame layout extra. From these layouts, it extracts UI objects such as text, buttons. When it clicks one text item, it checks whether new page or popup is opened. If it is a new page, it recursively calls itself to do the learning. If the UI object is a popup or a button, it is handled in a different manner. In this way, all UI objects are learned and then each item is clicked based on the type of widget. The type of widget can be text, radio button, button with text, button with description, system event such as back key press, home key press. The algorithm for generating menu tree is as shown in Algorithm 3.

Algorithm 3. Menu Tree Generation

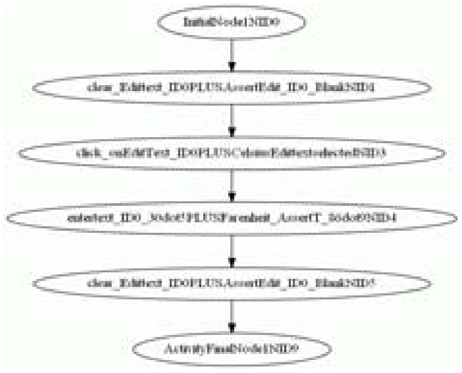
Menu tree algorithm stores all UI menu items along with its path from root and type of widget in a SQL ite database, as shown in Figure 3.



	rowno	colno	caption	widtype	widname	rootpath
1	1	0	Clock	Tabtext	Alarms	[illegible]
2	2	1	Alarms/Clock	TextView	Alarm	[illegible]
3	3	2	Alarm/Set alarm	TextView	Repeat	[illegible]
4	4	2	Alarm/Set alarm	TextView	Snooze duration	[illegible]
5	5	3	Snooze duration	Checkedtext	Off	[illegible]
6	6	3	Snooze duration	TextView	Snooze duration	[illegible]
7	7	3	Snooze duration	Checkedtext	5 min	[illegible]
8	8	3	Snooze duration	TextView	Snooze duration	[illegible]
9	9	3	Snooze duration	Checkedtext	10 min	[illegible]
10	10	3	Snooze duration	TextView	Snooze duration	[illegible]
11	11	3	Snooze duration	Checkedtext	15 min	[illegible]
12	12	3	Snooze duration	TextView	Snooze duration	[illegible]

Figure 3. Menu tree database

The type of menu item is stored under column 'widtype'. This can be TextView, CheckedText, Button, CheckBox etc. The menu item is stored under 'widname' column. The path traced by the Menu tree algorithm is stored under 'rootpath' column for each menu item. The menu item extracted from AD label is compared with each string under 'widname' column and then closest match is retrieved. In this way, even though user provides inaccurate menu value in AD label, accurate menu value is obtained using menu tree database.


InitialNode1NID0
clear_Edittext_ID0PLUSAssertEdit_ID0_BlankNID1
click_onEditText_ID0PLUSColumnEdittextselectedNID3
entertext_ID0_30dot5PLUSFarenheit_AssertT_86dot9NID4
clear_Edittext_ID0PLUSAssertEdit_ID0_BlankNID5
ActivityFinalNode1NID9

Input1	Input2	Input3	Input4
0	0	30.5	0
1	1	#*%$%	1

TEST CASES	Input1	Input2	Input3	Input4
1	0	0	30.5	0
2	0	1	#*%$%	1
3	1	0	#*%$%	0
4	1	1	30.5	1
5	~0	0	~30.5	1
6	~0	1	~#*%$%	0

Figure 4. Basis Path test case, Pair wise Input and corresponding Output.

6) Pairwise test cases

Apart from Basis path test cases, Pairwise test cases are generated to cover negative scenarios and alternative flow. Considering the first Basis Path test case in Figure 2, the domain values present in the AD labels are taken as inputs for generating combinatorial test cases. ID0, ID0, 30.5 and ID0 are extracted from AD labels *clear_Edittext_ID0*, *click_onEditText_ID0*, *entertext_ID0_30.5* and *clear_Edittext_ID0* respectively. From these domain inputs, combinatorial outputs are generated as shown in Figure 4.

In Pairwise input table, domain inputs are listed in the first row and their corresponding alternative flow inputs are listed in the second row. From ID0, 0 is extracted as domain input and 1 is selected as alternative flow input. Other than integer values, the other type of domain inputs is treated as string input. For 30.5, the alternative string is selected as “#%$%”. The Pairwise output table shows that six combinatorial test cases are generated from one BP test case.

7) Generate Java Scripts

String Template [31] is used to generate Java source code after identifying Robotium APIs from normal and alternative flow test cases. These Java files are executed from Eclipse environment with Android JUnit plug-in on the target which is either Emulator or Android mobile device. The result of a test case is displayed as Pass if all APIs are successfully executed.

C. Architecture of the proposed framework

UML Activity Diagram (AD) is parsed and Basis Path (BP) test cases are generated as shown in Figure 5. These BP test cases are extended to reach Activity final node. These test cases are called as Extended Basis Path (EBP) test cases. EBP ensures completeness of test scenario. From these EBP test cases, executable scripts are generated by identifying events to be triggered, test asserts and test data. Alternative flow test cases are identified by combining test data using the Pairwise technique.

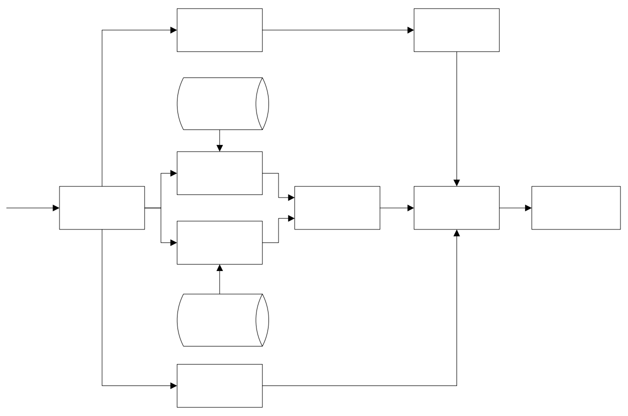


Figure 5. The architecture of the proposed approach for generating executable test scripts.

1) Identify Event

The event is identified by parsing transition in an AD. A transition consists of multiple words separated by an underscore (_). The last word is considered as the parameter. The rest of the words constitute the String Under Evaluation (Se). A set of string matching algorithms such as Longest Common Substring, Longest Common Subsequence and Levenshtein Distance is used to process Se and identify candidate APIs from API database of the target language. In our approach, Robotium is the target language. Genetic Algorithm is used to selecttarget API from the candidate APIs.

Genetic Algorithm

GA is used to select target API from a set of matching candidate APIs.

Chromosome

A chromosome is represented by a character sequence. A gene is represented by a single character from the set [A-Za-z].

Selection

The roulette wheel selection is used to select chromosomes.

Crossover

Multiple crossover is used

Fitness function

Fitness is calculated as follows:

(i). Evaluate_LCSubstring

totalFitness += (matching substring length of Se and Chromosome * 500);
totalFitness = totalFitness/Levenshteindistance(Se,Chromosome);
totalFitness += (matching substring length of Se and bastcandidate * 1000);
totalFitness = totalFitness/Levenshteindistance(Se,bastcandidate);

(ii). Evaluate_LCSubsequence

totalFitness += (matching subsequence length of Se and Chromosome*100);
totalFitness = totalFitness/Levenshteindistance(Se,Chromosome);
totalFitness += (matching subsequence length of Se and bastcandidate * 200);
totalFitness = totalFitness/Levenshteindistance(Se,bastcandidate);

(iii). Evaluate final fitness

final fitness= evaluate_LCSubstring + evaluate_LCSubsequence

Mutation

Mutation value is selected as 0.01. Smart mutation is used to replace the genes of selected chromosome with the genes of the best candidate. This helps to arrive at the solution faster.

For example, *click_ontext_open* is the transition having ‘open’ as the parameter. GA takes ‘click_ontext’ as input and generates ‘clickOnText’ as output, which is exactly matching API from API database.

TABLE II. ROBOTIUM API DATABASE

Robotium API	Type	Number of Arguments	1st Argument Type	2nd Argument Type
clickOnMenuItem	Event	1	Text	
clickOnButton	Event	1	Text	
goBack	Event	0	NA	
clickInList	Domain	1	Index	
enterText	Domain	2	Index	Text
clickLongOnText	Event	1	Text	
clickOnText	Event	1	Text	
assertCurrentActivity	Assert	1	Text	
assertMemoryNotLow	Assert	1	Text	
clearEditText	Domain	1	Index	
clearLog	Event	1	Text	
clickLongInList	Domain	1	Index	
clickLongOnScreen	Event	1	Text	
clickLongOnTextAndPress	Domain	2	Text	Index
clickOnCheckBox	Domain	1	Index	
clickOnImage	Domain	1	Index	

2) *Identify Parameter*

The last word of a transition is expected to represent the parameter of the event to be triggered. However, this string is searched in menu tree database of AUT by ignoring blank space and capital/small. If the match is found, matching string from menu tree is selected as a valid parameter. For example, *click_onmenu_EditTitle* is the transition having 'EditTitle' as the parameter. From the menu tree database,the corresponding menu item is found as 'Edit Title'.

3) *Identify Domain*

Table 2 shows Robotium API database, which consists of Robotium API, its type, the number of arguments and type of each argument. The domain value is identified by checking the type of target API. If the type of target API is 'Domain', then the last word of transition is considered as domain input. ID is used as a keyword in a transition to feeding index value. For example, *enterText_ID0_Hello* is the transition having domain input as 'Hello' and index value as '0'.

4) *Identify Assert*

Assert is identified by searching keywords 'AssertT' and 'AssertF' in the node name of AD. If 'AssertT' is present in the node name, then the last word in the node name is evaluated to be true. If 'AssertF' is present in the node name, then the last word of the node name is evaluated to be false. For example, if*Selected_AssertT_Note* is the node name, then the string 'Note' is expected to present in the current screen. If*Deleted_AssertF_Note* is the node name, thenthe string 'Note' is expected not to present in the current screen.

IV. Empirical Evaluation

In this section, experiment environment and experimental setup are explained. The performance of GA is analyzed and Efficiency of menu tree algorithm is studied. Code coverage is studied to improve test coverage. Mutation analysis is performed to find out defect removal efficiency of generated test cases.

A. *Experiment Environment*

We have developed a tool called Virtual Test Engineer (VTE) to parse AD to generate test cases and test scripts. A Control Flow Graph (CFG) is derived from AD and BP test cases are generated from this CFG. Graphviz tool [23] is used to display test cases. ALLPAIRS tool [22] is used to generate alternative flow (Pairwise) test cases. Robotium [21] scripts are generated using GA [24]. EMMA [18] is used for instrumenting AUT to find code coverage while running Robotium scripts. JACOCO [19] is used for Calculator application.

B. *ExperimentalSetup*

Android applications are downloaded from F-Droid [20] repository and used for our experiment. Basic metrics such as Number of Nodes, Number of Edges, Cyclomatic complexity, Number of EBP test cases, Number of Pairwise test cases and Total test cases are captured as given in Table 3. The effectiveness of our method is evaluated in terms of Performance, Efficiency, Codecoverage and Defect Removal Efficiency (DRE).

C. *Performance*

Research Question 1 (Performance): What is the time taken by GA to identify exact API from Robotium API database?

GA takes the transitions extracted from AD as inputs. Transition name consists ofeventname and its argument.

TABLE III. VALUES OF BASIC METRICS

Subjects	Number of Nodes	Number of Edges	Cyclomatic complexity	Number of Extended basis path test cases	Number of Pairwise testcases	Total testcases
Notepad	24	37	15	47	12	59
Temperature converter	10	10	2	2	6	8
Calculator	68	80	14	15	0	15
Contact manager	27	34	9	49	335	384
MunchLife	22	23	3	4	12	16
TippyTipperActivities	43	45	4	6	42	48
GameOfLife	12	12	2	2	0	2
WorldClock	22	22	2	2	18	20
StopWatch	11	11	2	2	0	2
AndroidTimer	11	11	2	2	0	2

Using this event name, Robotium API has to be identified. GA uses LCS and Levenshtein algorithms to identify exact Robotium API. Event string is matched with each API from Robotium API database and corresponding grade value is calculated. Top graded strings are shortlisted and identified as potential candidates for target API. GA generates a population of chromosomes and evaluates the fitness of each chromosome. A chromosome with highest fitness value survives and the weak one dies over subsequent generations. *Number of generations* and *time taken to* *identify*exact Robotium API are plotted as shown in Figure 6.*Number of generations* ranges from 6 to 35. Time taken by GA ranges from 1 second to 9 seconds. We observed that *number of generations* and *time taken by GA* are having lower values if the transition name is very close to target API. These parameters are having higher values if transition name is not closely matching with target API.

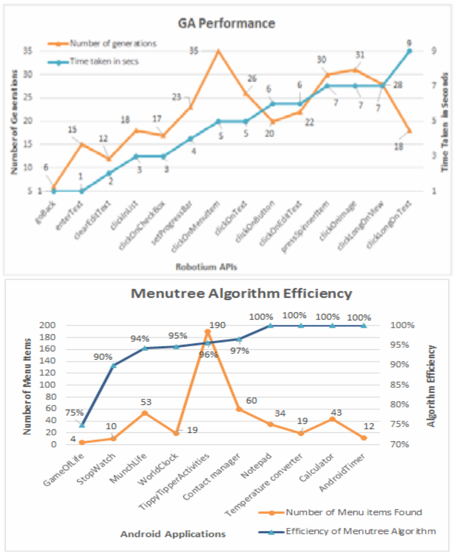


Figure 6. GA performance and Menu tree algorithm efficiency.

D. *Efficiency*

Research Question 2 (Efficiency): What is the efficiency of menu tree algorithm in extracting the menu items from Android mobile device?

Menu tree algorithm [6] recursively navigates through different screens and extracts menu items from Android mobile device. It handles different widgets such as menu items, buttons, popups, hot menus etc. The extracted widgets are stored in SQLite database. The transition name consists of both Robotium API and its argument. The argument is either menu item or value. To distinguish menu item from value, the argument is searched through menu tree database. If found, it is considered as a menu item, else it is considered as value.The efficiency of menutree algorithm is calculated as the ratio of menu item extracted and total menu items present in the Android phone.*Algorithm efficiency* is plotted against *number of menu items* present in AUTas shown in Figure 6. The efficiency ranges from 75% to 100% for Android applications under evaluation.

E. *Code coverage*

Research Question 3 (Coverage): What is the code coverage achieved using our approach?

EMMA is used to instrument the application source code. When test script is executed, code coverage is obtained as coverage report. Coverage is improved by adding more scenarios in AD. We observed that the code coverage is impacted by Java catch blocks, which are not reachable during normal operation. Robotium has its own limitation. It handles only AUT but fails when context switch happensfrom one application to another application. Because of these factors, 100% code coverage is not achievable.Lines of code (LOC) is plotted against percentage code coverage as shown in Figure 7. LOC ranges from 32 to 1156 for applications under evaluation. Code coverage ranges from 67% to 100%.

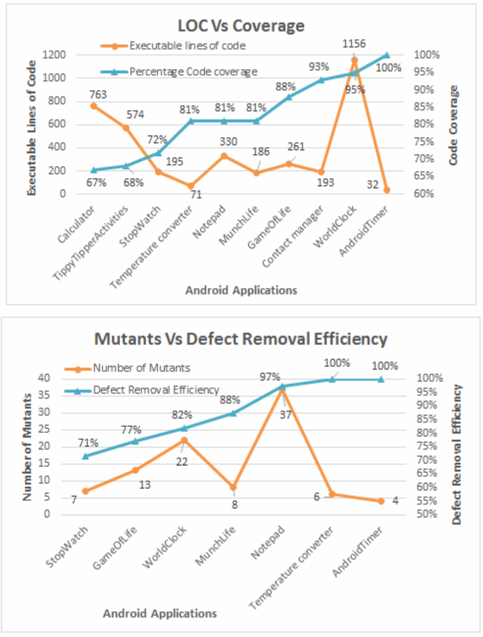


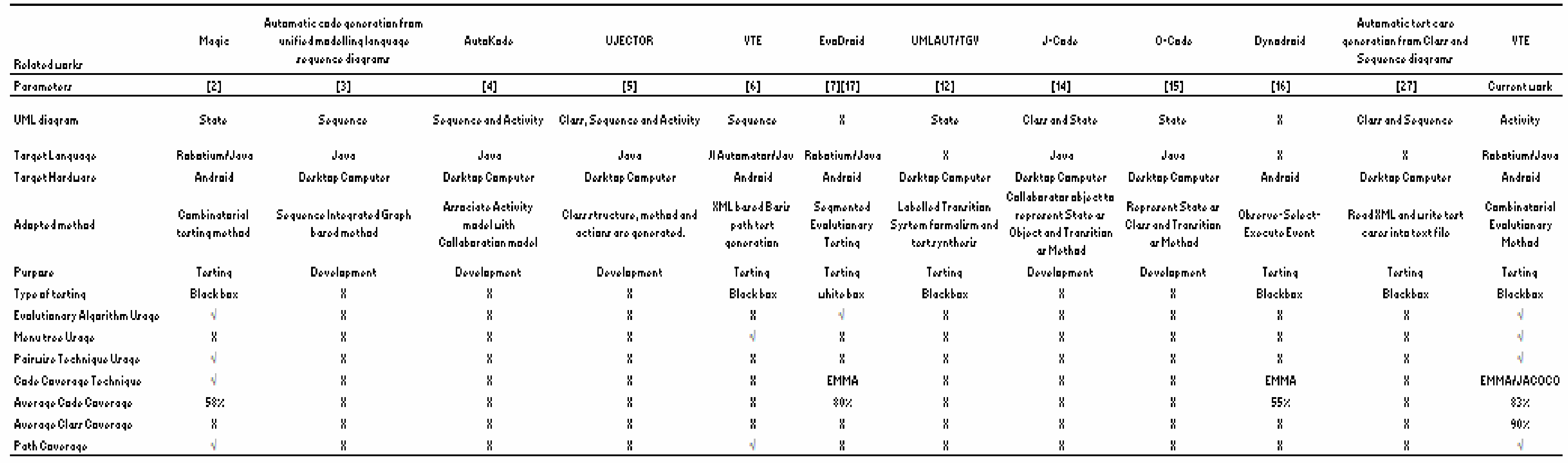
Figure 7. Code coverage and Mutation analysis for Android applications

F. *Mutation Analysis*

Research Question 4 (DRE): What is the defect removal efficiency of generated scripts?

Jester [8] is used to generate mutants automatically. Android mutation operators [9] are used except Android application life cycle and Android manifest related operators. These two operators are not used because of limitation of Jester. Out of 97 mutants, 86 mutants are killed.*Number of mutants* is plotted against *DRE* as shown in Figure 7. *Number of mutants* ranges from 4 to 37. *DRE* ranges from 71% to 100%. We observed that mutant operators related to Intents are not effective because of limitation of Robotium as mentioned in section 5.5

TABLE IV. COMPARISON WITH OTHER METHODS



Related works	Magic	Automatic code generation from unified modelling language sequence diagrams	AutoKode	UJECTOR	VTE	EvoDroid	UMLAUT/TGV	J-Code	O-Code	Dynodroid	Automatic test case generation from Class and Sequence diagrams	VTE
Parameters	[2]	[3]	[4]	[5]	[6]	[7][17]	[12]	[14]	[15]	[16]	[27]	Current work
UML diagram	State	Sequence	Sequence and Activity	Class, Sequence and Activity	Sequence	X	State	Class and State	State	X	Class and Sequence	Activity
Target Language	Robotium/Java	Java	Java	Java	JI Automator/Jav	Robotium/Java	X	Java	Java	X	X	Robotium/Java
Target Hardware	Android	Desktop Computer	Desktop Computer	Desktop Computer	Android	Android	Desktop Computer	Desktop Computer	Desktop Computer	Android	Desktop Computer	Android
Adapted method	Combinatorial testing method	Sequence Integrated Graph based method	Associate Activity model with Collaboration model	Class structure, method and actions are generated.	XML based Basis path test generation	Segmented Evolutionary Testing	Labelled Transition System formalism and test synthesis	Collaborator object to represent State or Object and Transition or Method	Represent State or Class and Transition or Method	Observe-Select-Execute Event	Read XML and write test cases into text file	Combinatorial Evolutionary Method
Purpose	Testing	Development	Development	Development	Testing	Testing	Testing	Development	Development	Testing	Testing	Testing
Type of testing	Black box	X	X	X	Black box	white box	Blackbox	X	X	Blackbox	Blackbox	Blackbox
Evolutionary Algorithm Usage	√	X	X	X	X	√	X	X	X	X	X	√
Menu tree Usage	X	X	X	X	√	X	X	X	X	X	X	√
Pairwise Technique Usage	√	X	X	X	X	X	X	X	X	X	X	√
Code Coverage Technique	√	X	X	X	X	EMMA	X	X	X	EMMA	X	EMMA/JACOCO
Average Code Coverage	58%	X	X	X	X	80%	X	X	X	55%	X	83%
Average Class Coverage	X	X	X	X	X	X	X	X	X	X	X	90%
Path Coverage	√	X	X	X	√	X	X	X	X	X	X	√

V. Comparison with other Methods

In [2], the transition name is mapped with Robotium APIs through XML file manually. In our approach, the mapping is done automatically by GA. In both approaches,BP and Pairwise test cases are generated.

In [7][17], GA is used to generate Robotium test cases from source code, by analyzing the program and then extracting call graph and interface information. The goal is to achieve maximum code coverage. Test casesto cover both normal flow and alternative flow are not in the scope. In our approach, CFG ensures path coverage and Pairwise technique ensures alternative flow test coverage.

The average code coverage achieved by [2], [7] and our approaches are 58%, 80%, and 83% respectively. The average class coverage achieved by our approach is 90%. In [2] and our approach, both path coverage and code coverage are achieved. In [7], only code coverage is focused. The type of testing adopted by [2], [12], [16] and our approach is black box test approach, whereas, in the case of [7], it is white box approach.

In [27], test cases are generated from Sequence Diagram. But these are abstract test cases and not executable. Coverage criteria such as path coverage and code coverage are not evaluated. Mutation analysis is not considered to calculate DRE.

In [6], executable tests are generated from Sequence Diagram automatically. But UI automator based library functions need to be written manually. Only these library functions need to be used in Sequence Diagram. Path coverage, code coverage, and mutation analysis are not considered.

Table 4showsthe comparison of our current workwith the related research works. Considering the parameters compared across these methods, our current method exhibits more advantage. Our method employs more accurate algorithms in identifying events, and menu items. Our approach ismore useful to the frequently changing UI environment.

VI. Conclusions

We presented an approach to generate Robotium scripts from UML Activity Diagram. GA is used to identify event name and its arguments and convert the event name into respective Robotium API automatically. This is the major contribution of this paper. Another contribution of this paper is that the menu tree database is used to identify right menu item which is used as an argument to Robotium API.

We found that the average time taken by GA to identify an API was 4.7 seconds. We observed that the average efficiency of menu tree algorithm was 95% to extract menu items from the phone. We used 10 different size applications with LOC ranging from 32 to 1156 for our experiment. We observed that AD for each application has 10 to 68 nodes and 10 to 80 edges. Path test cases and alternative flow test cases are generated.The Pairwise testing technique is used to generate alternative flow test cases. We obtained an average of 83% code coverage using this approach. Mutation analysis shows that 86 out of 97 mutants are killed by generated tests.

The comparison of our work with the related works shows that our approach is more efficient in extracting events and menu items from AD automatically. Our approach is more efficient in handling UI changes, as it can extract menu items from Android mobile.

In future, we will investigate automatic generation of test data and oracles using the evolutionary algorithm. We believe that this will help to generate more efficient test cases with the improved defect removal efficiency.

References

[1] AnirbanBasu, Software Quality Assurance, Testing and Metrics, PHI Learning, 2015.

[2] C. D. Nguyen, A. Marchetto, and P. Tonella. "Combining model-based and combinatorial testing for effective test case generation", In Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISSTA 2012.

[3] Debasish Kundu, Debasis Samanta, and Rajib Mall. "Automatic code generation from unified modelling language sequence diagrams", IET Softw., 2013, Vol. 7, Iss. 1, pp. 12–28.

[4] Sunitha Edacheril Viswanathan, and Philip Samuel. "Automatic code generation using unified modeling language activity and sequence models", IET Softw., 2016, Vol. 10, Iss. 6, pp. 164–172.

[5] Muhammad Usman, and Aamer Nadeem, “Automatic Generation of Java Code from UML Diagrams using UJECTOR”, International Journal of Software Engineering and Its Applications, vol. 3, no. 2 2009, pp. 21-38.

[6] Anbunathan R, and Anirban Basu. "Automatic Test Generation from UML Sequence Diagrams for Android Mobiles", International Journal of Applied Engineering Research, Vol. 11, No.7, pp. 4961-4979, 2016.

[7] R. Mahmood, N. Mirzaei, and S. Malek. "EvoDroid: segmented

evolutionary testing of Android apps", In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014).

[8] I. Moore, "Jester a Junit test tester," in proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering, Springer 2001.

[9] L. Deng, N. Mirzaei, P. Ammann and J. Offutt. "Towards Mutation Analysis of Android Apps", In proceedings of the Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE 2015.

[10] Sumit Dahiya, Rajesh K. Bhatia, and Dhavleesh Rattan. "Regression test selection using class, sequence and activity diagrams", IET Softw., 2016, Vol. 10, Iss. 3, pp. 72–80.

[11] Dragan M, “Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments”, IEEE Transactions on software Engineering, vol. 28, no. 4 2002, pp. 413-431.

[12] Simon Pickin, Claude Jard, Thierry J, and Jean-Marc J, “Test Synthesis from UML Models of Distributed Software”, IEEE Transactions on software Engineering, vol. 33, no. 4 2007, pp. 252-268.

[13] Richard A D, and Jefferson Offutt, “Experimental Results from an Automatic Test Case Generator”, ACM Transactions on Software Engineering and Methodology, vol. 2, no. 2 1993, pp. 109-127.

[14] Niaz, I.A.: ‘Automatic code generation from UML class and statechart diagrams’. Thesis Report, University of Tsukuba, Japan, 2005.

[15] J. Ali, and J. Tanaka, “An Object Oriented Approach to Generate Executable Code from OMT-Based Dynamic Model”, Journal of Integrated Design and Process Science, vol. 2, no. 4 1998, pp. 65-77.

[16] A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for android apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pages 224–234, 2013.

[17] R. Mahmood: ‘An Evolutionary Approach for System Testing of Android Applications’. Thesis Report, George Mason University, USA, 2015.

[18] EMMA, http://emma.sourceforge.net/.

[19] JACOCO, http://www.eclemma.org/jacoco/

[20] F-droid, https://f-droid.org/.

[21] Robotium, http://code.google.com/p/robotium/.

[22] ALLPAIRS Test Case Generation Tool. http://www.satisfice.com/tools.shtml.

[23] GraphViz, Graph Visualization Software, http://www.graphviz.org/Documentation/dotguide.pdf

[24] Hybrid Genetic Algorithm, https://github.com/carlosnasillo/Hybrid-Genetic-Algorithm/blob/master/README.markdown

[25] Alsmadi I, Alkhateeb F, Maghayreh E, Samarah S, and Doush I A. “Effective generation of test cases using genetic algorithms and optimization theory”, Journal of Communication and Computer 7, 11 (2010), 72-82.

[26] M. Shirole, A. Suthar, and R. Kumar, "Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm," in Proceedings of the 4th India Software Engineering Conference, pp. 125-134, 2011.

[27] Shah SA, Shahzad RK, Bukhari SS, Humayun M,“Automated Test Case Generation Using UML Class & Sequence Diagram”, British Journal of Applied Science & Technology,15(3): 1-12, 2016.

[28] C. Sun, B. Zhang, J. Li, “TSGen: A UML Activity Diagram-based Test Scenario Generation Tool,” In International Conference on Computational Science and Engineering, IEEE, Vancouver, Canada, 853-858, 2009.

[29] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong and Z. Guoliang,“Generating Test Cases from UML Activity Diagram based on Gray-Box Method", In Proceedings of the 11th Asia-Pacific Software Engineering Conference(APSEC), IEEE, Busan, Korea, 1-8, 2004.

[30] https://eclipse.org/papyrus/.

[31] http://www.stringtemplate.org/.