國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

安卓系統應用程式品質之測試自動化

Automated Testing for Quality of Android Applications

林修博

Hubert Lin

指導教授：王凡 博士

Advisor: Farn Wang, Ph.D.

中華民國 105 年 7 月

July, 2016

誌謝

獻上最誠摯的感謝，給我的雙親，你們的支持與照顧，是我一路上的精神糧食與動力。謝謝親愛的爸爸，總能給予我最真誠的建議，提升我論文的完整度及流暢度；也要感謝我的媽媽，總在我精疲力盡時，用最溫暖的關愛，讓我再次打起精神。

兩年研究所生活中，我首先要感謝指導教授王凡教授，在學習過程當中，從您身上學習到的不僅僅是軟體測試領域的專業，更常常蒙受您的教誨與叮嚀。

另外，要感謝重豪、俊瑋、元鴻、唯霖、梵清、謝橋，每每在實驗室與你們討論，總讓我受益匪淺，也感謝你們在一起修課的日子中，給予的協助及幫忙。

特別感謝汶宏學長，在撰寫論文的過程當中，協助我程式架構上的改進。也要感謝啟允、宗儒及上誼，一同面對各種挑戰，互相扶持與前進。另外，要感謝一起執行許多計畫的光奇，協助我進行許多實驗。

家薇、庭杰、正言、康棟、紹哲、家銘、文舜、冠宇、宥如、汝緯有了你們讓我課後之餘多了許多歡笑，有了各位的陪伴讓我生活變得精采。

最後，謹以此文獻給我摯愛的雙親。

中文摘要

為了提升安卓應用程式品質、減少測試時間、降低人力需求，本論文開發兩套黑箱測試自動化測試工具 Monkey with Vision (MONVIS) 與 Application Quality Testing (AQT)。使用者無需撰寫任何測試腳本，MONVIS 與 AQT 可自動執行應用程式找出品質不良之處。MONVIS 是一套無測試知識自動化測試工具，可以模擬使用行為。因此，MONVIS 用於檢測應用程式正確性包括閃退與例外。MONVIS 利用電腦視覺技術解析應用程式畫面，以提升模擬使用者行為的準確性。AQT 則進一步利用測試知識進而模擬使用者並且收集應用程式反應。AQT 會檢查應用程式是否能正確安裝並且執行、記憶體使用、連線狀況、使用者介面與穩定性。MONVIS 與 AQT 追蹤程式各種品質的不良狀況，並據以提供應用程式相關的執行日誌、畫面、與測試報告給使用者，使用者可根據測試報告快速有效的進行軟體除錯與修正。

關鍵字：安卓應用程式測試、圖型化使用者介面測試、安卓應用程式測品質、測試自動化、黑箱測試

ABSTRACT

We develop two black-box testing tool called Monkey with Vision (MONVIS) and Application Quality Testing (AQT). In order to extend the automation level of testing technique, we develop MONVIS and AQT that allow tester manual intervention to be reduced. Even user haven't written any test script for Android applications, MONVIS and AQT exercise applications automatically and detect applications bugs. MONVIS is automated testing tool without human knowledge of the expected behavior of the Android application. Thus, MONVIS test application correctness properties such as crash and exception. AQT leverage human knowledge about how the application should response. AQT check app install and launch, memory use, connectivity, user interface and stability. MONVIS and AQT analyze the structure of applications GUI and then explores it automatically by simulating user activities. It provides logs, screenshots and test report to user for determining the root cause if applications crash and find issues with applications quality. MONVIS and AQT test Android applications on different platforms, screen sizes and operating systems.

Keywords: Android application testing, GUI testing, Android application quality, testing automation, black-box testing

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Motivation

According to Gartner [1], Android accounted for 80.7 % of worldwide smartphone sales to end users by operating system in the fourth quarter of 2015. Android has become the most popular operating system. Android application testing represents a challenge, with Android fragmentation [2]. Android devices have different Android operating system version, screen sizes, performance and brand. Developers developed the application that work across all Android devices are extremely challenging and time-consuming. Testing requires a lot of human resource to complete. Therefore, the automated testing for quality of Android applications is more and more important. We develop a tool to test Android applications automatically without manpower.

1.2 Purpose

We present two black-box GUI testing tool to test the Android applications automatically. They are Monkey with Vision (MONVIS) and Application Quality Testing (AQT). MONVIS is automated testing tool without human knowledge of the expected behavior of the Android application. Thus, MONVIS test application correctness properties such as crash and exception. AQT leverage human knowledge about how the application should response. AQT check app install and launch, memory use, connectivity, user interface and stability. MONVIS and AQT test Android applications on different platforms, screen sizes and operating systems.

1.3 Organization

The outline of this paper is narrated in following. Chapter 2 shows related works including GUI testing tools, Android fragmentation, Android application testing. Chapter 3 lists preliminaries which are our basic knowledge for developing our tools. Chapter 4 depicted MONVIS. Chapter 5 depicted AQT. In Chapter 6, we present experiments to show results of our works. In Chapter 7, we summarize our contributions, limitation and future works to improve our tools and techniques.

Chapter 2 Related Work

MONVIS and AQT follow some Android automated testing framework and apply some user interface screenshot recognition techniques. The main issue of Android automated testing framework is how to detect bugs of applications following a set of guiding procedures and methods, such as model-based testing, UI testing scripts, capture-replay methods. Exploring applications GUI automatically is also an important issue. The following surveys some related works.

2.1 Android Automated Testing Framework

The following describes some important Android testing frameworks that are related to our work.

Amalfirano et al. [3][4][5] proposed some GUI crawling methods for crash testing and regression testing on Android applications. Their methods explored GUI by simulating user events and build related models correspondingly.

Liu et al. [6] introduced an idea of constructing an automatic testing framework and proposed a solution for mobile phone. The idea is to design a particular testing script for handling mobile device screen images. Screen verification, based on image-comparison, supports both the entire image and sub rectangle regions. The extended idea is using OCR for text data verification.

Semenko et al. [7] introduce an automated cross-browser testing tool (Browserbite) based on image segmentation and differencing in conjunction with decision tree and neural networks. Browserbite increase accuracy and reduce processing time.

UI Automator [8] is a UI testing framework provided by Google. The tool are suitable for cross-app functional UI testing across system and installed apps. It provides

APIs, which interacts between Android applications and testing system, to build UI tests.

Robotium [9] is an Android test automation framework that support for native and hybrid applications. User write test scripts to test Android applications.

Liu et al. [10] proposed an approach to automate the testing of Android applications based on the Capture-Replay method. The approach captures the user events, converts them into Robotium test script to replay the user events again, and verifies the outputs of Android UI components based on captured user interactions with interface.

Monkey [11] is a program that runs on Android device or emulator and generates user events such as clicks, touches and system-level events provided by Google. User use Monkey to stress-test applications in a random repeatable manner.

MonkeyRunner [12], released by Google without source code, provides APIs to write scripts that control Android applications on Android device or emulators. The tool is designed to test applications at functional levels and runs unit test suites. The MonkeyRunner API can take multiple test suites across multiple devices or emulators.

Google Cloud Test Lab [13] is a convenient cloud service platform with scalability for on-demand app testing. User can test Android applications on different brands, models, languages, screen orientations and Android operating system version. Users upload applications to Google Cloud Test Lab, and then install and run applications on virtual physical devices. It then returns the feedback of test results, including logs, videos, screenshots and Activity Map that graphically shows the visited paths of applications under test.

Appium [14] is a test automation framework for use with native, hybrid and mobile web apps. It drives Android and iOS applications using WebDriver protocol. Appium can test mobile apps from different languages and under different test frameworks by accessing Appium back-end APIs and DBs.

Model-based testing [3][4][5][13] is that crawls applications based on a model of the GUI depict the structure user interface and detects applications crash in Android. UI testing script provides APIs for UI tests to simulate user events within an application [10][11][12][14].

2.2 Vision-Based Technology for Testing

The followings are some vision-based technology testing methods that are relevant to our work.

Sikuli [15], which is an open source research project at MIT, provides some GUI interactive functions that sending events and monitoring the response on the screen . It is a capture/replay tool based on computer vision. The tool takes images and user written scripts to trigger components which match with image patterns.

Our laboratory member Chang [16] invented a tool using computer vision technology to analyze the screenshot and get the information as human did. The tool saves manpower and avoids losing application information from external components. The tool explores application to build a model including automata, state information, etc. However, application's advertisements such as banner and interstitial may cause a problem to the tool. When the tool explores the application, it analyzes screenshot to find contours in it that matches with application button contours with high possibility. Then it clicks each contour to check its response. If the responsive screenshot changes, the contour represents a button. The tool often needs to do a large amount of clicks. When the tool explores screenshot and meets advertisements, it will click advertisement, switch to web browser, link to the advertisement site, and never go back again. Thus, the tool is not qualified to build a good model.

To improve the accuracy of the Chang's tool [16], more information is needed to

exclude advertisements. Since XML files provide Android application UI information, it inspires us to simulate user events based on XML file and screenshot information together to increase accuracy to tools.

Chapter 3 Preliminaries

Some preliminary knowledge of building MONVIS and AQT are discussed in this chapter, including android application structure, GUI testing (Graphical User Interface testing), random testing and coverage criteria.

3.1 Android Application Structure

Android application are written in Java. The Android SDK (Software Development Kit) tools compile source code, data and resource files into an Android package (apk) files. Application components [17] are the essential building blocks of an Android applications. There are four types of application components that activities, services, content providers and broadcast receives. An activity represents a single screen of a user interface. A service is a component that runs in the background to perform long-running processes. A content provider manages a shared set of application data. A broadcast receiver is a component that responds to system-wide broadcast announcements.

3.2 Graphical User Interface Testing

A GUI provides a graphical front-end to the application. It implemented as an event-based system that user generator input to system and system provided graphical output.

GUI testing is a process to confirm the graphical user interface and specification of the application. GUI testing can be functional and non-functional. Functional GUI testing is to detect that user events are handle correctly. Difficulty of GUI testing are sequence problem and domain size problem. Sequence problem is that some function of the system may be occurred with a sequence of GUI events. Domain size problem is that system has too many GUI need to be tested.

Finite state machine, event-flow graph and GUI tree have been proposed to

automatically to generate test case that are simulate user behavior.

3.3 Random testing

Random testing feeds random inputs to applications. Its advantage is no specific knowledge of applications is needed. However, this method is difficult to get deep into the application logic.

Duran et al. [18] presented the random testing may often be more cost effective than partition testing schemes. Random testing allows one to obtain sound reliability estimates. Their experiments have shown some relatively subtle errors can be discover by random testing without a great deal of effort.

Sen [19] present a random partial order sampling algorithm (RAPOS) that solving multithreaded concurrent programs which exhibit wrong behaviors due to unintended interferences among the concurrent threads.

Amalfirano et al. [20] propose Monkey Fuzz Testing (MFT) tool on an application under test to define a stopping criterion, parameterized by the application preconditions. The tool solve the problem of stopping a random testing process at a cost-effective point that test adequacy is maximized and no testing effort is wasted.

Random testing feeds random inputs to applications. Without knowledge of the applications, this method is difficult to get deep into the application logic.

3.4 Coverage criteria

Huang and Wang [21] proposed Android Black-box Coverage Analyzer (ABCA). ABCA inserts commands into the Android application package file to dump coverage data on executed classes, method names and source code statement. That is ABCA produces source code coverage on Android application to keep track of its execution

without access to the source code directly.

Amalfirano et al. [22] presented coverage criteria that use events and event sequences to specify a measure to help determine whether a GUI has been adequately tested. Algorithms are constructed by event-flow graphs and an integration tree for GUI, and evaluate the coverage of test suite with respect to the coverage criteria.

App Quality Alliance [23] announced testing criteria for Android applications. Testing criteria have been developed to test different features and applications. Testing criteria consist of several test cases such as install and launch, memory use, connectivity, event handling, messaging and calls, external influence, user interface, language, performance, media, menu, functionality, keys, device and extra hardware specific tests, stability, data handling, security, multiplayer, metadata, privacy and user permissions, etc. Applications pass the testing criteria, ensuring their quality.

We implement code coverage in MONVIS to measure test case generator efficiency by ABCA [21].

Chapter 4 Monkey with Vision

Chiang [24] proposed a framework for automated testing on Android applications, which uses UI Automator dumping Android UI to XML files for GUI testing. However, there is a shortcoming that XML files do not contain the external components such as WebVew and Unity. This shortcoming can be solved by introducing computer vision technology which analyzes screenshot for additional information other than XML files. XML and image, such as PNG, working together compensate each other to gain more valuable information in a state.

MONVIS is a toolset for GUI testing of Android applications. It trigger sequences of user events and exposing failures. GUI is implemented as an event-based system that trigger as user events. MONVIS executes user events on the GUI and observes the result. MONVIS use sequences of event that are simulate human behaviors. MONVIS explores the GUI and detects crashes of the Android application.

We explore the application under test (AUT) by clicking the buttons and input text in EitText on the screen and collect the response of the AUT. Each state is defined as a screenshot and user events (e.g. click, input). There are many traditional coverage criteria, including line coverage, branch coverage, data-flow coverage, method coverage for white-box testing when the source are available. There are also other coverage criteria which is defined on the system or which is module interface for black-box testing. An empirical study suggested that coverage-based techniques are effective in detecting software faults [25]. MONVIS do quality test without the source code of AUT. Thus it is beneficial to the some aspects. If clients prefer to protect their intellectual properties, MONVIS is useful to bridge the clients and the test service providers. With the detailed coverage data, MONVIS can also help diagnosing why some parts of the code cannot be

reached or identified those code segments related to certain bug traces. Figure 4.1 shows MONVIS architecture. MONVIS consists of four main components, image-based object recognizer, test case generator, test case evaluator, coverage code instrumentor and test coverage collector.

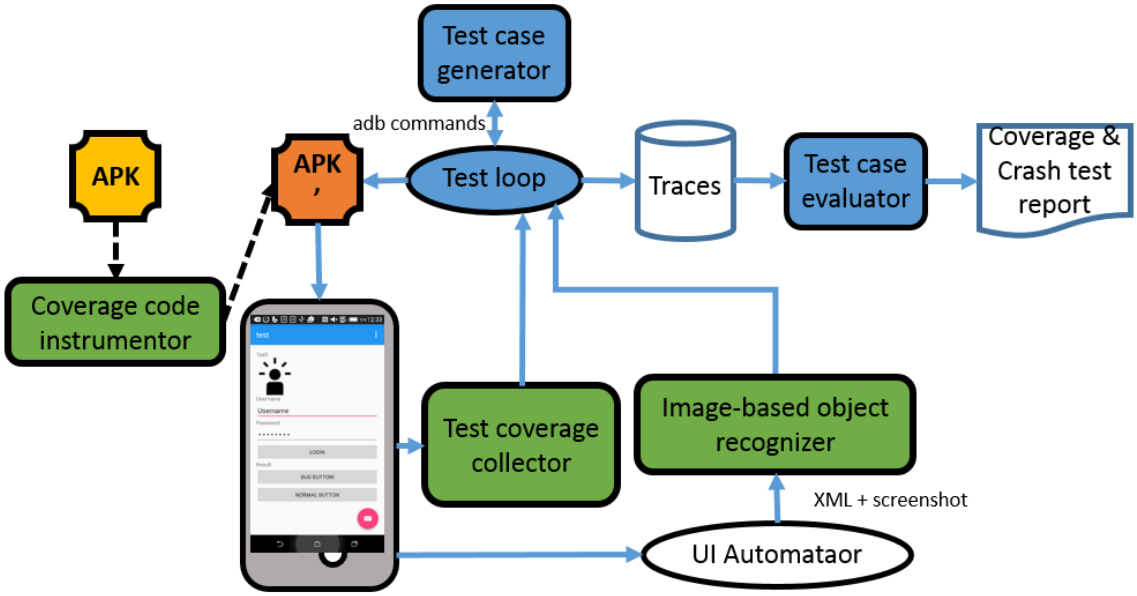

Figure 4.1 MONVIS architecture

4.1 Image-based object recognizer

The main purpose of Image-based object recognizer is to provide legal actions for test case generator. Image-base object recognizer consists of data extracting of XML and data extracting of image to support legal actions. In the following, data extraction of XML section presents how to extraction XML files from Android device and how to analyze XML files. Data extraction of image section presents how to extraction image files from screenshot and how to analyze images by computer vision.

4.1.1 Data extraction of XML

UI Automator

The structure of UI components in an activity is a hierarchy view. UI components includes Button, TextView, WebView and view groups such as Framelayout, linearLayout and RelativeLayout. UI Automator extracts the UI information from Android applications. The tool dump the current activity in hierarchy view to the XML file. Each view and view group is represented as a node that has 17 distinct attributes in a file. Figure 4.2 shows an example of the activity's screenshot and XML file generated by Android Debug Bridge (adb) and UI Automator. MONVIS extracts information from XML files by XML parser. Each node in the XML files that has 17 attributes represents UI components. Only four attributes among them are used in MONVIS, including "class", "package", "clickable" and "bounds". The "class" attribute represents the type of view such as Button, ImageView, TextView. The "package" attribute represents the package name of the application. The "clickable" attribute represents the views and view groups is clickable on the UI. The "bounds" attribute represents the position of the rectangle space on the views and view groups. A user event is an action through GUI by users. Each event can

be distinguished as command input event or data input event. MONVIS parse the XML files and select nodes of which “clickable” attribute are true to create user events.

Figure 4.2 Screenshot of an activity with partial corresponding XML

Challenge of insufficient information in XML file

There is a shortcoming that XML file do not contain the external components such as WebView Unity and Cocos2d-x. WebView [26] is an extension of Android View class that displays web pages as a part of activity layout. Unity [27] is the game development platform that build high-quality 2D and 3D games and deploy them across mobile. Cocos2d-x [28] is an cross platform free 2D game engine for mobile game development, known for its speed, stability, and easy of use. Figure 4.3 shows a screenshot and a XML file of WebView. Figure 4.4 shows a screenshot and a XML file of Unity. Figure 4.5 shows a screenshot and a XML file of Cocos2d-x. Since XML files cannot reserve UI components with missing buttons, text, etc., applications built on WebView, Unity and Cocos2d-x are not suitable to determine user events by XML file only. To restore the missing information, the screenshot is a good source to compensate for the shortcoming of XML file.

Figure 4.3 WebView screenshot of an activity with partial corresponding XML

Figure 4.4 Unity screenshot of an activity with partial corresponding XML

Figure 4.5 Cocos2d-x screenshot of an activity with partial corresponding XML

4.1.2 Data extraction of image

The main purpose of data extraction of image is that some clickable buttons cannot be identified by parsing the XML files. Therefore, MONVIS extracts addition information from images based on six steps: (1) converting RGB image to grayscale image, (2) filtering image, (3) detecting edges, (4) finding contours, (5) removing uninterested contours, (6) merging XML buttons and computer vision contours together.

Convert RGB image to grayscale image

Tippy Tipper is a tip calculator of open source that split bill for entered bill via custom keypad. Figure 4.6 shows a screenshot of Tippy Tipper. Tippy Tipper is used to plain MONVIS procedures.

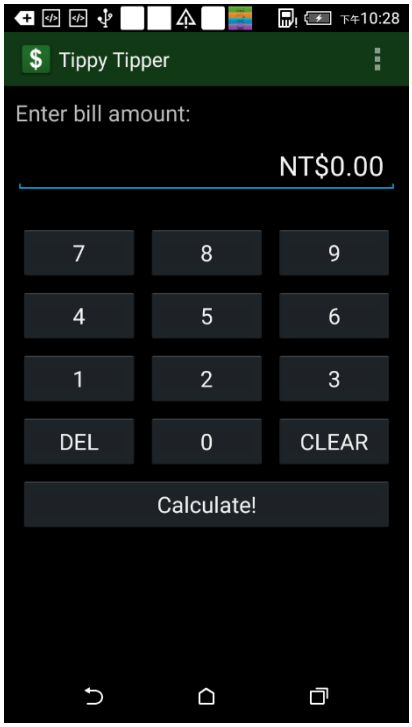

Figure 4.6 Screenshot of Tippy Tipper

Firstly, a preprocessing converts the color images into grayscale images that reduces redundant color information without losing critical contour information and increases contour searching speed. Standard NTSC conversion formula, see Equation 1, is used to calculate the effective luminance of a pixel. Figure 4.7 shows a gray converted screenshot of Tippy Tipper.

Intensity = 0.2989*red + 0.5870*green + 0.1140*blue (1)

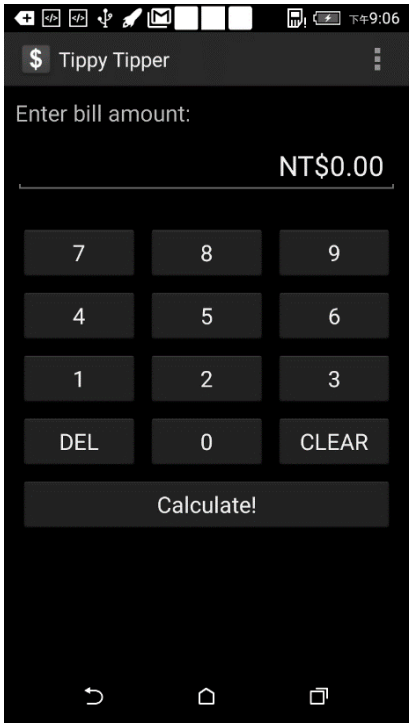

Figure 4.7 Gray converted screenshot of Tippy Tipper

Bilateral filter

Image filtering is image processing that delete unfavorable or enhance favorable pixels for specific purpose to improves image quality for further analysis. Tomasi [29] proposed bilateral filtering based on a nonlinear combination of nearby image characteristic values. Bilateral filtering retains objects with smooth edge-preserving property. Changzhen et al. [30] applied an adaptive bilateral filtering effective solution to compensate for edge loss problem by Canny edge detector. The bilateral filtering method

is applied to the gray image before edge detector works. Figure 4.8 shows a screenshot of Tippy Tipper after dealing with bilateral filtering.

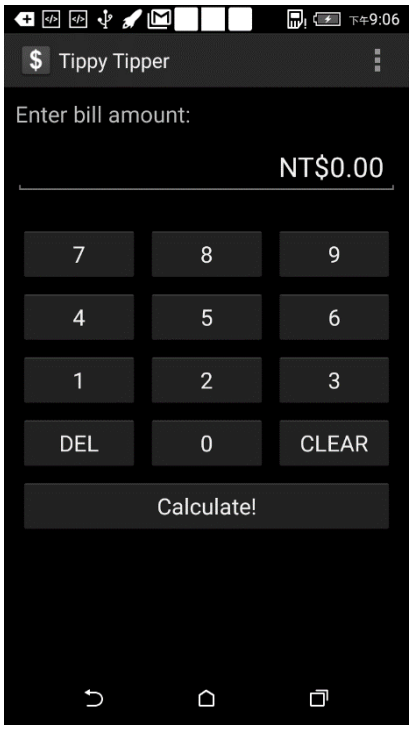

Figure 4.8 Screenshot of Tippy Tipper after dealing with bilateral filtering.

Detecting edges

Canny [31], presented Canny edge detection algorithm. Canny edge detector aims to satisfy three main criteria that low error rate, good localization and minimal response. Figure 4.9 shows clear edges of a screenshot from Tippy Tipper, after applying Canny's edge detection algorithm.

Figure 4.9 Screenshot of Tippy Tipper after applying Canny's edge detection algorithm

Finding contours

Arbelaez et al [32] presented two fundamental problems in computer vision that contour detection and image segmentation. We use contour detection to identify contours that closed area of arbitrary shape in a screenshot. Then, we use rectangle area substituted for arbitrary shape. Figure 4.10 shows a screenshot of Tippy Tipper contours.

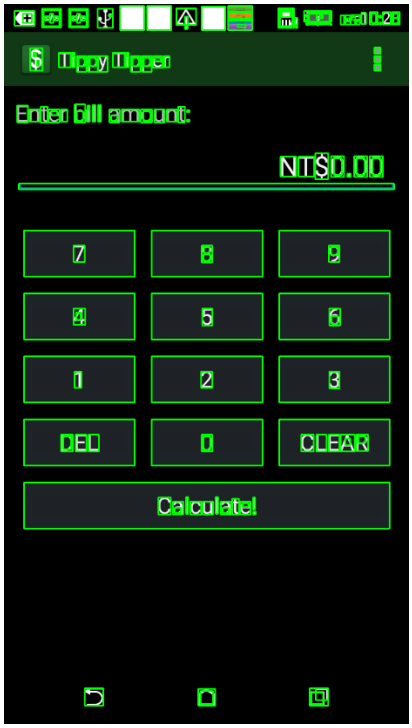

Figure 4.10 Screenshot of Tippy Tipper contours.

Remove uninterested contours

Small rectangles with high probability are not buttons, so contours smaller than a threshold size of fingertip are removed. A region of interest (ROI) is an interested subset of the image for a specific purpose. An Android system uses s a notification area to alert users on the top of screenshot, which is irrelevant to further quality analysis, so this region is removed beforehand. Figure 4.11 shows screenshot of Tippy Tipper after removing small contours and out of ROI region.

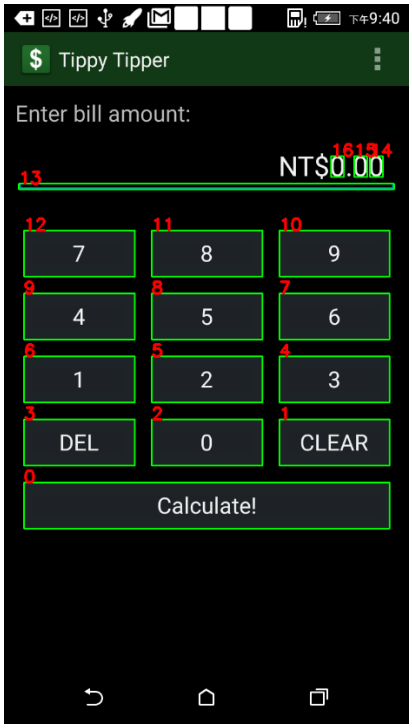

Figure 4.11 Screenshot of Tippy Tipper after removing small contours and out of ROI region.

Merge XML buttons and vision contours

After parsing XML files, MONVIS creates user events for nodes of which "clickable" attribute is true. Figure 4.12 show buttons of xml (blue line) and vision contours (blue line). It is obvious that most of them are overlapped, which shows high quality of computer vision technology. To increase test efficiency, both sets are union and some vision contours with center bigger than threshold of xml clickable button center are remove. Figure 4.13 show union of xml clickable button and vision contours without redundancy.

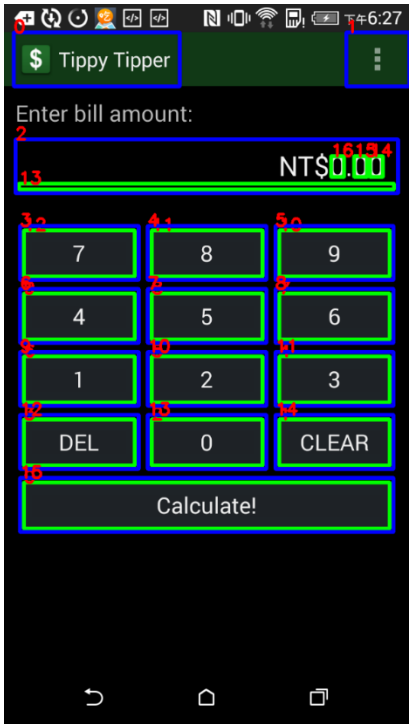

Figure 4.12 Buttons of xml (blue line) and vision contours (blue line).

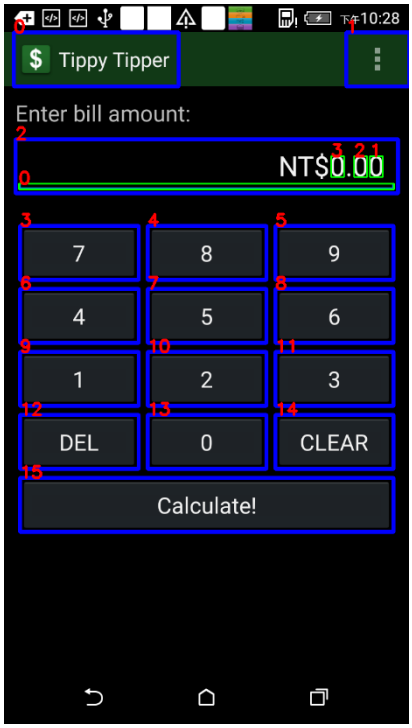

Figure 4.13 Union of xml clickable button and vision contours without redundancy.

4.2 Test case generator

Test case generators generate possible activities that simulate human behaviors interacting with GUI, such as click button and input text.

Two test case generator algorithms, MonkeyXML and MonkeyCV, are proposed here for MONVIS.

4.2.1 MonkeyXML

MonkeyXML is a test case generator in MONVIS Trace Collector. It is a random walk algorithm. MONVIS parses the XML files of nodes, and select nodes of which "clickable" attribute are true to create user events. MonkeyXML in this state generates all legal actions and returns the random choice from all of them. Algorithm 1, see below, is pseudo-code for MonkeyXML. Algorithm 2 shows function of selecting legal actions randomly from XML file.

Algorithm 1 MonkeyXML

```
1:  procedure MonkeyXML(AUT)
2:      Trace = emptySet
3:      while |Trace| < traceAmount do
4:          restart AUT
5:          t = emptyTrace
6:          while |t| < maxTraceLength do
7:              Add current state s to the end of t
8:              nextEvent ←selectActionXML(XML of s)
9:              execute nextEvent
10:             if Termination is True then
11:                 break
12:             end if
13:         end while
14:         Add t to Traces
15:     end while
16: end procedure
```

Algorithm 2 selectActionXML

```
1: function SELECTACTIONXML(XMLofs)
2:     clickableButtonXML ← collect clickable node in XML
3:     return randomly pick a button from clickableButtonXML
4: end function
```

4.2.2 MonkeyCV

Extending user events defined in clickableButtonXML, clickableButtonCV is union of clickableButtonXML and vision contours. MonkeyCV in this state gets all legal actions and returns the random choice from all of them.

The pseudo-code of the MonkeyCV is shown in Algorithm 3,see below. Algorithm 4 shows function of selecting legal actions randomly from union of XML file and vision contours.

Algorithm 3 MonkeyCV

```
1: procedure MONKEYCV(AUT)
2:     Trace = emptySet
3:     while |Trace| < traceAmount do
4:         restart AUT
5:         t = emptyTrace
6:         while |t| < maxTraceLength do
7:             Add current state s to the end of t
8:             nextEvent ←selectActionXML(XML of s)
9:             execute nextEvent
10:            if Termination is True then
11:                break
12:            end if
13:        end while
14:        Add t to Traces
15:    end while
16: end procedure
```

Algorithm 4 selectActionCV

1: **function** SELECTACTIONCV($XML, PNG, ROIofs$)
2: *clickableButtonXML* ← *collect clickable node in XML*
3: *grayImage* ← *Convert RGB PNG to gray image*
4: *filteredImage* ← *bilateralFilter(grayImage)*
5: *edges* ← *cannyEdgeDetector(filteredImage)*
6: *contours* ← *findContours(edges)*
7: remove small contours from contours
8: remove out of ROI contours from contours
9: *clickableButtonAll* ← *contours* ∪ *clickableButtonXML*
10: **return** *randomly pick a button from clickableButtonAll*
11: **end function**

4.3 Test case evaluator

After running test cases on a AUT, an automated testing tool collects corresponding traces for performance evaluation. A high quality evaluation is often not an easy work without human involved. MONVIS uses information retrieved from Android log system which provides a tool Logcat to access itself. Based on retrieved information, the occurrence of crashes and Android runtime error is reported. Logcat [33] is a command-line tool that dumps system log messages provided by Google. Message includes message from system and stack traces, exceptions thrown from apps for some specific events such as errors. Java exceptions of apps should catch or handle when errors occur; otherwise, applications could be crash in serious situations. MONVIS gets Android Runtime Errors by logcat, and generates report about events that trigger crash and Android Runtime Error to users.

.

4.4 Code coverage

Code coverage is an important information to measure testing tools quality for fault detection. The higher code coverage, the more reliable testing technique it is. MONVIS uses Android Black-box Coverage Analyzer (ABCA) [21] instrumenting AUT and produce code coverage information. ABCA analyze can measure code coverage without source code.

4.4.1 Coverage code instrumentor

MONVIS need to instrument an application under testing (AUT) before collecting traces. Given an APK file, ABCA inserts commands into the APK file to dump coverage data on executed classes, method names, and source code statement. A simple way to

evaluate significance of a test case is to see how many lines or methods it covers, i.e. the line coverage. ABCA produce source code coverage report on AUT execution while the source code of the AUT is actually not available.

4.4.2 Test coverage collector

The aim of test coverage collector is to measure testing code coverage of AUT. While an instrumented AUT is running with test cases on Android device, ABCA on PC collects relevant information through USB line. After the test session completes, ABCA generates a report on code coverage of the original Java source code.

4.5 Comparison

Vision-Based Technology for Automated App Testing merges close contours by threshold and merges fully contained rectangles by clicking and check. The advantage of Vision-Based Technology for Automated App Testing is no false positive clickable button. One major drawback of vision-based technology is false-negative clickable button problem. It clicks contours, and check their responses. If a response is a screenshot change, the contour is classified as a button. However, if the button is designed without change screen after clicking, the button is classified as unclickable button. Figure 4.14 shows false-negative click button problem. Another limitation of this approach is inaccurate bounds problem. The tool merges close contours as the same contour. Nevertheless, if buttons are designed closely, in such case merged buttons are wrong. Figure 4.15 shows inaccurate bounds. MONVIS combines data extraction of XML and data extraction of image together, so both problems are prevented. Figure 4.16 shows MONVIS solves false-negative click button problem. Figure 4.17 shows MONVIS solves inaccurate

bounds problem. However, MONVIS has a disadvantage of false-positive clickable button problem with a little possibility.

Another drawback of vision-based Technology are long execution time. MONVIS only returns contours not appearing on XML files and no other post-processing like ones in vision-based Technology, so it has no redundancy node and short execution time.

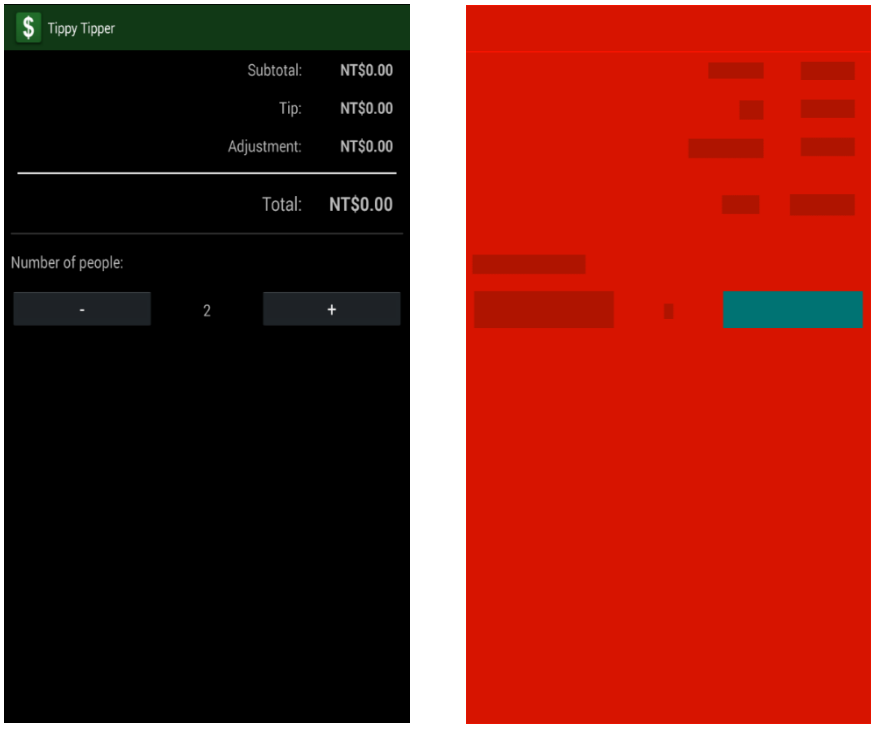

Figure 4.14 false-negative click button problem

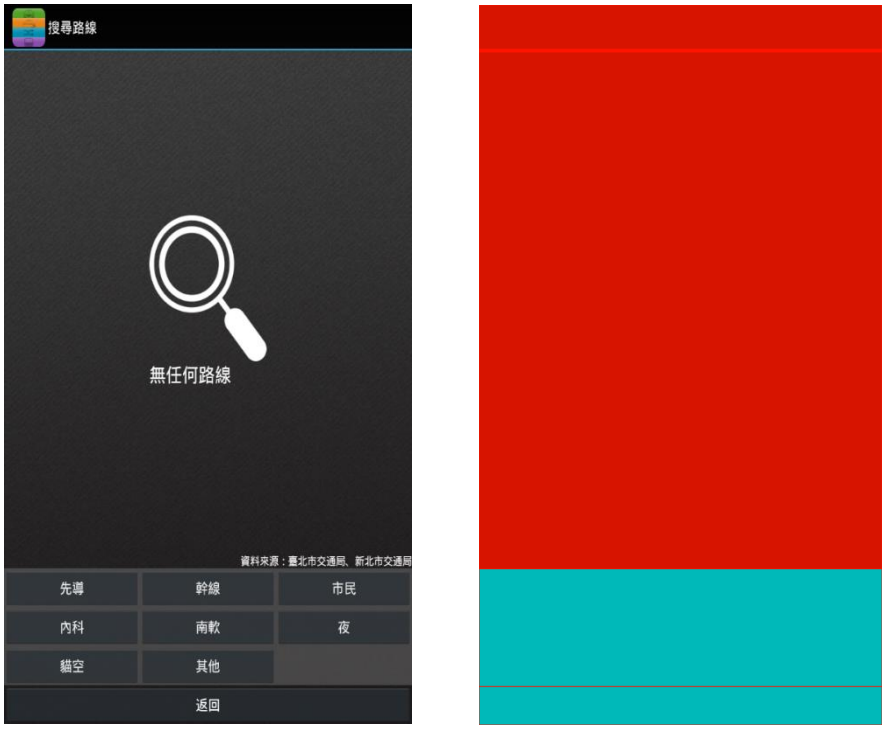

Figure 4.15 inaccurate bounds problem

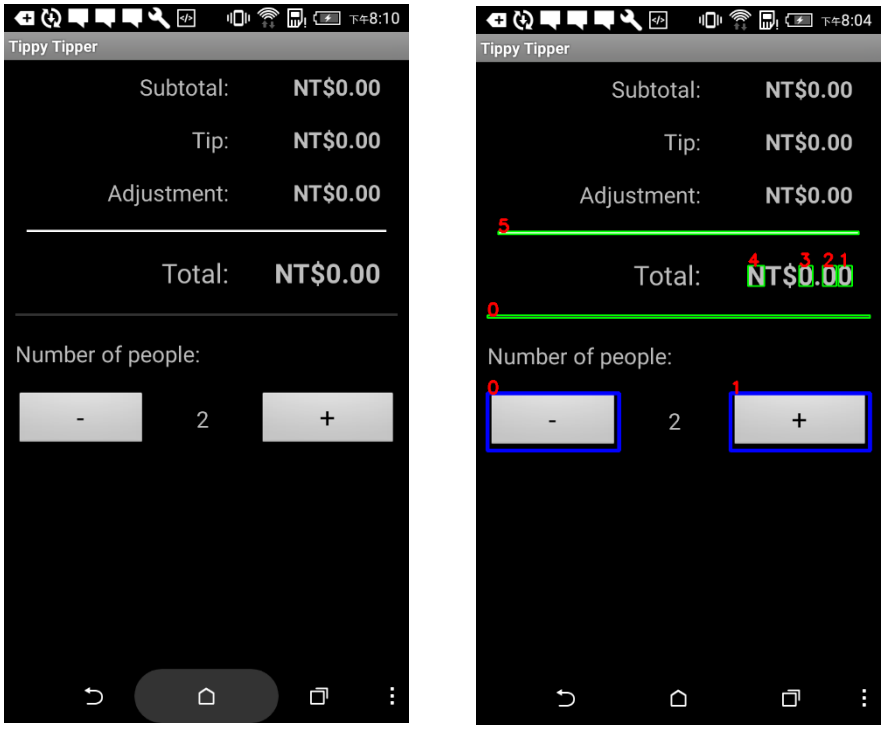

Figure 4.16 MONVIS solve false-negative click button problem

Figure 4.17 MONVIS solve inaccurate bounds problem

Chapter 5 Application Quality Testing

App Quality Alliance [23] announce testing criteria for Android applications. Testing criteria have been developed to test different features and applications. Testing criteria include several test cases such as install and launch, memory use, connectivity, event handling, messaging and calls, external influence, user interface, language, performance, media, menu, functionality, keys, device and extra hardware specific tests, stability, data handling, security, multiplayer, metadata, privacy and user permissions. Applications passing the testing criteria means it ensure high quality.

Wen-Hong Chiang proposed Experiment of a framework for automated testing of Android Applications [24] that provided an Android testing framework. We design AQT based on Experiment of a framework for automated testing of Android Applications. The framework provide adb commands, the construction of folders, parse of XML files, etc. Thus, AQT focus on developing test case generator and test case evaluator. Figure 5.1 shows AQT architecture. AQT has three main components such as trace recorder, test case generator and test case evaluator.

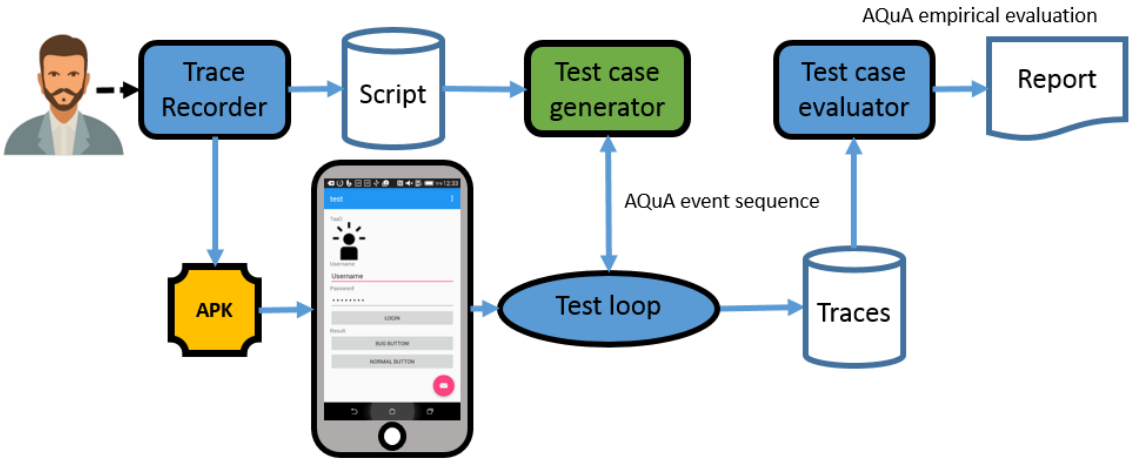

Figure 5.1 AQT architecture

5.1 Test case generator

The main purpose of test case generator is to simulate user behavior. We want to generate large amount of traces. MONVIS simulate user events by random click buttons or input text. However, only click buttons and input text cannot totally simulate user behavior. Thus, we define AQT event sequence to simulate user behavior by combining user actions to a sequence.

5.1.1 AQuA test criteria

We develop 7 AQuA test criteria such as install and launch, memory use, connectivity, user interface and stability in our work. We describe detail of them in chapter5.3 – chapter5.7. We generate test case for 7 AQuA test criteria by AQuA event sequence.

5.1.2 AQuA event sequence

The goal of AQuA event sequence is generate traces for AQuA test criteria. AQT simulate user events by action include control the device install app, click button , input text, click back button, click home button, re-launch app, restart app, rotate screen, network. AQuA event sequence is a sequence of actions (e.g. 1. Install app 2. Click button 3. Click home button 4. Re-launch app).

5.2 Test case evaluator

We collect lots of traces by test case generator automatically, how to evaluate each trace is difficult. Thus, AQuA test case evaluator evaluate traces passed of failed by AQuA empirical evaluation.

5.2.1 AQuA empirical evaluation

Most users use friendly Android applications. AQT define friendly Android applications principle such as application have multiple display format, completely

removed from the device, should resume from where it was suspended, must not freeze or exit unexpectedly at any time, etc. Those feature let applications have good user experience. If the traces conflict with our friendly Android applications principle, the traces failed in the test.

5.3 Install and Launch

5.3.1 Uninstall App

The Android application that must install and uninstall from the device. If the Android application cannot install and uninstall successfully, the application should be modified. Tester test uninstall app criteria process that install AUT in Android device, check AUT install successfully remove AUT by manual. Figure 5.2 shows an example of the tester from the settings menu of the device, uninstall the application by manual. The process waste tester time for testing.

Uninstall App test criteria:

1. Required for all applications.
2. The application is completely removed from the device.

The processes of AQT uninstall app method of each AUT can be described as follows AQuA event sequence:

1. Install the application via adb command.
2. Launch the application.
3. Click the application's button to check application install successfully.
4. Uninstall the application via adb command.

AQuA empirical evaluation:

1. Check they are no data form the application remains on the device.

AQT print all packages and filter to only show third party packages. AQT check AUT package in package manager list. If AUT package does not in package manager list, it mean AUT uninstall successfully.

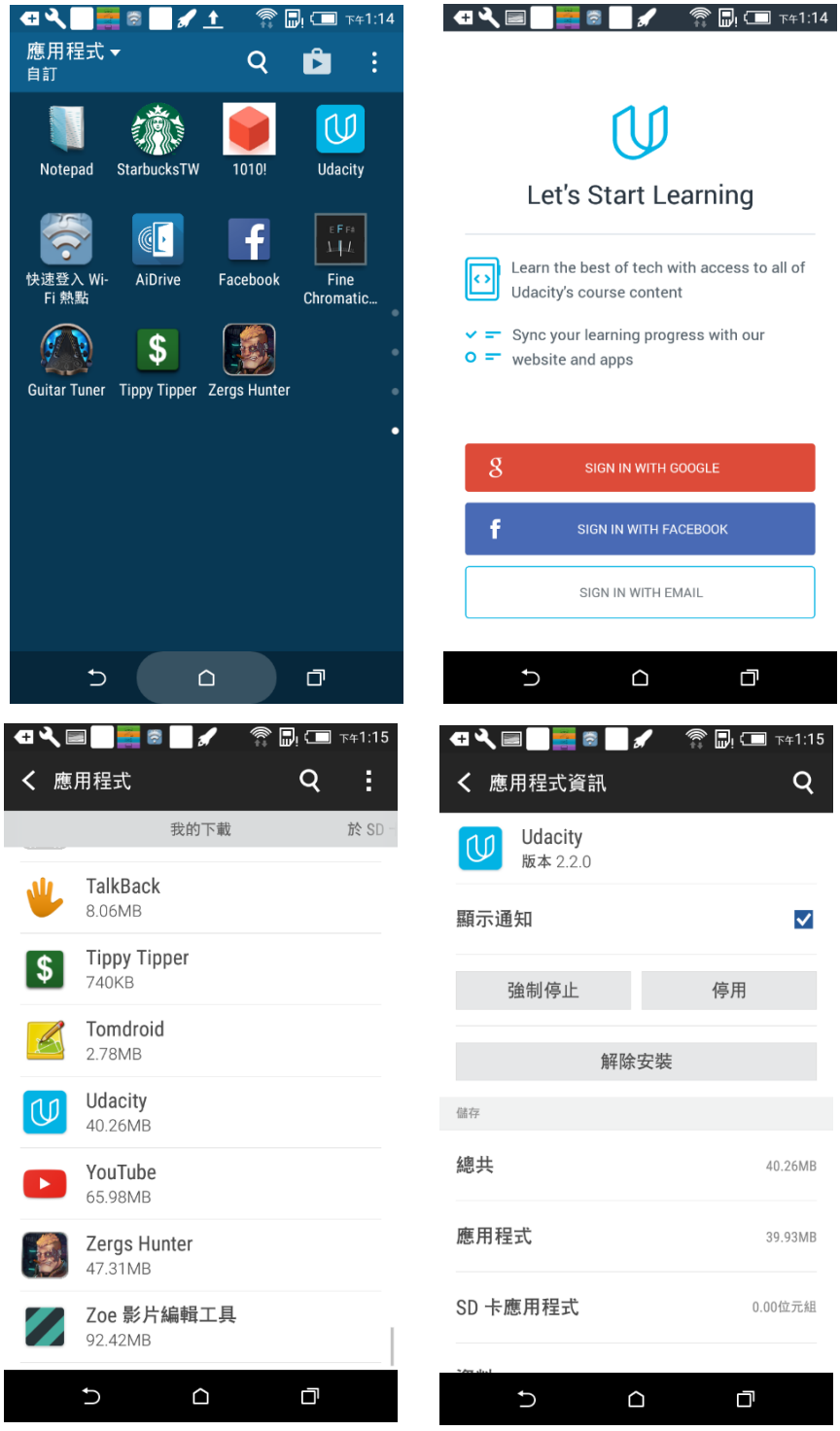

Figure 5.2 Tester uninstall app manually

5.4 Memory Use

5.4.1 Multiple Launch

When the user use the application, user can tap home key to return home and choose another app. If the Android application suspend and re-launch is handle incorrectly, the application should be modified. Tester test multiple launch criterial process that install AUT in Android device, start to test the application, suspend the application by device home key and re-launch the application from the app menu. Figure 5.3 shows an example of the tester observe the application multiple launch result.

Multiple launch test criteria:

1. Required for all applications.
2. The application should resume from where it was suspended
3. Some applications may be designed to reset to the initial condition if that makes more sense than resuming
4. It must not be possible to see two instances of the application running Suspend and re-launch of the application is handled correctly.

The processes of AQT multiple launch method of each AUT can be described as follows AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.
2. Install the APK file on the Android device.
3. Click the application's button to check application install successfully
4. Simulate user events.
5. Simulate home button by KeyEvent

AQuA empirical evaluation:

1. Compare xml files between before simulate home button and after simulate home button.

AQT generate report and record failures that the application should not resume and lose information from where it was suspended.

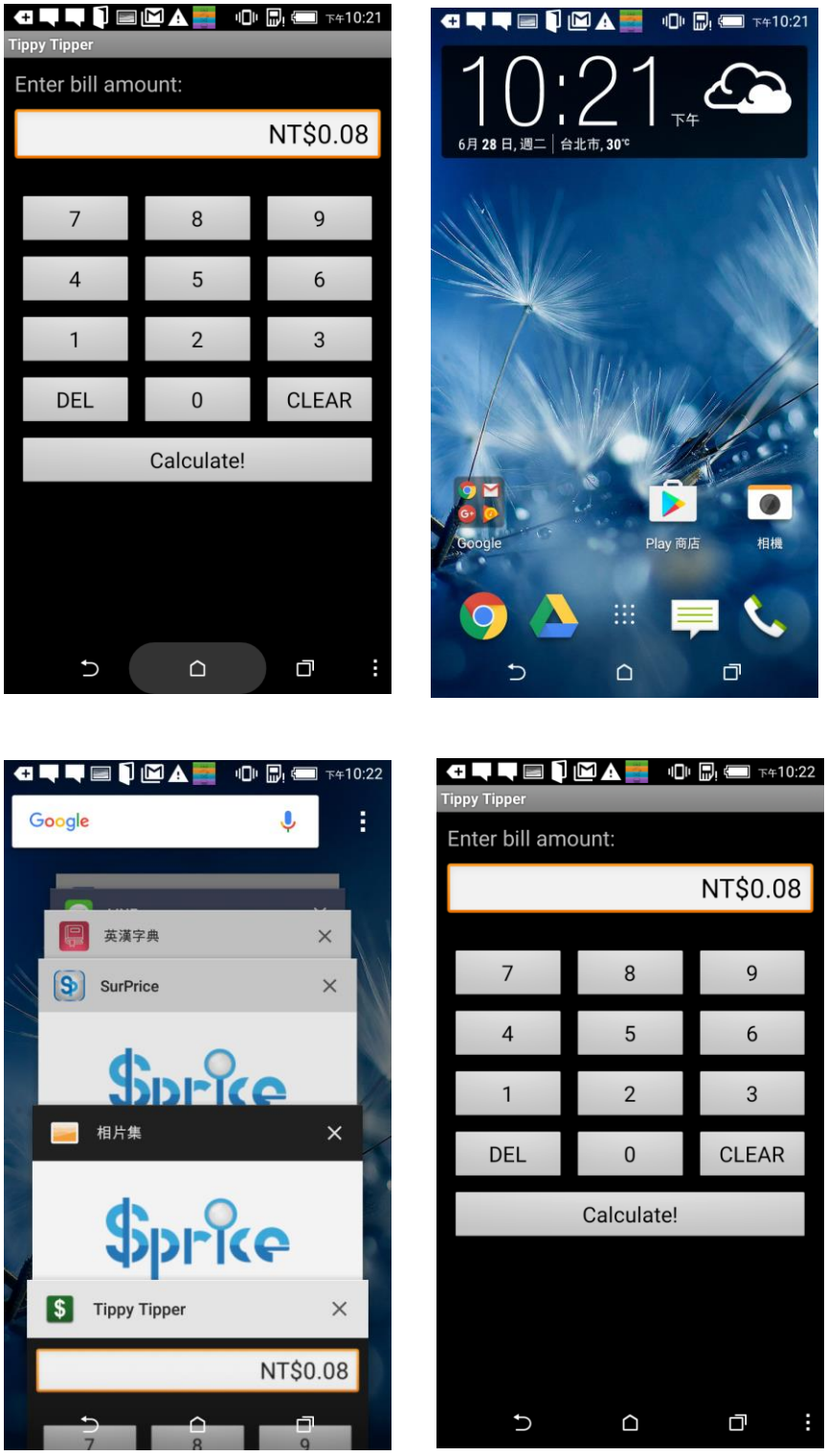

Figure 5.3 Tester suspend and re-launch app manually

5.5 Connectivity

Application connects to the internet or local network for update information from server. A device can have various types of network connections such as Wi-Fi or mobile network connection. When the Application uses network capabilities, it must be able to handle network delays and any loss of connection. Tester test network connectivity criteria process that install AUT in Android device, check AUT install successfully, and observe the Application behavior during the testing. Figure 5.4 shows an example of the tester observe the application behavior and detect error message to the user indicating an error with the connection.

Error message Toast [34], Snackbar [35] and application information message. A toast is a view containing a quick little message for the user. Toast is a floating view over the application view. Snackbar show a brief message at the top or bottom of the screen. Toast and Snackbar automatically disappear after a timeout or user interaction on the screen. Figure 5.4 shows an example of the Snackbar, Toast and application information message.

AQT parse the error message XML files and detect error message. However, the XML files cannot provide error message. Toast and Snackbar are floating view over the application, XML files only provide application view. Application information message can be provided by XML files, but they have too much error message type to detect. Thus, AQT use computer vision technique to detect error message. Pixel compare method compare the two images in pixel level to check if the screenshot has pixel level difference. If there are some changes in the two screenshots, AQT will label the error message. AQT assume that two screenshot have the error message if there is less than 90% different pixel.

We assume 10% error for notification message that time and application information on the screenshot.

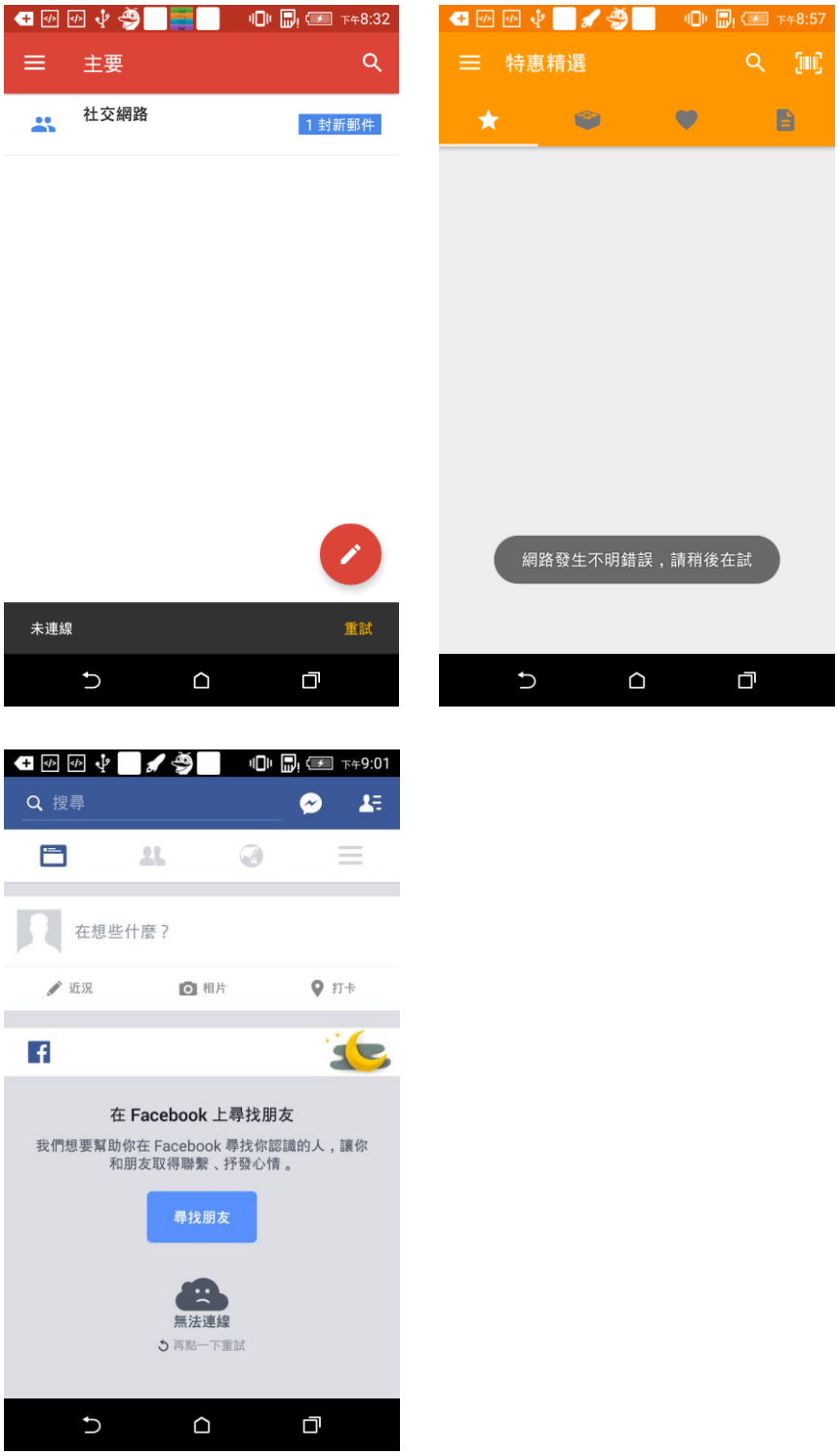

Figure 5.4 Connectivity error message

5.5.1 Network connectivity - Network delays and the loss of connection

Network delays and the loss of connection test criteria:

1. Required for application which uses network connection.
2. The application will work until time out and then give an error message to the user indicating there was an error with the connection.

The processes of AQT network delays and the loss of connection method of each AUT can be described as follows

AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.
2. Install the APK file on the Android device.
3. Click the application's button to check application install successfully
4. Simulate user events
5. Turn Airplane mode on
6. Turn Airplane mode off

AQuA empirical evaluation:

1. Pixel compare screen before airplane mode on and after airplane mode. If screen are different indicate app give an error message to user.

   When the Application uses network capabilities, it must be able to handle network delays and any loss of connection.

5.5.2 Network connectivity - Airplane mode

Network connectivity – airplane mode test criteria:

1. Required for application which uses network connection.
2. The application will give a meaningful error message to indicate that the device is in Airplane mode and the application cannot run successfully.

The processes of AQT network connectivity – airplane mode method of each AUT can be described as follows

AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.
2. Install the APK file on the Android device.
3. Turn Airplane mode on
4. Click the application's button to check application install successfully.
5. Simulate user events
6. Turn Airplane mode off

AQuA empirical evaluation:

1. Pixel compare screen before airplane mode on and do not turn airplane mode on. If screen are different indicate app give an error message to user.

AQT dump the screenshot files and compare images to detect error message. Then, AQT generate report and record failures that the application do not give an error message to indicate the device is in network delays or airplane mode.

5.6 User Interface

User interface is the space that interactions between humans and machines. Android provides UI components, UI controls and UI modules to build the application graphical user interface. Android has hundreds of devices with different screen sizes, ranging from small to extra-large mobile devices. User interface testing lets developer ensure that the application meets its functional requirements and achieves a high quality. Human tester verify the application behavior is most simple for GUI testing. However this manual method can be time-consuming, error-prone and tedious. Another method is to write UI tests script for automatic GUI testing. Wring UI tests script allows tester to run tests quickly and reliably. Android provides UI testing framework Espresso [36] that allow developer to programmatically simulate user actions and test complex user interactions. However writing UI tests script method can be time-consuming to write test script and if the application UI is modified, the scripts should be rewrite. AQT provides the methods that tester do not need to manual and write test script by exploration the application and analysis the screenshot automatically.

5.6.1 Multiple Display Format Handling

Some Android applications support multiple display in portrait or landscape formats by design. The elements of the application should be correctly format in different Android devices. If the Android application does not support multiple display formats, the application should be modified. Tester test multiple display format handling criteria process that install AUT in Android device, start to test the application and operate the application and make use of all available display formats in multiple function. Tester rotate the screenshot by rotate Android device and observer screenshot change format

successfully. Figure 5.5 shows an example if the tester observe the application portrait and landscape format. The application should switch correctly between the display orientations. Where the device and Application can display in multiple formats (e.g. portrait / landscape, internal / external display), the elements of the application should be correctly formatted in all display environments.

Figure 5.5 Tippy Tipper portrait and landscape screenshot

AQT test multiple display format handling check the display orientations and TextView in application correct. AQT dump app portrait and landscape resolution provided by Android OS and xml files. The node that class = “android.widget.FrameLayout” represent the application size of its largest visible padding on the xml files. Thus, AQT use “android.widget.FrameLayout” information to compare with Android application resolution to check the application display orientations are correct. Figure 5.6, 5.7 shows portrait xml files and landscape xml files.

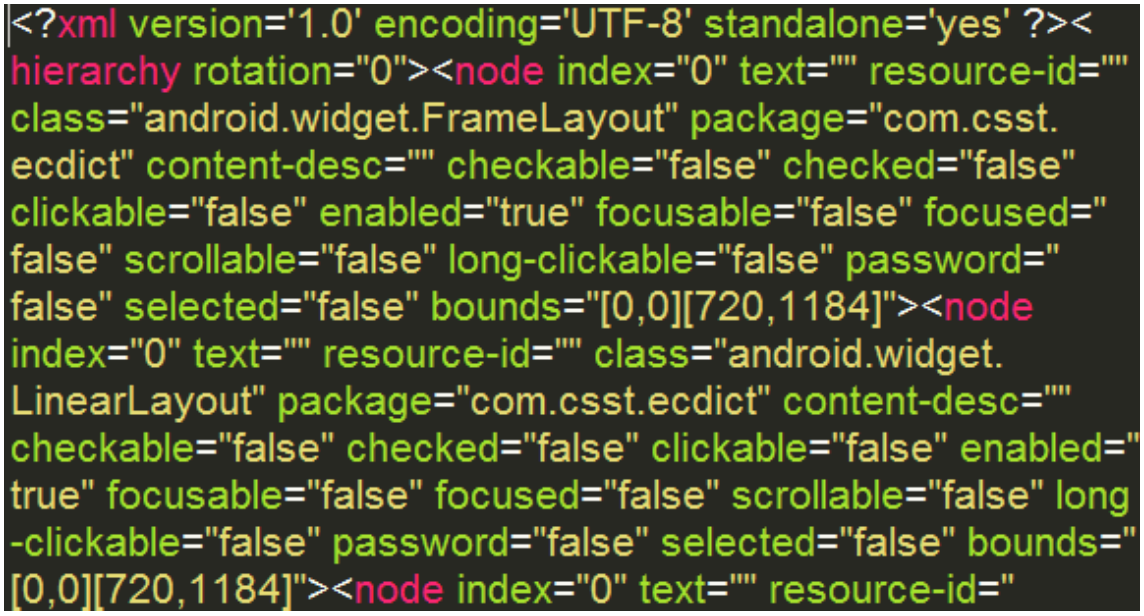

Figure 5.6 Portrait xml file

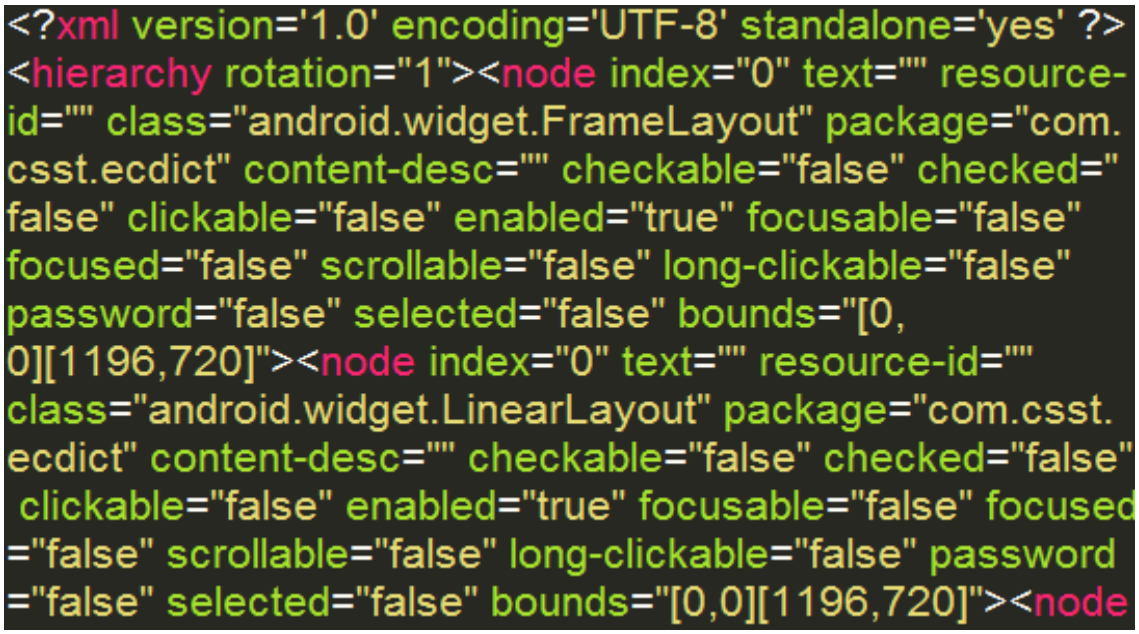

Figure 5.7 Landscape xml files

When the device rotate the screen orientation changes that application restore its state. The system destroys and recreates the activity for apply new screen configuration alternative resources when the screen orientation changes. Thus, developer should let activity completely restores its state when it is recreated. An Activity [37] is an

application component that provides a screen with users can interact. The system uses the Bundle instance state to save information about each View object in activity. Activities in Android OS are managed as an activity stack. When application create a new activity, it is placed on the top of the stack and becomes the running activity and the previous activity remains below it in the stack. When the new activity stop old activity will come to the screen. Figure 5.8 [37] shows the state paths of an Activity lifecycle.

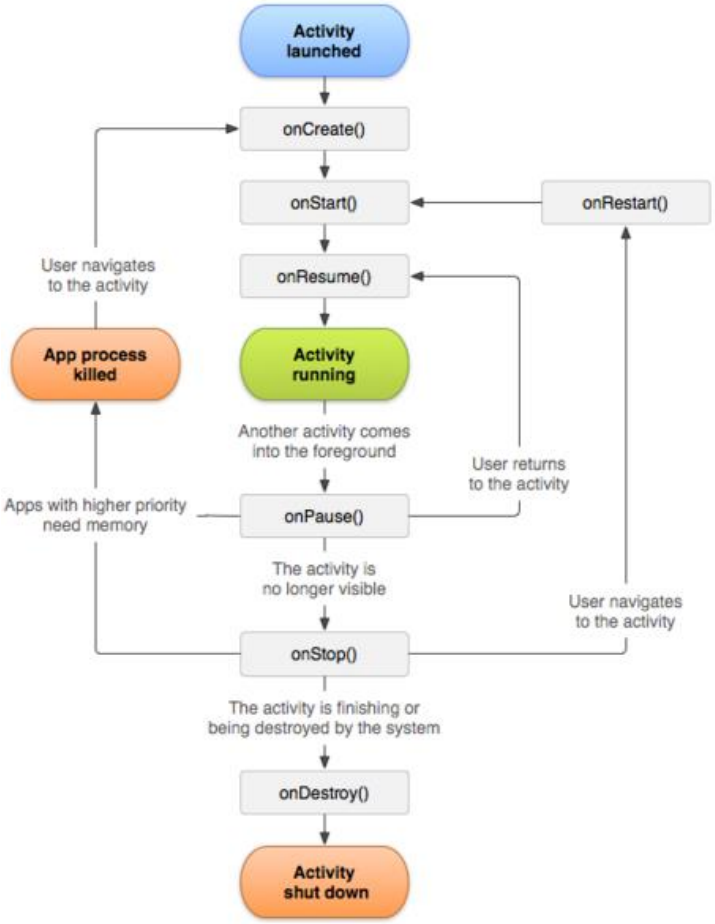

Figure 5.8 State paths of an Activity lifecycle

Application provide TextView and EditText to present information and input value by user. When the device rotate the screen orientation changes that application should restore its state. If the application do not restore is state, the application information will be error. AQT check this problem by vector space model. AQT generate user event such

as click button and input text. After trigger user event, AQT analyze two xml to detect which node is change that will check this node on portrait and landscape. If two xml change more than one node, we assume the application change to another state. Thus, we do not need to check this condition. AQT dump portrait and landscape XML files with same state. The most important problem that we cannot detect the same node on portrait and landscape XML files. Thus, AQT use vector space model [38] to solve the problem. Figure 5.9 shows the node has 17 attribute on XML file. We define query node is the node that change between user event. We define the document that the node has 17 attribute and assign 17 attribute has same weights. Collection are all the node on landscape XML files. We represent query and document by a term vector. Each term defines one dimension. We measure similarity by the cosine between the query vector and document vector in the vector space (normalized dot product). Thus, AQT can find the most similarity node between portrait and landscape XML files.

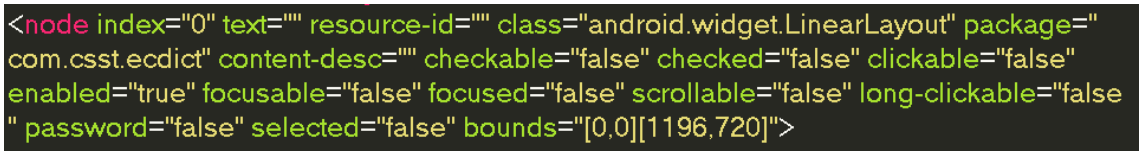

Figure 5.9 Attribute in xml files

Multiple display format handling test criteria:

1. Required for applications that support multiple display formats, on device with multiple display formats support.
2. The application should display correctly without obvious errors in all formats.

The processes of AQT multiple display format handling method of each AUT can be described as follows:

AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named

Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.

2. Install the APK file on the Android device.
3. Get portrait and landscape devices resolution
4. Using UI Automator dump app xml file. Android device in portraint and click a button to next state.
5. Analyze the information of the UI. Rotate the device. Assert portrait node and landscape node are same and assert portrait framelayout and landscape framelayout equal to OS initial display.

AQuA empirical evaluation:

1. Upload the report to database.

AQT dump the screenshot xml file and rotate screenshot automatically. Then, AQT generate report and record failures that the application display incorrectly without errors in all formats and switch incorrectly between the display orientations.

5.6.2 Differing screen sizes

Android devices come in many shapes and sizes all around the world. In order to be as successful on Android, application needs to adapt to various device configurations such as different languages, Android version and screen sizes. Android application designed to work on multiple devices, the application must be able to display correctly on differing screen sizes. Android applications are favored to applying multiple devices. Thus, if an Android application does not support to multiple devices, the application should be modified.

Tester test application differing screen size criteria process that install AUT in

multiple Android device, start to test the Application and Operate the Application on two devices with differing screen sizes include small, normal, large and extra-large. The tester need to observe the application display correctly on small, normal, large and extra-large Android devices.

Differing screen sizes test criteria:

1. Required for applications that support multiple devices
2. The application should display correctly on Android devices that screen sizes as small, normal, large and extra-large.
3. The application should display correctly without errors.
4. The application should use the whole of the screen area

The processes of AQT differing screen sizes method of each AUT can be described as follows:

AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.
2. Install the APK file in different screen size Android devices.
3. Get different Android devices resolution that Android OS will be set application initial display.
4. Using UI Automator dump the application screen xml file.
5. Simulate user events.

AQuA empirical evaluation:

1. Analyze the information of the UI. Check Android framelayout assert to application initial display resolution.

AQT parse the XML files and find the node that the “class” attribute is

“android.widget.FrameLayout”. AQT generate report and record failures that inability to display correctly on devices with different screen size.

5.7 Stability

5.7.1 Application stability

Android applications that must not crash or freeze at any time while running on the device. If the Android application crash or freeze during testing, the application should be modified. Tester test application stability criteria process that install AUT in Android device, start to test the application and observe the application behavior during the testing Figure 5.10 shows an example of the tester observe the application behavior and detect crash by manual. The tester need to focus on input user events and observe problem in lots of time.

Application stability test criteria:

1. Required for all applications.
2. The application must not freeze or exit unexpectedly at any time.

The processes of AQT application stability method of each AUT can be described as follows:

AQuA event sequence:

1. Analyze the APK file. The apk file contains a configuration file named Androidmanifest.xml. It contains the configuration of the app such as version of SDK, package name.
2. Install the APK file on Android devices.
3. Simulate user events

AQuA empirical sequence:

1. Analyze logcat and record Android runtime error.

AQT parse the XML files and select nodes that the “clickable” attribute is true to create user events. Use the procedure in the state to get the all legal actions and return the random choice from all legal actions. AQT generate report that record the event that trigger crashed and Android Runtime Error message for the user.

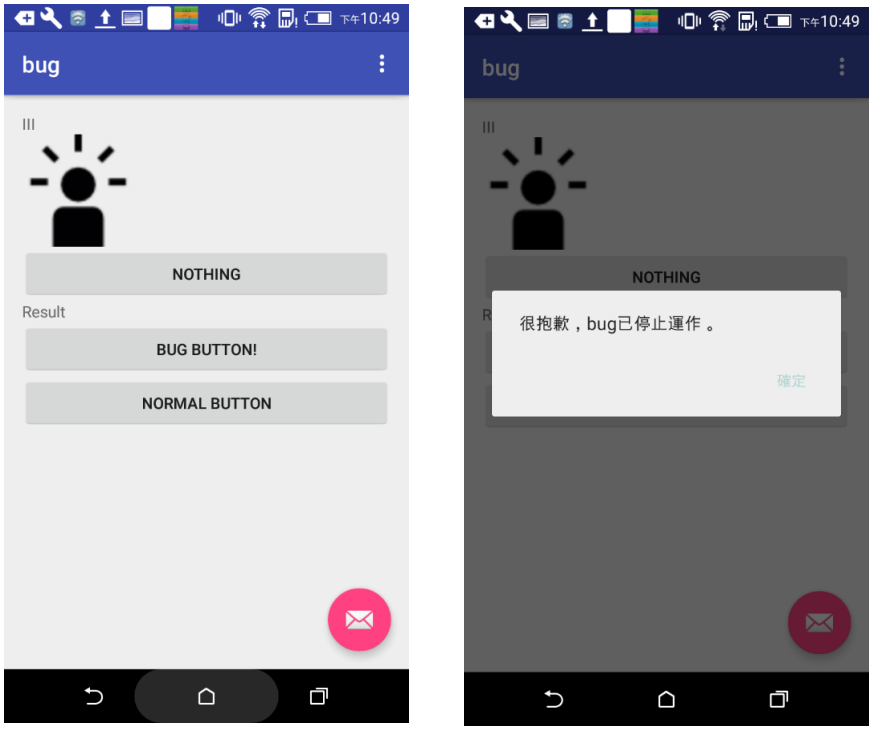

Figure 5.10 Application crash

Chapter 6 Experiments

6.1 Experiment environment

We program our tools and algorithm with Python 3.4.3 and use OpenCV 3.0 for computer vision technique. Our tools and techniques are developed based on framework for automated testing of Android Applications [24]. We executed MONVIS and AQT on Window 8.1 and control the device via adb. The specification of the laptop we used to run our experiments is shown in Table 6.1. The specification of the mobile phone is shown in Table 6.2.

Laptop	Lenovo
Operating system	Windows 8.1
CPU	Intel i7
RAM	8GB
Storage	1TB
Display Size	14” LED

Table 6.1 The specification of the Laptop

Mobile phone	hTC Desire 820s dual sim
Android version	5.0
CPU	1.7 GHz 八核心處理器
RAM	2GB RAM / 16GB ROM
Resolution	1280 x 720 pixels

Table 6.2 The specification of the mobile phone

6.2 Experiment design

In the chapter 6.3, we prove that MONVIS can find application crash and enhance line coverage successfully. We use the MonkeyXML and MonkeyCV test case generators to generate traces and use the crash evaluator to detect the crash. We find 11 applications that are the most popular on the google play store. The detailed application names and information are showed in Table 6.3. In the chapter 6.4, we prove that AQT can test application quality by AQuA test criteria. We find 6 applications that on the google play store. The detailed application names and information are showed in Table 6.4.

6.3 MONVIS

We select 11 applications that are shown in Table, there games, camera applications, tool applications, etc. We use ABCA to instrument SUT and generate coverage information. We measure SUT code coverage in 400 steps by MonketXML, MonkeyCV and manual.

Application name	Initial coverage (%)	MonkeyXML Coverage (%)	MonkeyCV Coverage (%)	Manual Coverage (%)	MonkeyCV - MonketXML (%)	comment
Tippy Tipper	19.697	28.084	29.004	30.623	0.92	
How Old do I Look ?	20.539	20.539	29.006	29.887	8.467	
嬰兒遊戲	38.568	38.568	38.568	39.56	0	unity

Peppa						
卡車司機 3D：Offroads	16.358	16.358	16.358	19.29	0	unity
1010!	6.739	25.286	34.339	34.75	9.053	unity
Magic Touch: Wizard for Hire	0.638	20.127	48.456	63.68	28.329	cocos 2dx
虫族獵人	7.274	9.374	9.5945	14.57	0.2205	unity
網路第四台	16.992	31.726	42.786	47.95	11.06	
綜合所得稅結算申報稅額試算	2.609	3.569	3.526	4.63	-0.043	
人臉年齡測試	30.848	35.276	43.026	46.35	7.75	
雙鐵時刻表	8.66	18.321	18.432	30.235	0.111	

Table 6.3 Application under test

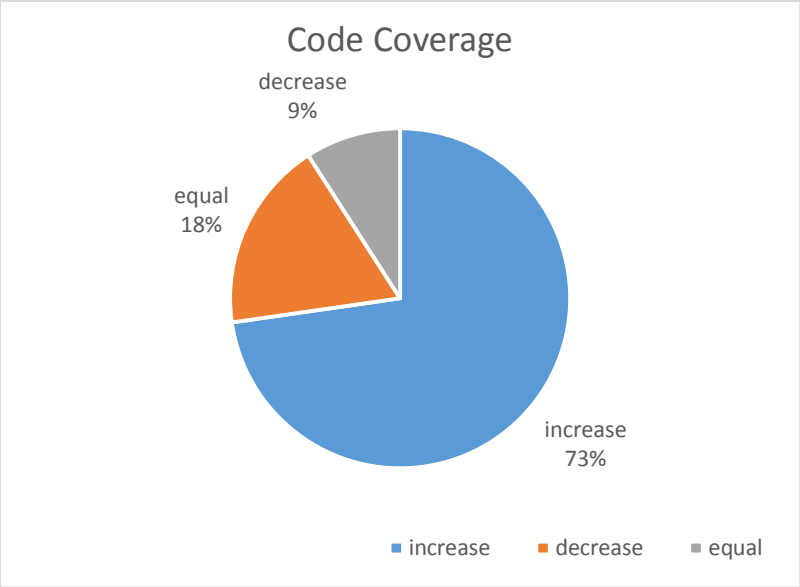

Figure 6.1 Test results of AUT code coverage

After calculating the coverage of MonkeyXML and MonkeyCV, we can provided that the MonkeyCV test case generator can get higher code coverage than MonkeyXML test case generator. The MonkeyXML test case generator cannot handle serveral external components such as WebView Unity and Cocos2d-x. On the other hand, MonkeyCV can find more button by computer vision technique. Figure 6.1 shows the test result of AUT code coverage. Figure 6.3 shows a legal action of MonketXML (blue line) and MonkeyCV (green line). We can observer MonkeyCV can find more button than MonkeyXML. Thus, MonkeyCV can get high code coverage than MonkeyCV

In order to prove that MONVIS can find an application bug successfully, we show the test results of the MonkeyXML and MonkeyCV test case generator. In the crash experiment, our benchmark is Bug. Bug has two different screens except the setting configuration. We insert a bug that caused java.lang.ArithmeticException on the screen. Thus, we use MONVIS to find the bug which we inserted into the Bug with MonkeyXML and MonkeyCV. Then, generate report about trace information and crash information. Figure 6.4 shows trace information and Android runtime error to the user.

We can observe that MonkeyXML and MonkeyCV find bug successfully. Moreover, MonkeyCV can get high code coverage. Thus, MonkeyCV is more efficient than MonkeyXML.

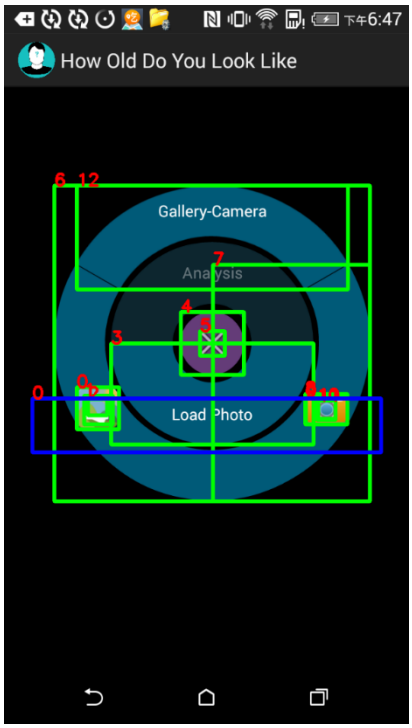

Figure 6.2 Legal action of MonketXML and MonkeyCV

```
---------- testcase 0 fail----------
adb -s HT53EYC00527 shell input tap 684 106
adb -s HT53EYC00527 shell input tap 360 710
adb -s HT53EYC00527 shell input tap 632 1096
adb -s HT53EYC00527 shell input tap 632 1096
adb -s HT53EYC00527 shell input tap 360 614
['E/AndroidRuntime(29970): FATAL EXCEPTION: main\r\n', 'E/AndroidRuntime(29970): Process: com.iii.bug,
PID: 29970\r\n', 'E/AndroidRuntime(29970): java.lang.ArithmeticException: divide by zero\r\n', 'E/
AndroidRuntime(29970): \tat com.iii.bug.MainActivity$3.onClick(MainActivity.java:81)\r\n', 'E/
```

Figure 6.3 MONVIS test report

6.4 AQT

We find 6 applications that on the google play store. Table 6.4 shows characteristics of our benchmark Android applications. SurPrice 智慧比價網 is an app allows users shop online for the best deals and lowest prices. Starbucks TW is an app provide online menu and store information. 麥當勞歡樂送 is an app that users order foods online. 英漢字典 EC Dictionary is a English and Chinese dictionary. MoMo 寵物記帳 is an app that users keep the account. Tippy Tipper is a tip calculator to calculate bill.

The results on from running AQT on the six Android applications show that as 6 tests fail out of a total 42 test case. Table 2 shows the failures and bug categories. We can observe SurPrice 智慧比價網, Starbucks TW and 麥當勞歡樂送 failed on connectivity test criteria. These applications do not provide error message to the user indicating an error with the connection. SurPrice 智慧比價網 failed on stability test criteria. This application crash during the testing. MoMo 寵物記帳 failed on memory use test criteria. When AQT suspend and re-launch the application, the application is handle incorrectly.

App	Rating	Downloads	Size	Publish Data
SurPrice 智慧比價網	4.6/5	1K-5K	2672KB	2016/5/24
Starbucks TW	3.9/5	1000K-5000K	7811KB	2016/3/30
麥當勞歡樂送	2.9/5	500K-1000K	13701KB	2016/4/19

英漢字典 EC Dictionary	4.2/5	1000K-5000K	20780KB	2016/6/20
MoMo 寵物記帳	4.0/5	10K-50K	4043KB	2014/6/26
Tippy Tipper	4.6/5	100K-500K	100KB	2013/12/16

Table 6.4 The applications used in the evaluation of AQT.

	Install and Launch	Memory Use	Connectivity		User Interface		Stability
App	UA	ML	NCND	NCAM	MDFH	DSS	AS
SurPrice 智慧比價網	Passed	Passed	Failed	Failed	Passed	Passed	Failed
Starbucks TW	Passed	Passed	Failed	Passed	Passed	Passed	Passed
麥當勞歡樂送	Passed	Passed	Passed	Failed	Passed	Passed	Passed.
英漢字典 EC Dictionary	Passed	Passed	Passed	Passed	Passed	Passed	Passed

MoMo 寵物記帳	Passed	Failed	Passed	Passed	Passed	Passed	Passed
Tippy Tipper	Passed	Passed	Passed	Passed	Passed	Passed	Passed

Table 6.5 Failures and bug categories

Chapter 7 Conclusion

7.1 Summary

We presented the black-box testing tool MONVIS and AQT. MONVIS is automated testing tool without knowledge of the expected behavior of the Android application. Thus, MONVIS test application correctness properties such as crash and exception. However, the application quality not only detect application crash but also check user interface, network connect, etc. AQT leverage the knowledge that common sense of how the application should response. Thus, MONVIS and AQT detect Android applications quality and help developers diagnose the report.

Tester verify the application behavior to meet its functional requirements and achieve a high quality. Tester can use manual method or write test scripts for testing. However, manual method can be time-consuming and writing test scripts cannot be automated modify test case. MONVIS and AQT provides the methods that tester do not need to manual and write test script by exploration the application and analysis the screenshot automatically.

Our experiment on Android applications demonstrates that find applications memory use, connectivity, user interface and crash problem. In addition, our result show that we can get high code coverage than analyze xml files by computer vision technique.

7.2 Limitation

There are four types of Android application components such as Activity, Services, Content providers and Broadcast receivers. However, MONVIS and AQT are only compatible with Activity. In order to analyze xml files, AQT is only suitable with applications that use UI elements from the Android UI framework.

7.3 Future Work

One of future works is trying to make MONVIS and AQT more powerful and efficiency. For now there are some problems could not be solved. MONVIS cannot extract WebView to words. If we can get word information, it possible to simulate events likes human. MONVIS and AQT use computer vision techniques to figure object in the screenshots. We can use another computer vision algorithms such as Scale-invariant feature transform and Optical Character Recognition to increase our tools.

MONVIS return the contours not appearing on XML that have the advantage of execution time. However, the contours have the probability that not be real clickable button for applications. We merge close contours and fully contained contours by neural network. The input of the neural network is histograms, coordinates, bounds, etc. The output of the neural network is two contours belong to the same group or not. We try to use the method to have low false positive rate.

AQT implements 7 test criteria in App Quality Alliance [23] announce testing criteria for Android applications. Thus, AQT can implements another test criteria to verify Android application's event handling, messaging & calls, Data handling and security.

REFERENCE

[1] "Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015." [Online]. Available: http://www.gartner.com/newsroom/id/3215217.

[2] "Android Fragmentation Visualized." [Online]. Available: http://opensignal.com/reports/2015/08/android-fragmentation/.

[3] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A GUI crawling-based technique for android mobile application testing," *Proc. 4th IEEE Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2011*, pp. 252–261, 2011.

[4] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and G. Imparato, "A toolset for GUI testing of Android applications," *IEEE Int. Conf. Softw. Maintenance, ICSM*, pp. 650–653, 2012.

[5] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using GUI Ripping for Automated Testing of Android Applications," *IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 258–261, 2012.

[6] Z. Liu, B. Liu, and X. Gao, "Test automation on mobile device," *Proc. 5th Work. Autom. Softw. Test AST 10*, no. 2007, pp. 1–7, 2010.

[7] N. Semenenko, M. Dumas, and T. Saar, "Browserbite: Accurate Cross-Browser Testing via Machine Learning over Image Features," *2013 IEEE Int. Conf. Softw. Maint.*, pp. 528–531, 2013.

[8] "UI Automator." [Online]. Available: https://developer.android.com/topic/libraries/testing-support-library/index.html.

[9] "Robotium." [Online]. Available: http://robotium.com/.

[10] C. H. Liu, C. Y. Lu, S. J. Cheng, K. Y. Chang, Y. C. Hsiao, and W. M. Chu, "Capture-replay testing for android applications," *Proc. 2014 Int. Symp. Comput.*

Consum. Control. IS3C 2014, pp. 1129–1132, 2014.

[11] "Monkey." [Online]. Available: https://developer.android.com/studio/test/monkey.html.

[12] "Monkeyrunner." [Online]. Available: https://developer.android.com/studio/test/monkeyrunner/index.html.

[13] "Google Cloud Test Lab." [Online]. Available: https://developers.google.com/cloud-test-lab/.

[14] "Appium." [Online]. Available: http://appium.io/.

[15] "Sikuli." [Online]. Available: http://www.sikuli.org/.

[16] W. L. Chang, *Vision-Based Technology for Automated App Testing*. National Taiwan University, 2015.

[17] "Android Application Fundamentals." [Online]. Available: https://developer.android.com/guide/components/fundamentals.html.

[18] J. W. Duran and S. C. Ntafos, "An Evaluation of Random Testing," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 4, pp. 438–444, 1984.

[19] K. Sen, "Effective random testing of concurrent programs," *22nd IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE 2007)*, pp. 323–332, 2007.

[20] D. Amalfitano, N. Amatucci, A. R. Fasolino, P. Tramontana, E. Kowalczyk, and A. M. Memon, "Exploiting the Saturation Effect in Automatic Random Testing of Android Applications," *Int. Conf. Softw. Eng.*, pp. 33–43, 2015.

[21] S. Huang, C. Yeh, and F. Wang, "ABCA : Android Black-box Coverage Analyzer of Mobile App Source Code," *Work. Autom. Verif. Anal. Synth.*, pp. 1–5, 2015.

[22] A. M. Memon, M. Lou Soffa, and M. E. Pollack, "Coverage criteria for GUI testing," *ACM SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, p. 256, 2001.

[23] "App Quality Alliance." [Online]. Available: http://www.appqualityalliance.org/.

[24] W. H. Chiang, *Experimet of a framework for automated testing of Android Applications*. National Taiwan University, 2015.

[25] P. A. and J. Offutt, *Introduction to Software Testing*. Cambridge University, 2008.

[26] "WebView." [Online]. Available: https://developer.android.com/reference/android/webkit/WebView.html.

[27] "Unity." [Online]. Available: http://unity3d.com/cn/.

[28] "COCOS2D-X." [Online]. Available: http://www.cocos2d-x.org/.

[29] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," *Int. Conf. Comput. Vis.*, pp. 839–846, 1998.

[30] C. Xiong, L. Chen, and Y. Pang, "An Adaptive Bilateral Filtering Algorithm and its Application in Edge Detection," *2010 Int. Conf. Meas. Technol. Mechatronics Autom.*, vol. 1, no. 1, pp. 440–443, 2010.

[31] J. Canny, "A computational approach to edge detection.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[32] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation, IEEE Trans," *Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, 2011.

[33] "Logcat." [Online]. Available: https://developer.android.com/studio/command-line/logcat.html.

[34] "Toast." [Online]. Available: https://developer.android.com/reference/android/widget/Toast.html.

[35] "Snackbar." [Online]. Available: https://developer.android.com/reference/android/support/design/widget/Snackbar.html.

[36] "Espresso." [Online]. Available:

https://developer.android.com/training/testing/ui-testing/espresso-testing.html.

[37] “Activity.” [Online]. Available: https://developer.android.com/reference/android/app/Activity.html.

[38] P. R. and H. S. Christopher D. Manning, *Introduction to Information Retrieval*. Cambridge University, 2008.