

國立臺灣大學電機資訊學院電機工程學研究所



碩士論文

Graduate Institute of Electrical Engineering
College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

基於 AQuA 之自動化安卓軟硬體互操作性測試
Automated Platform Interoperability Testing for Android
Application based on AQuA

陳昱成

Yu-Chen Chen

指導教授：王凡 博士

Advisor: Farn Wang, Ph.D.

中華民國 109 年 7 月

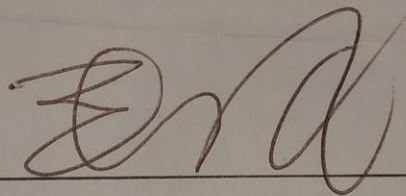
July, 2020

國立臺灣大學碩士學位論文
口試委員會審定書

基於 AQuA 之自動化安卓軟硬體互操作性測試
Automated Platform Interoperability Testing for Android
Application based on AQuA

本論文係陳昱成君（學號 R07921098）在國立臺灣大學電機工程學系完成之碩士學位論文，於民國 109 年 7 月 27 日承下列考試委員審查通過及口試及格，特此證明。

口試委員：



（簽名）

（指導教授）

王帛爵

張純明

李宏毅

戴欽權

雷欽隆

吳忠熾

系主任

（簽名）

誌謝



能完成這篇論文，我要特別感謝我的指導教授王凡老師，程式出問題時老師總是不厭其煩的幫助我解決瓢蟲，還有我的爸爸、媽媽、哥哥的支持，也要感謝實驗室的冠甫、宇謙、書維對我的支持與幫忙，謝謝大家。在論文有困難需要幫忙時互相討論，一起解決問題是撰寫論文時的一大樂趣。也感謝歷屆學長姐幫我們把路都鋪好了，讓我們可以乘坐在巨人的肩膀上寫程式。

中文摘要



在現在這個人人一台智慧型手機的時代，手機 APP 呈現爆炸性的成長，但對其好壞的判斷標準卻不多，目前我們看一個 APP 的好壞標準大概都是去商店看下面評價有幾星，這樣的作法略顯主觀，而其中一個組織 Application Quality Alliance，簡稱 AQuA，寫了一套評斷 APP 的標準，本篇論文把這套標準程式化、降低人工成本，盡量透過模擬使用者操作，達成黑箱測試來評斷測試 APP 有無符合 AQuA 所列之標準。本篇論文包含以下技術：

- I. 根據 AQuA 所制訂之標準，實現模擬測試者使用安卓手機操作硬體與系統，達到軟硬體與系統互操作性測試。
- II. 透過執行步驟與測試結果，測試者可取得相應資訊與潛在問題。

關鍵字：程式品質、自動化測試、安卓程式、黑箱測試

ABSTRACT



In this generation, everyone holds a smart phone and the applications have become numerous. There are not many standards can decide an application's quality. People usually judge application's quality through the play store's user responses. These responses may be a little bit subjective. Among the standards, there is an organization called AQuA (Application Quality Alliance). They wrote a suite of standard for android application. In this thesis, we transform these manual operations into programs, simulating people's behavior as possible and achieve black-box testing to evaluate the application's quality. This thesis includes following techniques:

- I. According to AQuA, we simulate the behaviors of hardware operating and system changing to achieve the interoperability testing of hardware and software.
- II. Testers can gain potential problems through testing steps and test reports.

Keywords : application quality 、 automated testing 、 Android application 、 black-box testing

CONTENTS



口試委員會審定書	#
誌謝	i
中文摘要	iii
ABSTRACT	iv
CONTENTS	v
LIST OF FIGURES	vii
LIST OF TABLES	viii
Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Purpose	2
1.3 Research Method	2
1.4 Organization	3
Chapter 2 Related Work.....	4
2.1 Stress Testing of Android Applications	4
2.2 Application Quality Testing.....	5
Chapter 3 Preliminaries	8
3.1 Android Automated Testing Framework	8
3.2 Android Application Structure.....	11
3.3 Android Debug Bridge	13
3.4 Test as a Dragon.....	14
Chapter 4 Testing Algorithms and Procedures	16
4.1 AQUA 3.1&3.2&3.3 HTTP and Network Connectivity	19

4.1.1	AQuA 3.1 HTTP Usage.....	19
4.1.2	AQuA 3.2&3.3 Network Connectivity.....	20
4.2	AQuA 3.4 Resource downloading	21
4.3	AQuA 4.7 Effects of timezone change	22
4.4	AQuA 6.1&6.2 Sdcard operation	23
4.5	AQuA 8.1 Language - Correct operation.....	24
4.6	AQuA 9.1&9.2&9.3 Suspend and resume.....	25
4.7	AQuA 9.5 Resource sharing	26
Chapter 5	Test Reports	27
5.1	AQuA 3.1 HTTP Usage & AQuA 3.2&3.3 Network Connectivity	28
5.2	AQuA 3.4 Resource downloading	30
5.3	AQuA 4.7 Effects of timezone change	31
5.4	AQuA 6.1&6.2 Sdcard operation	33
5.5	AQuA 8.1 Language - Correct operation.....	34
5.6	AQuA 9.1&9.2&9.3 Suspend and resume.....	37
5.7	AQuA 9.5 Resource sharing	38
Chapter 6	Conclusion	40
6.1	Summary.....	40
6.2	Future work.....	40
	Reference.....	41



LIST OF FIGURES



Fig. 3.4-1 issue tracker	15
Fig. 3.4-2 redmine	15
Fig. 4.1-1 AQuA 3.1&3.2&3.3 HTTP and Network Connectivity	19
Fig. 4.2-1 AQuA 3.4 Resource Downloading.....	21
Fig. 4.3-1 AQuA 4.6 Effects of daylight time change	22
Fig. 4.4-1 AQuA 6.1&6.2 Sdcard operation	23
Fig. 4.5-1 AQuA 8.1 Language – Correct operation	24
Fig. 4.6-1 AQuA 9.1&9.2&9.3 Suspend and resume	25
Fig. 4.7-1 AQuA 9.5 Resource sharing.....	26
Fig. 5.1-1 “OPEN POINT” crashed	29
Fig. 5.3-1 “斷食追蹤” crashed.....	32
Fig. 5.3-2 “OPEN POINT” crashed	32
Fig. 5.5-1 “極光清理” in English.....	35
Fig. 5.5-2 “極光清理” in Chinese (Simplified)	35
Fig. 5.5-3 “極光清理” in Chinese (Traditional).....	36
Fig. 5.7-1 “OPEN POINT” crashed	39

LIST OF TABLES



Table 2.2-1 The test cases we write	7
Table 5.1-1 Test report of AQuA 3.1&3.2&3.3	28
Table 5.2-1 Test report of AQuA 3.4	30
Table 5.3-1 Test report of AQuA 4.7	31
Table 5.4-1 Test report of AQuA 6.1&6.2	33
Table 5.5-1 Test report of AQuA 8.1	34
Table 5.6-1 Test report of AQuA 9.1&9.2&9.3	37
Table 5.7-1 Test report of AQuA 9.5	38

Chapter 1 Introduction



1.1 Motivation

Years ago, people used to browse website using computer or laptop. As smartphone took place dumbphone recently, websites and applications such as video streaming, games, second-hand car dealer, and lots of companies developed their own applications on smartphone. While using smartphone applications, an issue occurs that how to determine the quality of an application. Low quality may decrease the number of users and then lose profit. The quality testing issues above had two major challenges:

- I. High cost [1]: User interfaces differ from application to application in spite of slight difference between two system versions, even the same system version but two different devices. This kind of heterogeneity [2] results in failure of porting one testing script for this application or device to another application or device, and the testing script must be rewritten. Thus, many software test cases still rely on manual operation which cost significant and the test report can also be different from person to person.
- II. Lack of testing standards: up to what number of testing cases can you ensure the quality of application? Does the testing script gain sufficient confidence? This is the reason why we lack testing standards.




1.2 Purpose

As the challenges mentioned above and the rapidly growing number of application, world's companies observe the profit among it. They keep developing their own app quality specifications and criteria. And in this one, there is a non-profit global organization called AQuA (Application Quality Alliance) which headed by volunteers and knowledge contributors and holds the most influential. AQuA dedicates themselves to support the industry improving smartphone application quality, cooperating with business partner such as Motorola, Nokia, LG. So far, AQuA have announced several application quality criteria including memory usage, network connectivity, event handling, messaging and calls, etc. In Addition, each application quality criteria test distinctly defined test scope, test steps, and test results.

Till this moment in the whole world, the people who uses smartphone over who have cellphone is approximately 70% ~ 80%. Among them, the market share of Android is obviously more than iOS. Therefore, we developed an automated testing script for AQuA testing criteria on Android platform.

1.3 Research Method

In this thesis, we focused on the interoperation of the hardware and the software. We implemented black-box automated testing scripts on our platform, *TaaD* (Test as a Dragon) based on AQuA. When user wants to test an Android application,



what he or she needed to do is only to connect the device to the computer and click the AQuA button in TaaD. Once AQuA button be clicked, the testing scripts we designed based on AQuA will start from beginning to end. The procedure of AQuA button can be divide into three part: head, middle, and result. Each of the testing scripts we put in head phase will only execute once. The middle phase will take a number parameter from user, and go through each page until the number of pages we passed reach the number parameter. The testing scripts we placed in middle phase will execute in every page we got. After all, an AQuA testing report will be generated and will be uploaded to our server.

1.4 Organization

In Chapter 2 and 3, we introduced related work, frameworks, and some preliminaries we used. In Chapter 4, we showed the three part of AQuA button and presented the algorithm of the testing scripts we designed based on AQuA. In Chapter 5, we showed our testing results. In Chapter 6, we had a conclusion of our research and future wok.

Chapter 2 Related Work

2.1 Stress Testing of Android Applications



If we had done unit test and functionality test, then it will not be too much to do stress testing.

The easy way is hiring a monkey, allowing it to press anything it could touch casually. Under this circumstance, is our application still fine? Thus, Google's Android development team developed a command-line tool called Monkey. However, Monkey is more intelligent than the real monkey. You can command Monkey to trigger some events periodically or limit the button's proportion, etc. Monkey includes a number of options, but they can break down into four primary categories: Basic configuration options, Operational constraints, Event types and frequencies, Debugging options. Due to triggering random events, Monkey has vital limitations.

A. R. Fasolino, D. Amalfitano, and P. Tramontana et al. proposed an approach based on stress testing and regression testing [3][4][5]. They automatically built an application GUI model and generated executable test cases by a crawler.

The approaches mentioned above need no programming scripts. This is not only advantage but also disadvantage. Within the limitation of non-scripts, we can only detect crash result, furthermore cannot obtain the information of system and user experience. In comparison with our automated testing tool, we can either do monkey or

obtain the information of system and user experience.



2.2 Application Quality Testing

AQT (Application Quality Testing) is an automated black-box testing platform. It is designed for developing testing techniques to provide API specification for screenshot and management of the screen activity in Android device [6]. Making use of AQT, developers can send ADB commands through the platform to Android device and execute. The platform can perform installation, uninstallation, click, swipe, modification of the volume, making phone call, and obtain the information of logcat, CPU usage status, memory usage status, and XML layouts.

Every time AQT performing an action, the result will be save into an `ActionResult` list. Developers can check if the action successful or not by checking `ActionResult`. For example, when developers uninstall an application on device, there will be a "TRUE" message in the returned `ActionResult`. If the uninstallation fails, there would be a "FALSE" message in the returned `ActionResult`. On the other hand, when developers want to check the status of hardware, for example, GPS, the returned `ActionResult` will contain "TRUE" if the GPS was activated. The returned `ActionResult` will contain "FALSE" if the GPS is deactivated. In addition to `ActionResult`, AQT will give a problem detection in the beginning. Developers can take the report as a reference.



Surya Kant Josyula, and Daya Gupta implemented an application that connect their server. Through the communication between server, they will test the interoperability of the device's electrical components [7]. Cuixiong Hu, and Iulian Neamtii proposed a method to automated find bugs in GUI [8].

Liu et al. mentioned an automated testing method which according to both Capture and Replay approach for Android applications [9]. The method converted captured user interaction events and input arguments into test scripts and replayed by Robotium.

Lin, Wang, and Hsiao had implemented some AQuA test cases in AQT [10][11][12]. In this thesis and K. F. Chen, we completed the rest of AQuA test. This thesis focused on the interoperation of hardware and software, and K. F. Chen put an effort on user experience [13]. Table 2.3-1 listed the terms which we implemented, and the terms with * mean those are finished by K. F. Chen.



Table 2.2-1 The test cases we write

ID	Title	ID	Title
AQuA 3.1&3.2&3.3	Network connectivity	*AQuA 7.14	Spelling errors
AQuA 3.4	Resource downloading	AQuA 8.1	Language - Correct operation
AQuA 4.7	Effects of timezone changing	AQuA 9.1&9.2&9.3	Suspend and resume
AQuA 6.1&6.2	Sdcard operation	AQuA 9.5	Resource sharing
*AQuA 7.1	Readability	*AQuA 13.1&13.4	Scrolling in menus & Text field scrolling
*AQuA 7.2	Read time	*AQuA 13.2	Text field scrolling
*AQuA 7.5	Key layout ease of use	*AQuA 13.5	Multiple touch
*AQuA 7.8	Function progress		

Chapter 3 Preliminaries



We review some Android automated testing techniques and some Android automated testing frameworks. We will take a briefly look at how these techniques detect the bugs or unusual results based on stress testing, UI (user interface) test scripts, response capture, etc.

3.1 Android Automated Testing Framework

Appium is an open source automated testing tool. It can be roughly considering as a HTTP web server. It can manage a number of WebDriver sessions and has had their REST API opened. When collaborate with Selenium WebDriver API and specific client libraries, it can have the advance ability of crossing platform testing. In addition to supporting almost every programming language, it can execute on both Android and iOS.

Espresso Test Recorder is a testing script generation tool. It can establish your own UI testing scripts by recording your testing scenario without writing any line of program. You can also use it to add an assert into your application screenshot to test particular UI element.



Robotium is an open source Android automated UI black-box testing framework. It provides finding and assert API, and can simulate gesture operation on elements like click, long click, swipe, etc. With the support of it, test case developers can write function using JAVA and user acceptance testing scenarios, spanning multiple Android activities. On the other side, Robotium can collaborate with Maven, Gradle, and Ant, and can perform testing by your code or even non-code which just based on the APK. Robotium has an advance version called Robotium Recorder. Robotium Recorder is more powerful than Robotium. It is a pity that it needs pay, and not free of charge.

Sikuli is a pretty interesting automated UI testing tool. It started as an open source project originally by Taiwanese student in MIT (Massachusetts Institute of Technology), then it was taken by CU Boulder (University of Colorado Boulder) and released public. It uses real time pattern recognition of image powered by OpenCV to detect the trigger GUI components and send events for UI testing [14]. Detection of elements pop out or disappear, and click or swipe ain't too much for Sikuli to jam. Sikuli's UI comes quite friendly. It is basically the button you can click. After you clicked, Sikuli will do screenshot and inject your Jython code. The power of image recognition in Sikuli can not only use as a testing tool but also book tickets automatized for you.



MonkeyRunner is an API toolkit in Android SDK which supported by Google's Android development team. Programmers can write python script to simulate keyevent, click, swipe, etc. Once you write your script in advance, MonkeyRunner can complete a series of simulated action for you, achieving the purpose of automated testing. MonkeyRunner has three main modules: MonkeyRunner, MonkeyImage, MonkeyDevice. MonkeyRunner class provides the API to connect the device or emulator and is responsible for controlling the mission in your script. MonkeyImage class can do screenshot and then compare the similarity of two screenshots. MonkeyDevice class provides the API such as installing, uninstalling, opening activity, sending keyevent, and is mainly in charge of delivering commands to smartphone. The difference between MonkeyRunner and Monkey is Monkey does not support scripts and can only generate some random events.

UI Automator is a simple UI automated testing framework provided by Google's Android development team. Within Android testing, UI testing accounts for lots proportion. In tradition, people testing UI in labor had more bothering and boring. With the invention of UI Automator, it solved the potential error that might occur in traditional testing. It is more convenient to test different mission or different operating scenario using the framework.



Espresso is an automated UI testing framework developed by Google's Android development team. It is an automated UI testing tool and is mainly aimed at emulate user operations at a single app project. It provides synchronization testing and uses an independent UI thread to work. Espresso is suitable for white-box testing. In traditional testing, we often used sleep or retry to catch UI after UI refreshed. The advantage of Espresso is the synchronization of the UI thread. We do not need to write waiting code, predicting UI refreshed. It can automated detect whether main thread is idle or not, and execute the program we wrote. In other words, Espresso is dependent on Activity Life Cycle.

3.2 Android Application Structure

At present, people develops Android applications in Kotlin or Java mostly and the screen page in XML. The Android SDK (Software Development Kit) will include all of data and code into an APK (Android package). After all, APK can be installed on user's Android device. APK elements can be divided into four different types:

- I. Activities: A single activity represents a page of UI. An APK has many activities.

For example, when user wants to make a phone call, he/she will open the contact application. At the beginning of contact application, there will be a list of contacts.

This page is the main activity of the contact application. Then, user might tap



somebody which user wants to call and the screen will change into the page of detailed information. That page is another activity.

- II. Services: The components run in background. It is similar to the kernel service in OS. Service does not offer UI, and thus users will not be able to touch the service or notice it. For example, getting data through network or the interaction between activities.
- III. Content providers: One single content provider can manage one group of shared application data. Developers can save the data into file system, SQL, network, or anywhere they want. If another application wants to access or modify the data, it needs the permission of content provider.

Broadcast receivers: It is a component for broadcasting the notifications. Most notifications are broadcasted by system. For example, low battery, screen closing, etc. Applications can also broadcast, for example, notifying the specific resource finished downloading and can be used.

3.3 Android Debug Bridge



ADB (Android Debug Bridge) is a command-line tool included in the Android SDK Platform-Tools package. It provides access to an Android device like Unix shell, so that you can command the device. ADB executes like a client-server program that includes three components: client, `adb`, server. Client runs on developer's computer as a Unix shell and `adb` is a daemon process runs on Android device. Server is responsible for the communication between the client and the `adb`. We listed some ADB commands in common use below:

Check device connection:

```
adb devices
```

Dump current screen XML tree:

```
adb shell uiautomator dump && adb pull /sdcard/window_dump.xml .
```

Screenshot:

```
adb shell screencap -p /sdcard/screencap.png && adb pull /sdcard/screencap.png
```

Go Home page:

```
adb shell input keyevent 3
```

Go back:

```
adb shell input keyevent 4
```



3.4 Test as a Dragon

TaaD (Test as a Dragon) is an automated black-box testing platform we developed. It can test applications across three systems: Web, Android, and iOS. When developers open an application through TaaD, TaaD will convert the screen's UI into our IR. IR can tell developers which element on screen is clickable or text input field, so that developers can use IR for black-box testing. If an Android application has web or iOS version, their IRs are supposed to be same. Here are some arguments and APIs if developers want to write testing scripts:

- `currentStateDict`: A dictionary saves the current screen's IR.
- `automataDict`: A dictionary saves our `currentStateDict` from TaaD executed.
- `Reach`: If developers want to go to next page or some button, they can return `callAlgorithm` and `Reach` and the number of the button in TaaD. TaaD will bring the browser or device to another page.
- `queryCurrentStateIndex`: If developers want to do some action through our IR, they need to get current screen's IR first. The way they get current IR is to call `queryCurrentStateIndex` to get `currentStateIndex` and the current IR will be in `automataDict[currentStateIndex]`.



Fig. 3.4-1 issue tracker

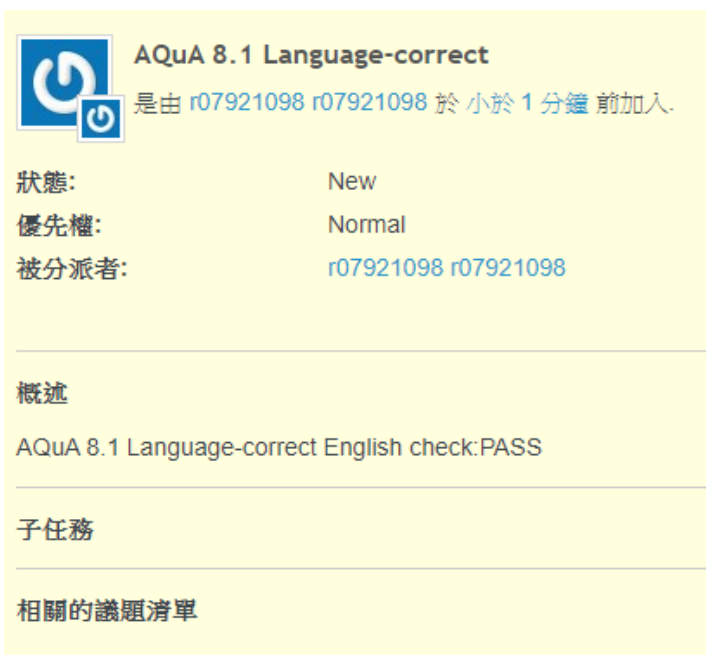


Fig. 3.4-2 redmine

Fig. 3.4-1 and Fig. 3.4-2 is the email issue tracker and redmine we will send when generate test reports.

Chapter 4 Testing Algorithms and Procedures



First, we implemented APIs for others who also wants to write Android AUT (Application Under Test) scripts on TaaD. Then, we'll present our testing algorithms based on AQuA test cases. Here are APIs we implemented:

- `check_app_in_background.check(finding_string):`

This API can help developers checking if the application is still work in background or not. When calling, developers need to give the application's package name as `finding_string` argument.

- `contact.accessing():`

This API modifies the contacts on device. Developers must be careful when using this API because after calling it, contacts will be clear.

- `install_sdcard_check.can_install_to_sdcard(apk_path):`

This API can tell developers where an APK can be installed. When calling, developers need to give the APK path in computer as `apk_path` argument. In APK's `manifest.xml`, `installLocation(0x010102b7)` records a number. 0 and 2 represents that the APK can install to both internal storage and sdcard, and the default is 0. 1 represents that the APK is not allowed to install to sdcard. This API will return True if the number is 0 or 2, else False.

- `check_installation.check(package_name):`



This API can help developers checking if the application installed successfully. When calling, developers need to give the application's package name as `package_name` argument.

- `listening_logcat.runProcess(finding_string):`

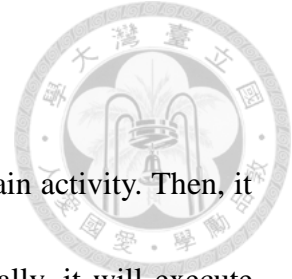
This API can tell developers if specific event had had triggered from logcat. We used it for listening HTTP request events. When calling, developers need to give the event keyword which they want to find as `finding_string` argument. It will return True if found it in logcat, else False.

- `multitouch_check .check(apk_path):`

Multitouch has two explanations. One is touching same position sequentially. Another is touching two positions simultaneously This API can tell developers if an application accepts touching two positions simultaneously. When calling, developers need to give the APK path in computer as `apk_path` argument. It will return True if XML file accepts multitouch, else False.

- `ping_google.ping():`

This API can tell developers if the network of the device is connecting. Nowadays, how do people check the Internet connection? The answer is open your command line or terminal, and `ping 8.8.8.8`, which is Google's DNS server IP. We used this method and ping Google from our device. It will return True if ping successful, else False.



- `take_photo()`:

This API can divide into three part. First, it will open the camera main activity. Then, it will execute eventkey 27, which is take photo from rear lens. Finally, it will execute eventkey 4, which is go back and save the photo.

- `Uninstallation.uninstall(package_name)`:

Before, we only had checking uninstallation API related to uninstallation. Now, this API can help developers uninstalling the application if they want. When calling, developers need to give the application's package name as `package_name` argument. It will return True if remove successful, else False. If the returned value is False, then probably the application is not installed on the device at the beginning.

- `battery.change(percent)`:

This API can fake your device battery percent. Give an integer in 1~100 as argument.

- `broadcast.broadcast()`:

This API can send broadcast to specific package. Default we send to `com.android.test`.

- `gps.access()`:

This API can tell developers if the GPS of the device is on. It will return True if gps component is on work, else False.



4.1 AQuA 3.1&3.2&3.3 HTTP and Network Connectivity

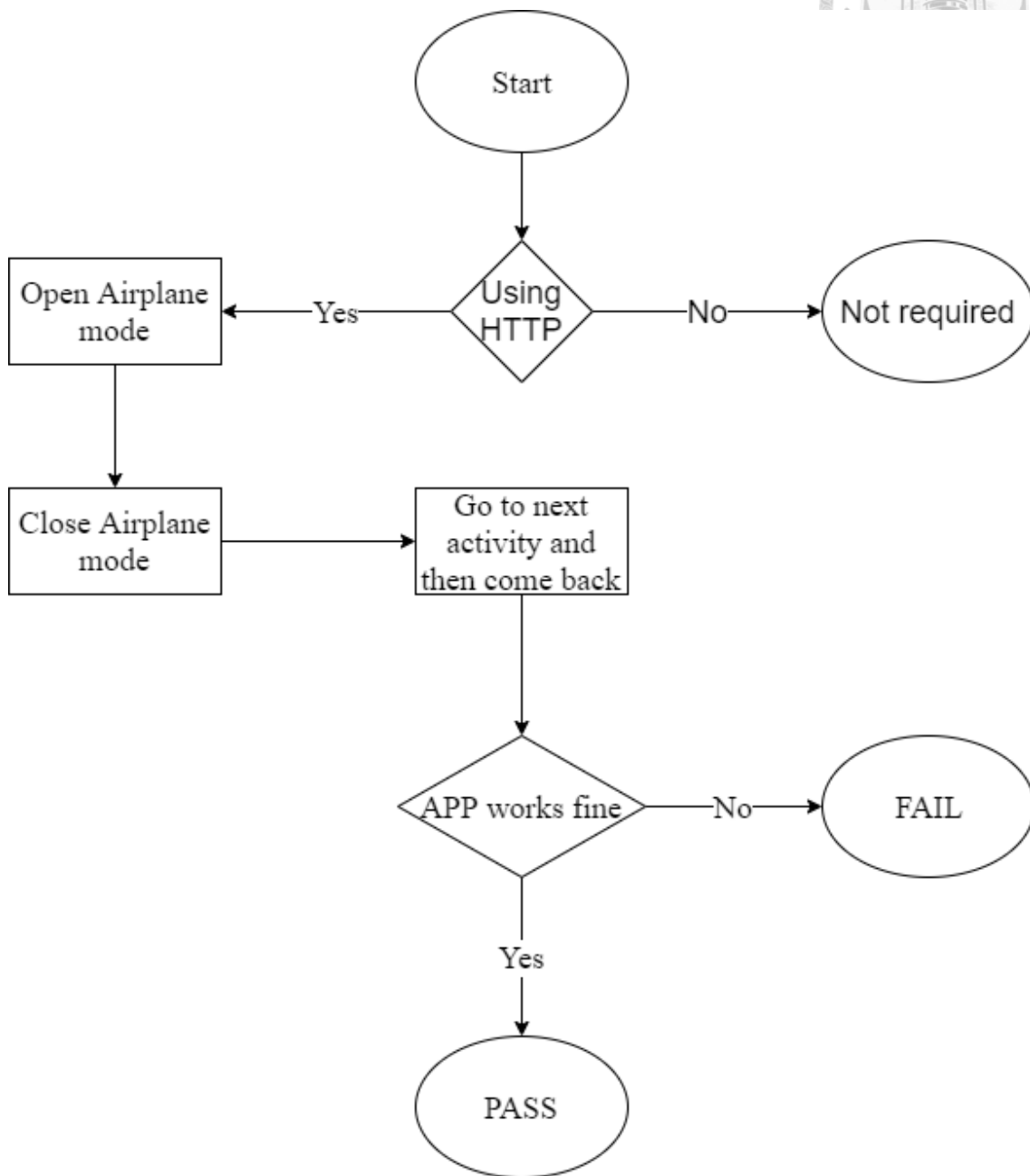


Fig. 4.1-1 AQuA 3.1&3.2&3.3 HTTP and Network Connectivity

4.1.1 AQuA 3.1 HTTP Usage

Before testing AQuA 3.2 and 3.3. We will check if current activity uses http. Our HTTP

API will tell us the result. We will put this result and the result of AQuA 3.2 and 3.3

together at the test report section. The diagram is shown in Fig. 4.1-1.



4.1.2 AQuA 3.2&3.3 Network Connectivity

As we show in Fig. 4.1-1, first, we open airplane mode to disconnect the network. Then, we will turn it back and check whether the application still works fine. If the activity cannot pass all the checkpoint, the test result is FAIL, else PASS.

4.2 AQuA 3.4 Resource downloading

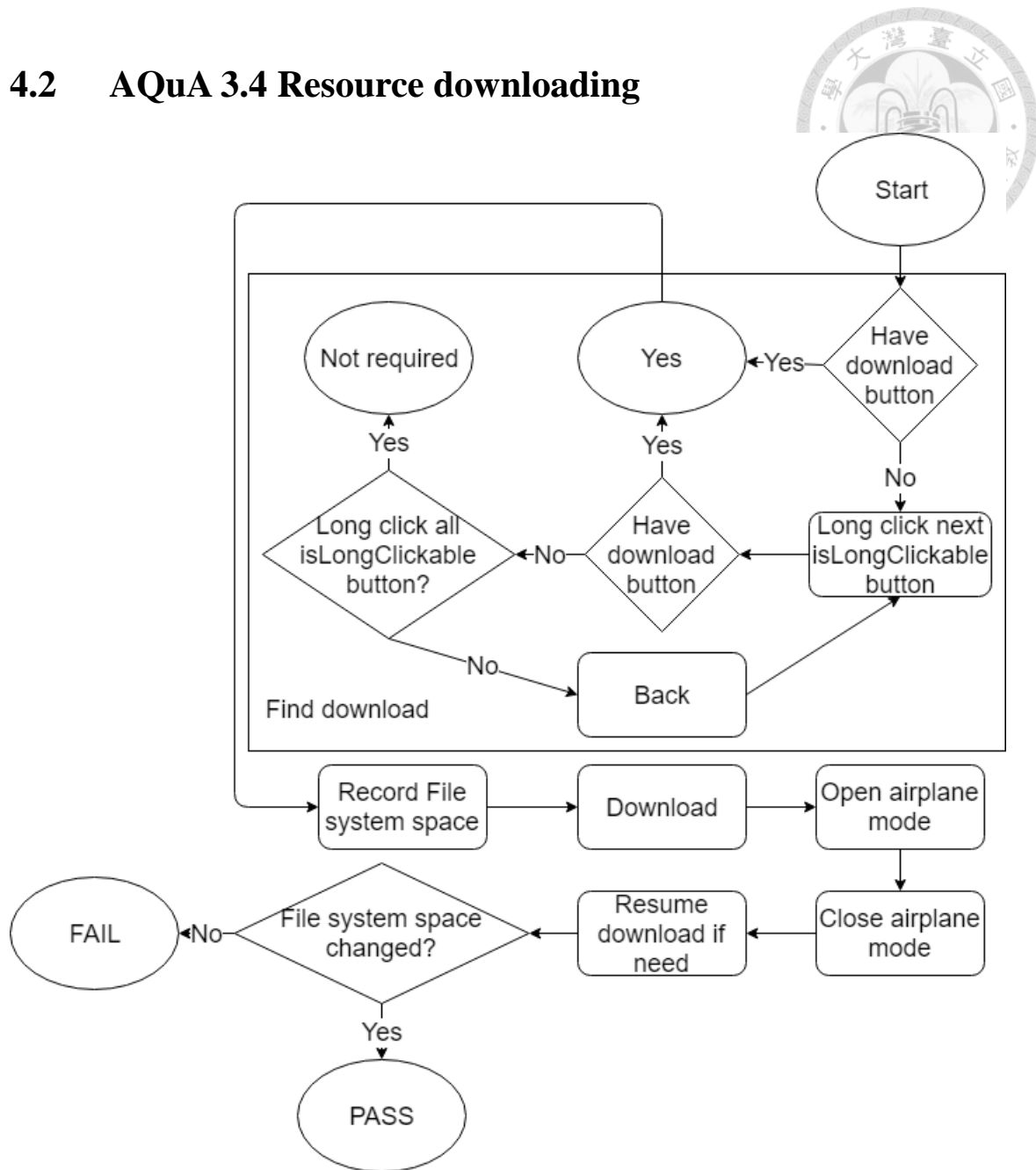


Fig. 4.2-1 AQuA 3.4 Resource Downloading

The diagram we shown in Fig. 4.2-1 is AQuA 3.4. When we entering an activity, we will loop every button and isLongClickable button. Every time we press a button we will check if there is button which text contain “save” or “download” or “下載” or “儲存”. If the result is Yes, we will download it.



4.3 AQuA 4.7 Effects of timezone change

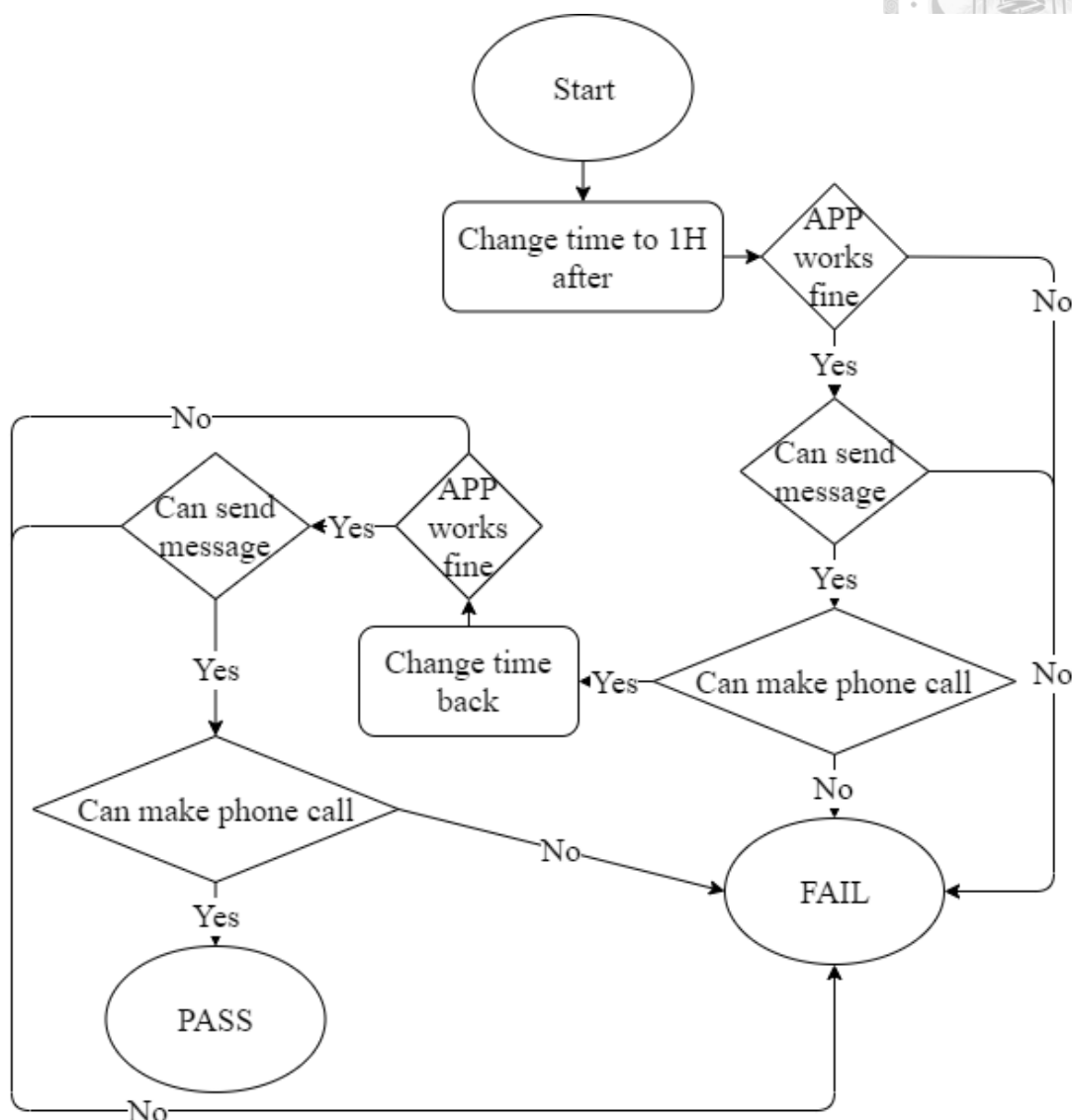


Fig. 4.3-1 AQuA 4.6 Effects of daylight time change

In AQuA 4.7, our purpose is to test if the time change of the device will affect the others components and the diagram is shown in Fig. 4.3-1. First, we will change our device time to the future 1 hours and see if the network and phone call work fine. Then, we will change back the time and check again. If everything works fine, the test report is PASS.

If any of them fails, it is FAIL.



4.4 AQuA 6.1&6.2 Sdcard operation

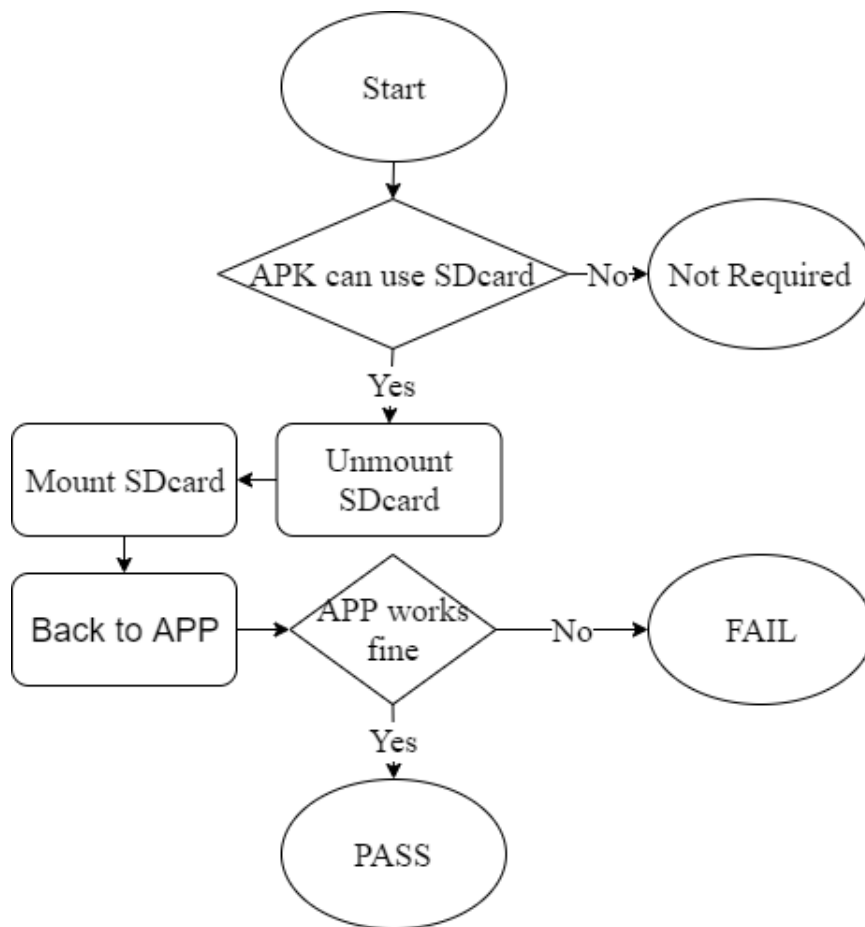


Fig. 4.4-1 AQuA 6.1&6.2 Sdcard operation

The diagram we combined AQuA 6.1 with AQuA 6.2 is shown in Fig. 4.4-1. What AQuA 6.1 wants to know is whether the APK can use sdcard. If the APK specify cannot use sdcard, it is not required. If yes, we will try rmmount sdcard and run monkey to check if the application is still work.

4.5 AQuA 8.1 Language - Correct operation

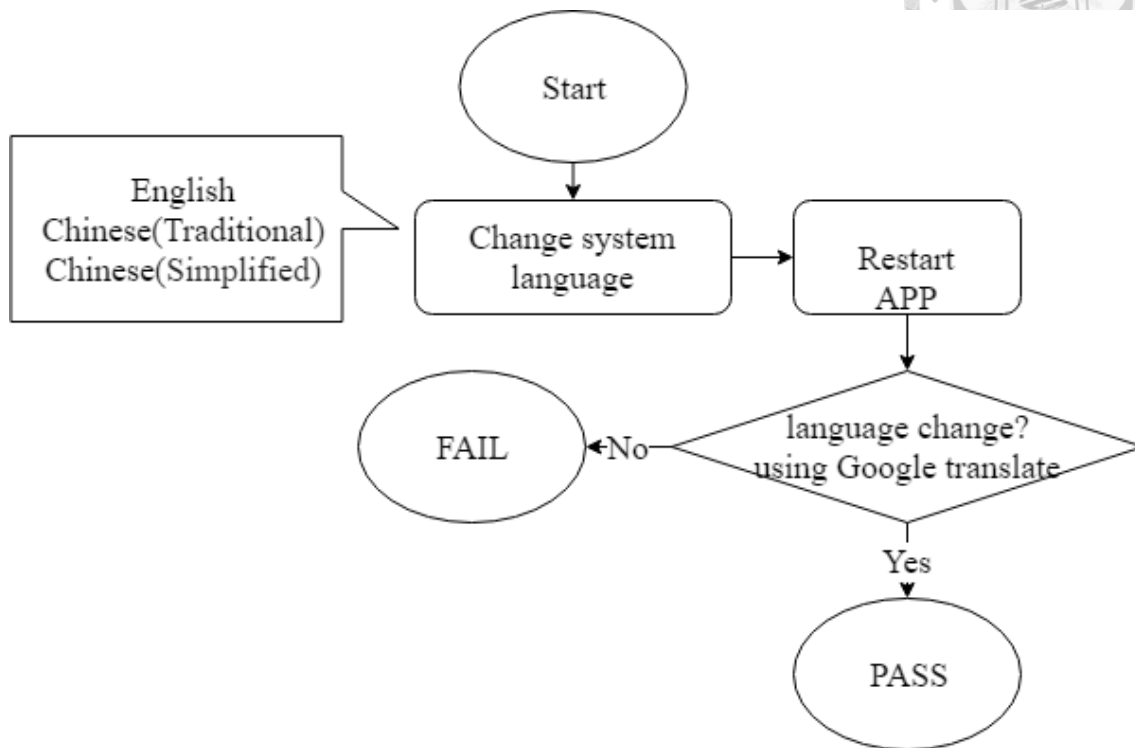


Fig. 4.5-1 AQuA 8.1 Language – Correct operation

In AQuA 8.1, we want to check if we change the system language, the application language will change or not. Thus, we change device language first. Then, we will close and restart the application. If the application's language does not change, the test report is FAIL, else PASS. The diagram is shown in Fig. 4.5-1.

4.6 AQuA 9.1&9.2&9.3 Suspend and resume

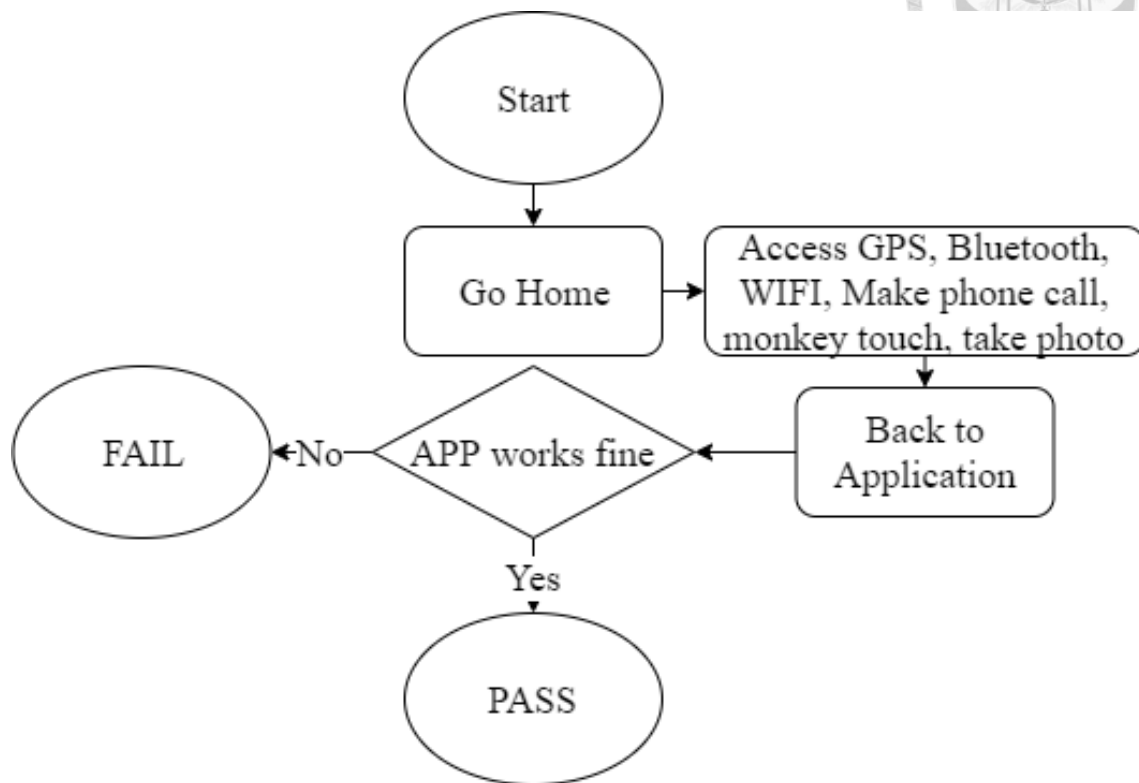


Fig. 4.6-1 AQuA 9.1&9.2&9.3 Suspend and resume

We combined AQuA 9.1, 9.2, and 9.3 in Fig. 4.6-1. Here, we want to check all of the hardware listed in AQuA. When we first open an application, we will go Home page and try accessing GPS, Bluetooth, WIFI, camera, touching screen, and phone calling. After, we go back the application and check if the application works fine.



4.7 AQuA 9.5 Resource sharing

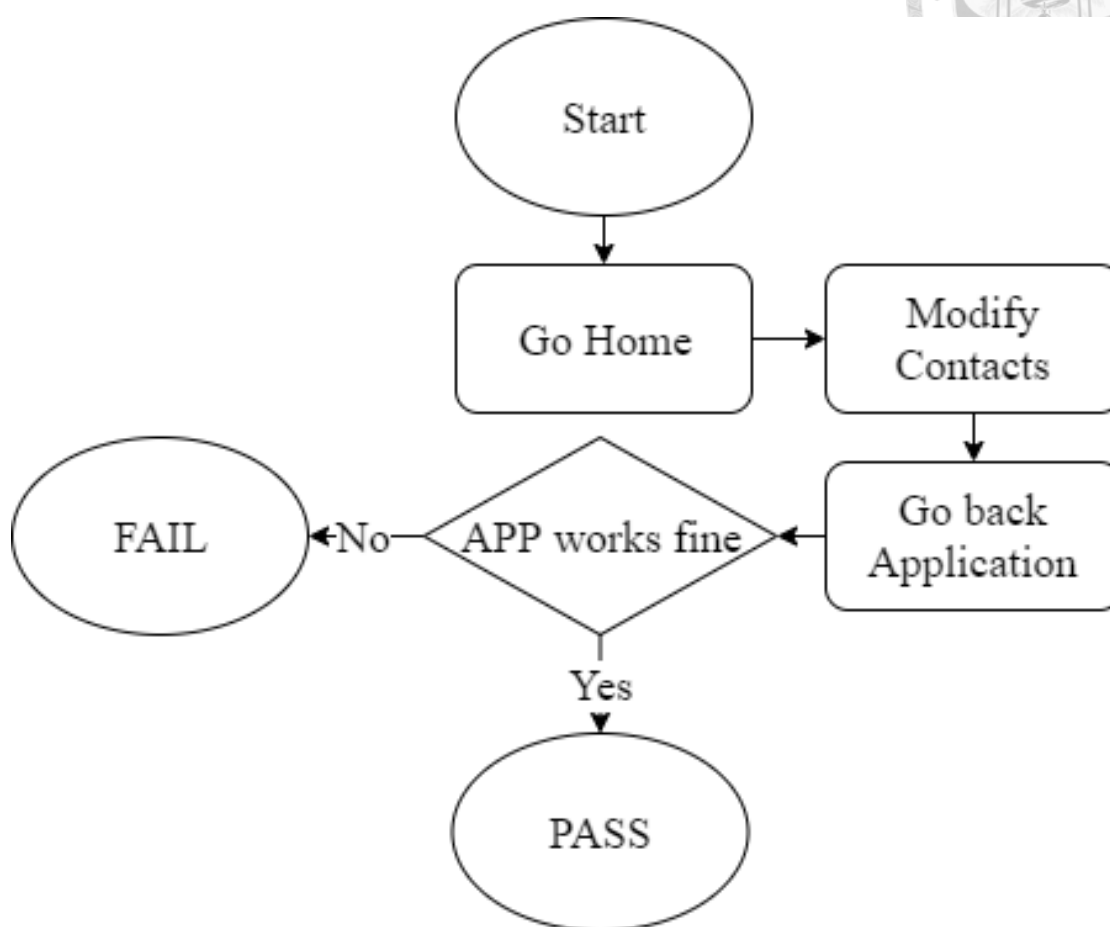


Fig. 4.7-1 AQuA 9.5 Resource sharing

Fig. 4.7-1 represents AQuA 9.5. In AQuA 9.5 they defined the shared database as contact book. When we get in an activity, we will go Home page and try modify and delete the contacts on the device. After accessing the contacts, we go back to the application and check if it is still work.

Chapter 5 Test Reports



Here, we show our AQuA test reports in this section. The devices we use is

Android emulator Nexus 6P API 23 (4G RAM / 10G ROM, Android version is 7.1.1).

The applications we test are “神盾測速照相”， “歡歌-免費在線K歌，全民音樂交

友必備軟體”， “極光清理 2020 — 殺毒、加速、清理、應用鎖”， “全民健保

行動快易通 | 健康存摺”， “間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，

禁食，減肥”， “Akinator”， “接龍”， “郵保鑣”， “Hi-Life VIP”， “OPEN

POINT：消費累點 回饋優惠”。 First 7 of them have 4~5 star rate on google play. 郵

保鑣, Hi-Life VIP have only 3 star on google play. The last one has only 2 star on

google play. Users can find these application under SUT folder.

5.1 AQuA 3.1 HTTP Usage & AQuA 3.2&3.3 Network Connectivity



Table 5.1-1 Test report of AQuA 3.1&3.2&3.3

APP	Network Connectivity
神盾測速照相	PASS
歡歌-免費在線K歌，全民音樂交友必備軟體	PASS
極光清理 2020 — 殺毒、加速、清理、應用鎖	PASS
全民健保行動快易通 健康存摺	PASS
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	PASS
Akinator	PASS
接龍	Not required
郵保鑣	PASS
Hi-Life VIP	PASS
OPEN POINT：消費累點 回饋優惠	FAIL

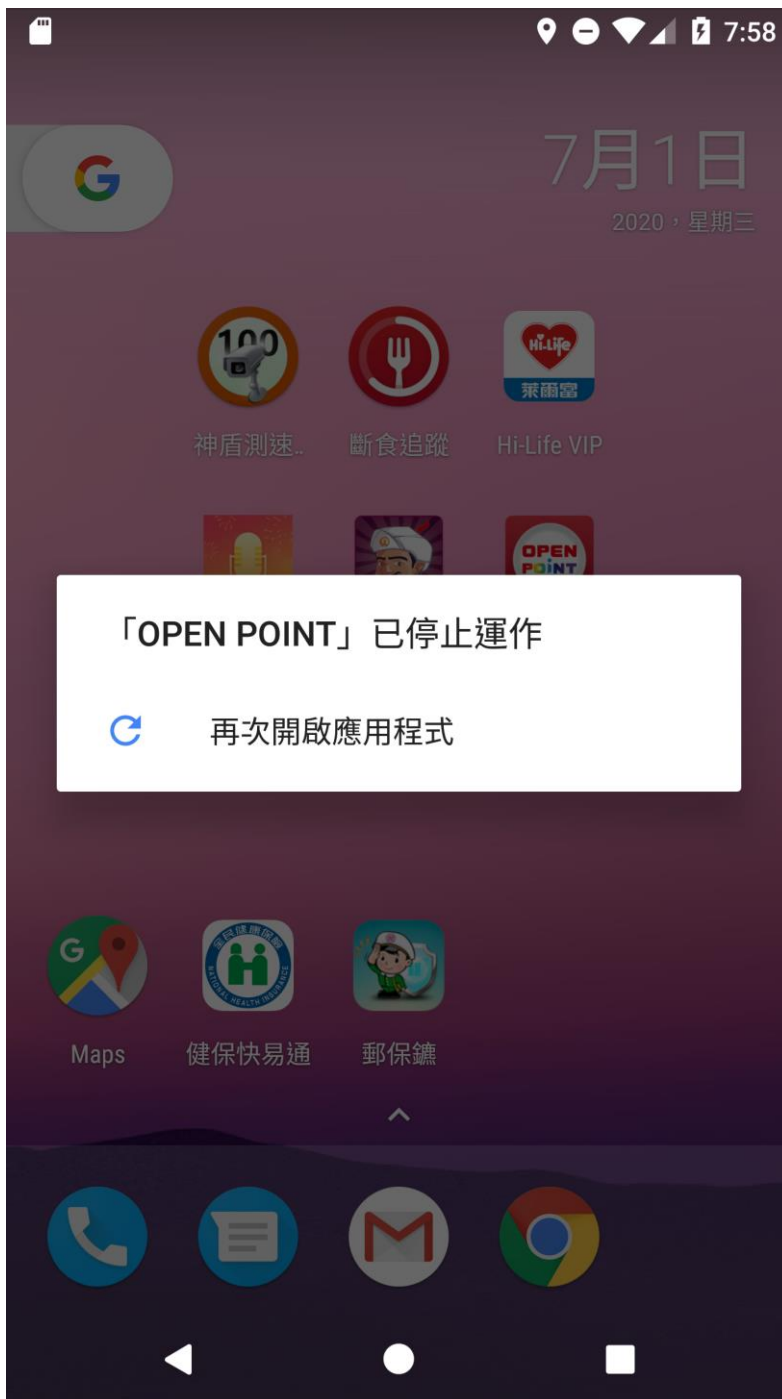


Fig. 5.1-1 “OPEN POINT” crashed

The test report of AQUA 3.1&3.2&3.3 is shown in Table 5.1-1. Fig. 5.1-1 is the failure result of application, “OPEN POINT：消費累點 回饋優惠” . The application crashed when we check the application working fine.

5.2 AQuA 3.4 Resource downloading



Table 5.2-1 Test report of AQuA 3.4

APP	resource downloading
神盾測速照相	Not required
歡歌-免費在線K歌，全民音樂交友必備軟體	Not required
極光清理 2020 — 殺毒、加速、清理、應用鎖	Not required
全民健保行動快易通 健康存摺	Not required
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	Not required
Akinator	Not required
接龍	Not required
郵保鑣	Not required
Hi-Life VIP	Not required
OPEN POINT：消費累點 回饋優惠	Not required
Messenger	PASS

Table 5.2-1 shows the test report of AQuA 3.4. The results may be predictable because seldom applications need to download resources additionally.

5.3 AQuA 4.7 Effects of timezone change



Table 5.3-1 Test report of AQuA 4.7

APP	Effects of timezone change
神盾測速照相	PASS
歡歌-免費在線K歌，全民音樂交友必備軟體	PASS
極光清理 2020 — 殺毒、加速、清理、應用鎖	PASS
全民健保行動快易通 健康存摺	PASS
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	FAIL
Akinator	PASS
接龍	PASS
郵保鑰	PASS
Hi-Life VIP	PASS
OPEN POINT：消費累點 回饋優惠	FAIL



Fig. 5.3-1 “斷食追蹤” crashed

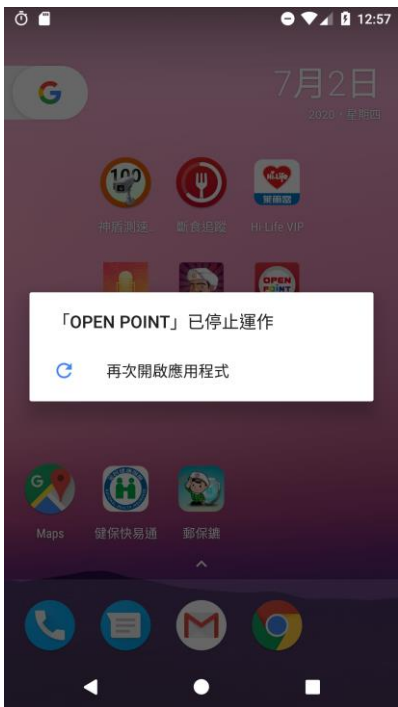


Fig. 5.3-2 “OPEN POINT” crashed

Fig. 5.3-1 and Fig. 5.3-2 are the failure event in Table 5.3-1 when we check application working fine.

5.4 AQuA 6.1&6.2 Sdcard operation

Table 5.4-1 Test report of AQuA 6.1&6.2



APP	Sdcard operation
神盾測速照相	Not required
歡歌-免費在線 K 歌，全民音樂交友必備軟體	Not required
極光清理 2020 — 殺毒、加速、清理、應用鎖	Not required
全民健保行動快易通 健康存摺	Not required
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	Not required
Akinator	PASS
接龍	PASS
郵保鑣	Not required
Hi-Life VIP	Not required
OPEN POINT：消費累點 回饋優惠	Not required

Table 5.4-1 shows the test report of sdcard operation. “Akinator” and “接龍” can run successfully after remount the sdcard. Others APK file does not allow to use sdcard.

5.5 AQuA 8.1 Language - Correct operation



Table 5.5-1 Test report of AQuA 8.1

APP	Chinese (Traditional)	Chinese (Simplified)	English
神盾測速照相	PASS	FAIL	FAIL
歡歌-免費在線K歌，全民音樂交友必備軟體	PASS	FAIL	FAIL
極光清理 2020 — 殺毒、加速、清理、應用鎖	PASS	PASS	PASS
全民健保行動快易通 健康存摺	PASS	FAIL	FAIL
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器， 禁食，減肥	PASS	PASS	PASS
Akinator	FAIL	FAIL	PASS
接龍	FAIL	FAIL	PASS
郵保鑣	PASS	FAIL	FAIL
Hi-Life VIP	PASS	FAIL	FAIL
OPEN POINT：消費累點 回饋優惠	PASS	FAIL	FAIL

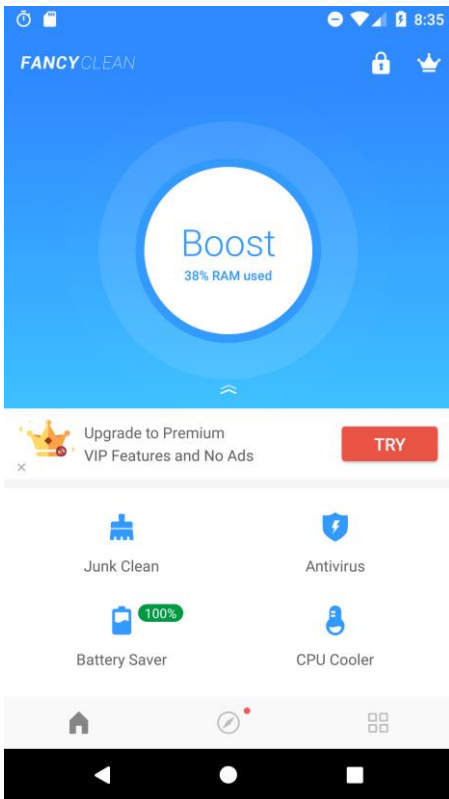


Fig. 5.5-1 “極光清理” in English



Fig. 5.5-2 “極光清理” in Chinese (Simplified)



Fig. 5.5-3 “極光清理” in Chinese (Traditional)

Table 5.5-1 shows the language list after we changed system language. Fig. 5.5-1, Fig. 5.5-2, and Fig. 5.5-3 are the screenshot of all-PASS application, “極光清理 2020 — 殺毒、加速、清理、應用鎖”.

5.6 AQuA 9.1&9.2&9.3 Suspend and resume



Table 5.6-1 Test report of AQuA 9.1&9.2&9.3

APP	Suspend and resume
神盾測速照相	PASS
歡歌-免費在線 K 歌，全民音樂交友必備軟體	PASS
極光清理 2020 — 殺毒、加速、清理、應用鎖	PASS
全民健保行動快易通 健康存摺	PASS
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	FAIL
Akinator	PASS
接龍	PASS
郵保鑣	PASS
Hi-Life VIP	PASS
OPEN POINT：消費累點 回饋優惠	FAIL

Table 5.6-1 shows the results after we access the hardware list in AQuA. “間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥” and “OPEN POINT：消費累點 回饋優惠” failed after we accessed the hardware and go back the application.

5.7 AQuA 9.5 Resource sharing

Table 5.7-1 Test report of AQuA 9.5



APP	Resource sharing
神盾測速照相	PASS
歡歌-免費在線K歌，全民音樂交友必備軟體	PASS
極光清理 2020 — 殺毒、加速、清理、應用鎖	PASS
全民健保行動快易通 健康存摺	PASS
間歇性斷食 - 零卡路里斷食追蹤，斷食計時器，禁食，減肥	PASS
Akinator	PASS
接龍	PASS
郵保鑣	PASS
Hi-Life VIP	PASS
OPEN POINT：消費累點 回饋優惠	FAIL

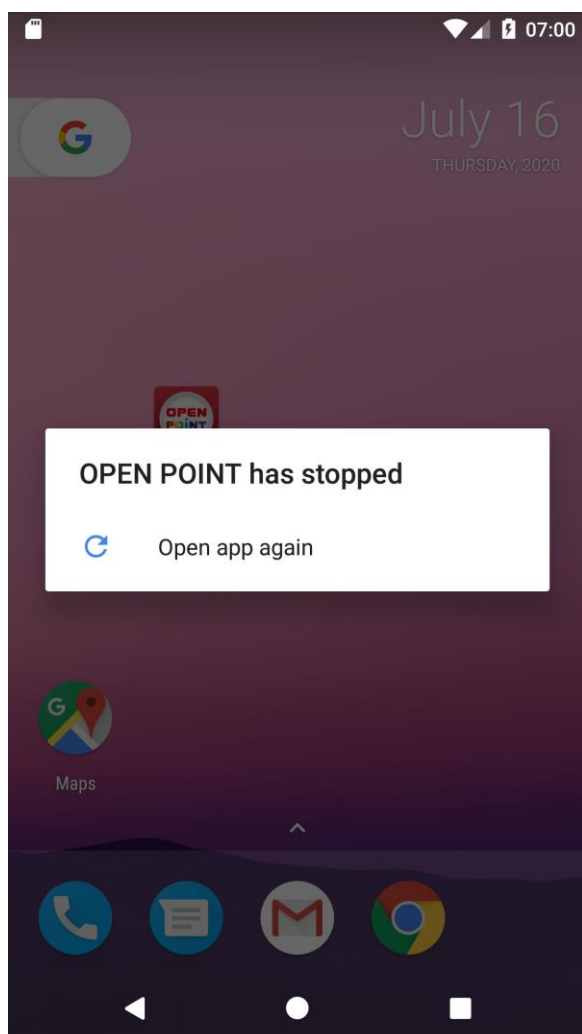


Fig. 5.7-1 “OPEN POINT” crashed

Table 5.7-1 shows the results after we modify the contact database. Fig. 5.7-1 is the failure event of “OPEN POINT：消費累點 回饋優惠”. It has the similar behavior with the result we access hardware.



Chapter 6 Conclusion

6.1 Summary

The test cases above are the part of interoperability of hardware and software in AQuA. We implement it for testers who want to test their application's quality. Testers can get test reports and check the potential problems of the application.

We transform the test cases from manual into automated testing. As long as user download our AQuA project from TaaD website, he or she can perform his/her own AQuA testing on any android application without write any other codes.


6.2 Future work

Recently, TaaD can perform testing on iOS system. At the future, there will be an AQuA – like version for iOS version. iOS system is a little different to Android system, such as sdcard. iOS cannot plug in external sdcard. Thus, we cannot directly transplant the original AQuA into iOS version.

Reference



- [1] A. Seesing and A. Orso, “InsECTJ: a generic instrumentation framework for collecting dynamic information within Eclipse,” in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, 2005, pp. 45-49: ACM.
- [2] Chawla and A. Orso, “A generic instrumentation framework for collecting dynamic information,” The ACM/SIGSOFT International Symposium on Software Testing and Analysis, In Online Proc. of the ISSTA Workshop on Empirical Research in Software Testing, vol. 29, no. 5, pp. 1-4, 2004.
- [3] R. Fasolino, D. Amalfitano, and P. Tramontana, “A gui crawling-based technique for android mobile application testing,” in *2011 IEEE fourth international conference on software testing, verification and validation workshops*, 2011, pp. 252-261: IEEE.
- [4] R. Fasolino, D. Amalfitano, G. Imperato, P. Tramontana, and S. De Carmine, “A toolset for GUI testing of Android applications,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 650-653: IEEE.
- [5] M. Memon, A. R. Fasolino, D. Amalfitano, P. Tramontana, and S. De Carmine, “Using GUI ripping for automated testing of Android applications,” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 258-261: ACM.

- 
- [6] W. H. Chiang, “Experiment of a framework for automated testing of Android Application,” Master Thesis, Electrical Engineering, National Taiwan University, 2015.
- [7] S. K. Josyula, D. Gupta, “Internet of things and cloud interoperability application based on Android” in *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*.
- [8] C. Hu, I. Neamtiu, “Automating GUI testing for Android applications” in *Proceedings of the 6th International Workshop on Automation of Software Test*.
- [9] 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,” in *2014 International Symposium on Computer, Consumer and Control*, 2014, pp. 1129-1132: IEEE.
- [10] H. Lin, “Automated Testing for Quality of Android Applications,” Master Thesis, Electrical Engineering, National Taiwan University, 2016.
- [11] G. Q. Wang, “Automated Testing for Quality Android Applications,” Master Thesis, Electrical Engineering, National Taiwan University, 2017.
- [12] T. Hsiao, “Automated AQuA Testing for Android Applications,” Master Thesis, Electrical Engineering, National Taiwan University, 2019.
- [13] K. F. Chen, “” Master Thesis, Electrical Engineering, National Taiwan University, 2020.

- [14] T. Yeh, T.-H. Chang, and R. C. Miller, “Sikuli: using GUI screenshots for search and automation,” in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 183-192: ACM.

