Automated Configuration Testing and Coverage Analysis for Android Apps
安卓應用程式之自動化組態測試與涵蓋率分析

學生：黃上誼（電子所EDA組）
指導教授：王凡 教授

Outline

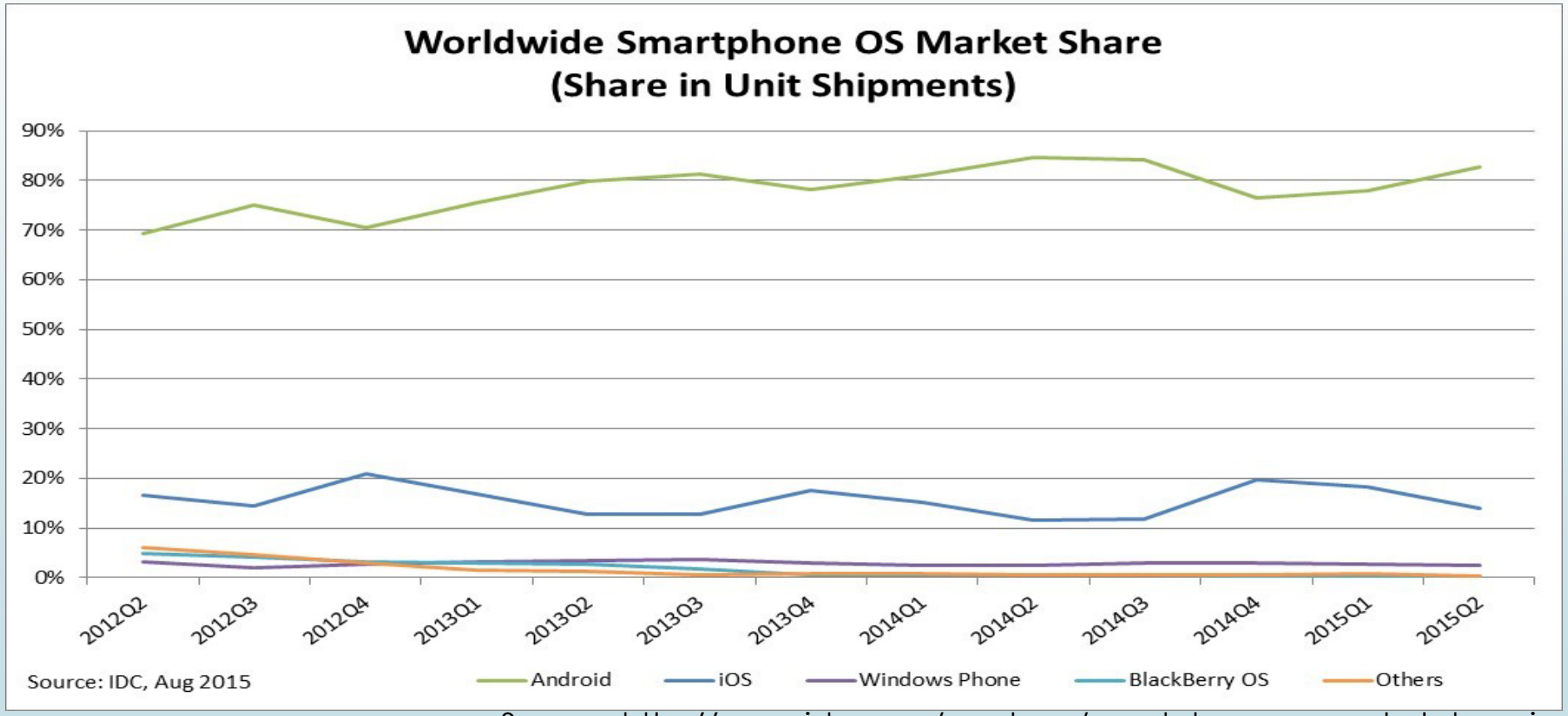
- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

Background

- In 2015Q2, the market share of the Android smart phones is 82.8%, and the market share of IOS is 13.9%.



Source: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Background

- Android global smartphone market share is 84.1% in 2016Q1
- Recently, the booming of Android app market in the cloud has prompted many small teams and individuals to adventure in this market.

Worldwide Smartphone Sales to End Users by Operating System in 1Q16 (Thousands of Units)

Operating System	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
Android	293,771.2	84.1	264,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	**349,251.4**	**100.0**	**336,297.8**	**100.0**

Source: Gartner (May2016)

Source: http://bgr.com/2016/05/23/smartphone-market-share-q1-2016/

Motivation

- In practice, the methods of automatic verification usually used the Monkey algorithm randomly clicking on the screen.
- The sequence of random click lacks of logic.

- In order to reducing a lot of cost for testing, we try to make the test cases more logical.
- With the lowest cost of manpower added, we want to improve the efficiency of random click.

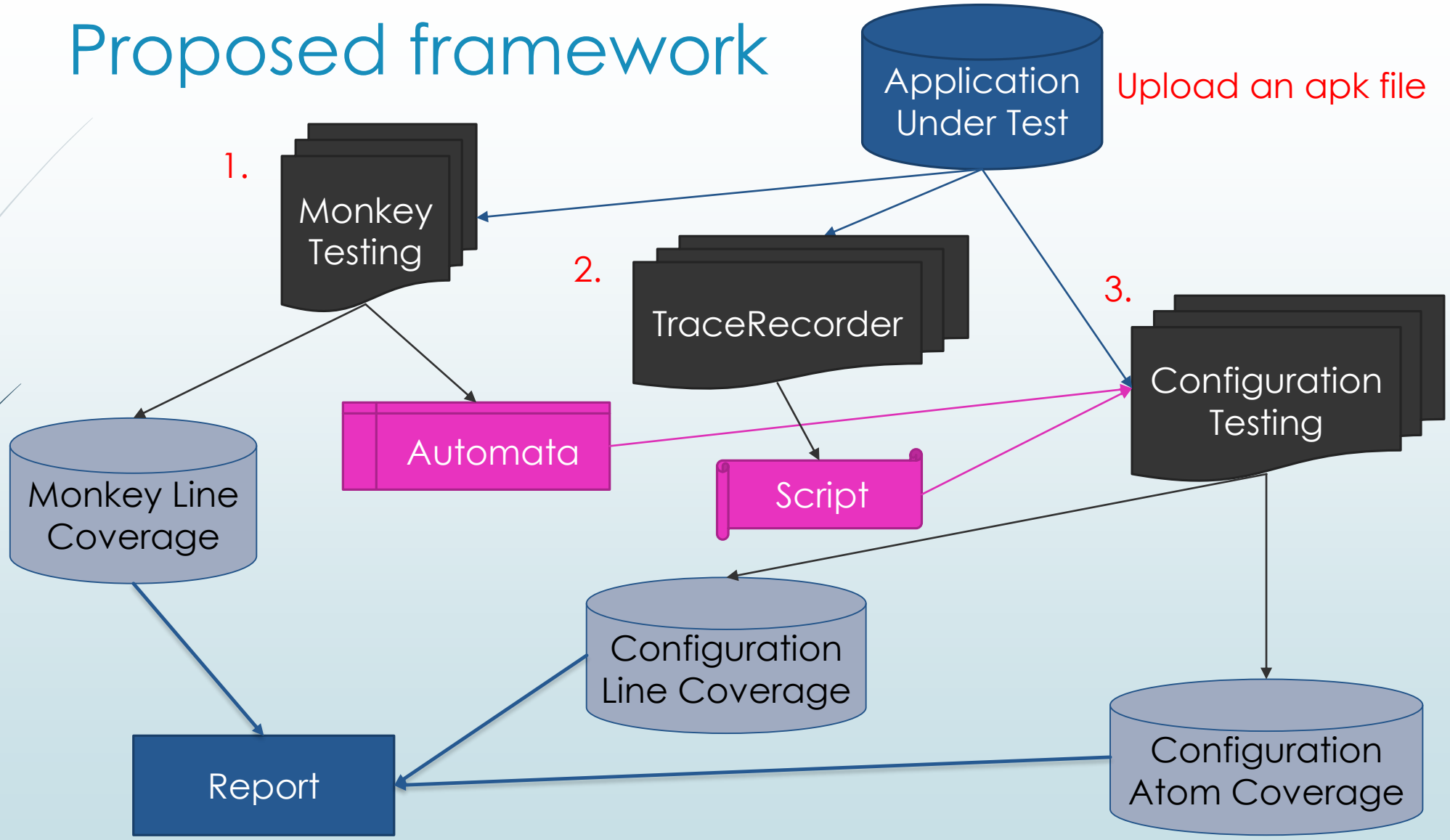
Purpose

- We propose a framework to do Configuration Testing. Normally, many code of apps would be triggered by changing settings.
- This framework includes a tool "TraceRecorder" for helping users to build a script easily.
- we also proposed a detection technique to calculate code coverage. With coverage information, we can verify the efficiency improvement.

Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

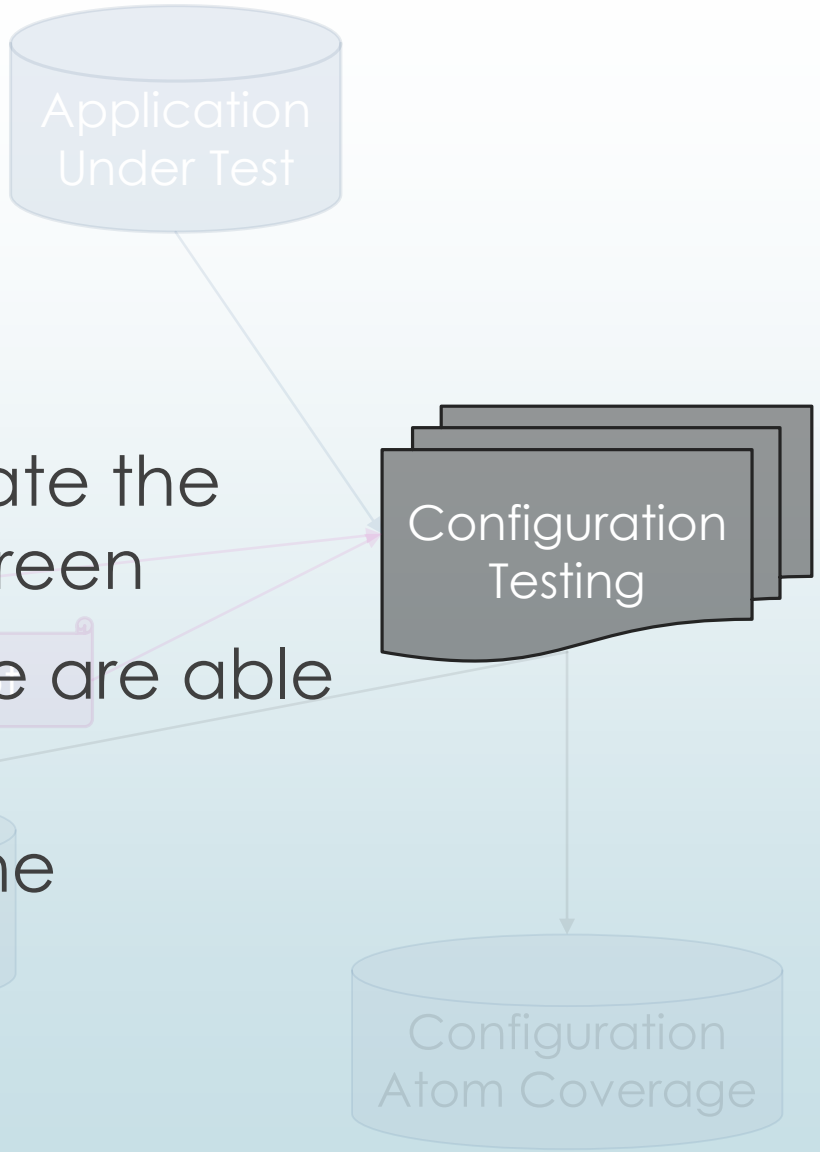
Proposed framework



Configuration Testing

- Input: Automata, Script, Depth

- With Automata, we can calculate the shortest path between each screen
- Script execution ensures that we are able to walk through setting screen
- Depth of exploration decides the complexity of settings



Configuration Testing

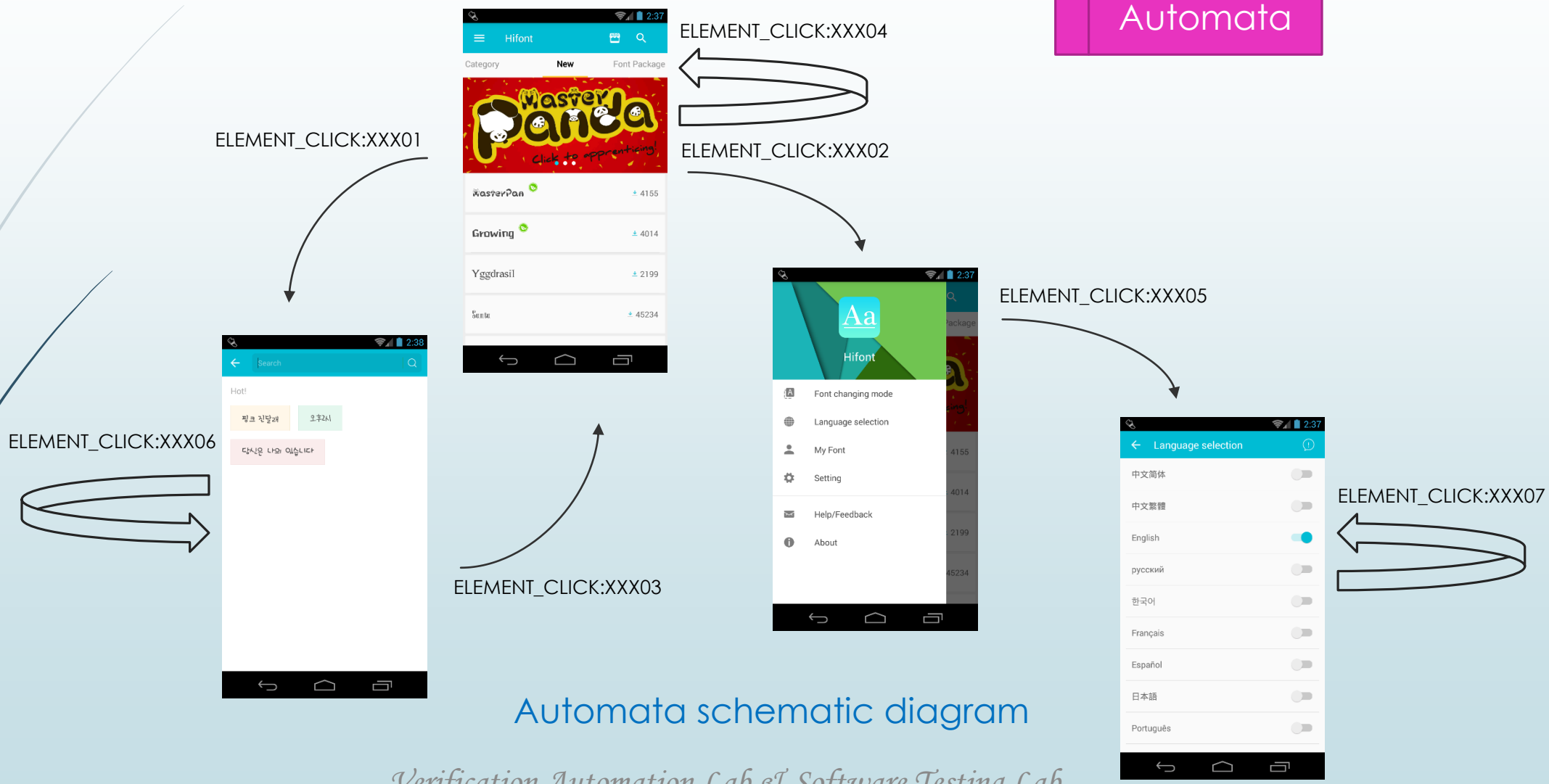
Automata

- After finishing first Monkey Testing, we will build a version Automata
- Automata records :

  the state of each screen, the object features in screens, …
- More importantly, It records the changes and the relationship between each screen.

Configuration Testing

Automata



Automata schematic diagram

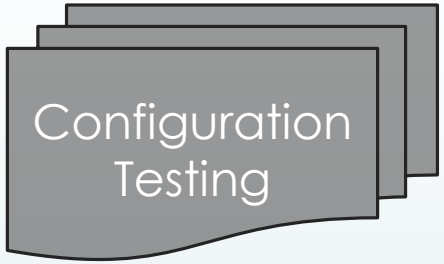
Configuration Testing

Script

- After finishing TraceRecorder, we will get a script
- Script records :

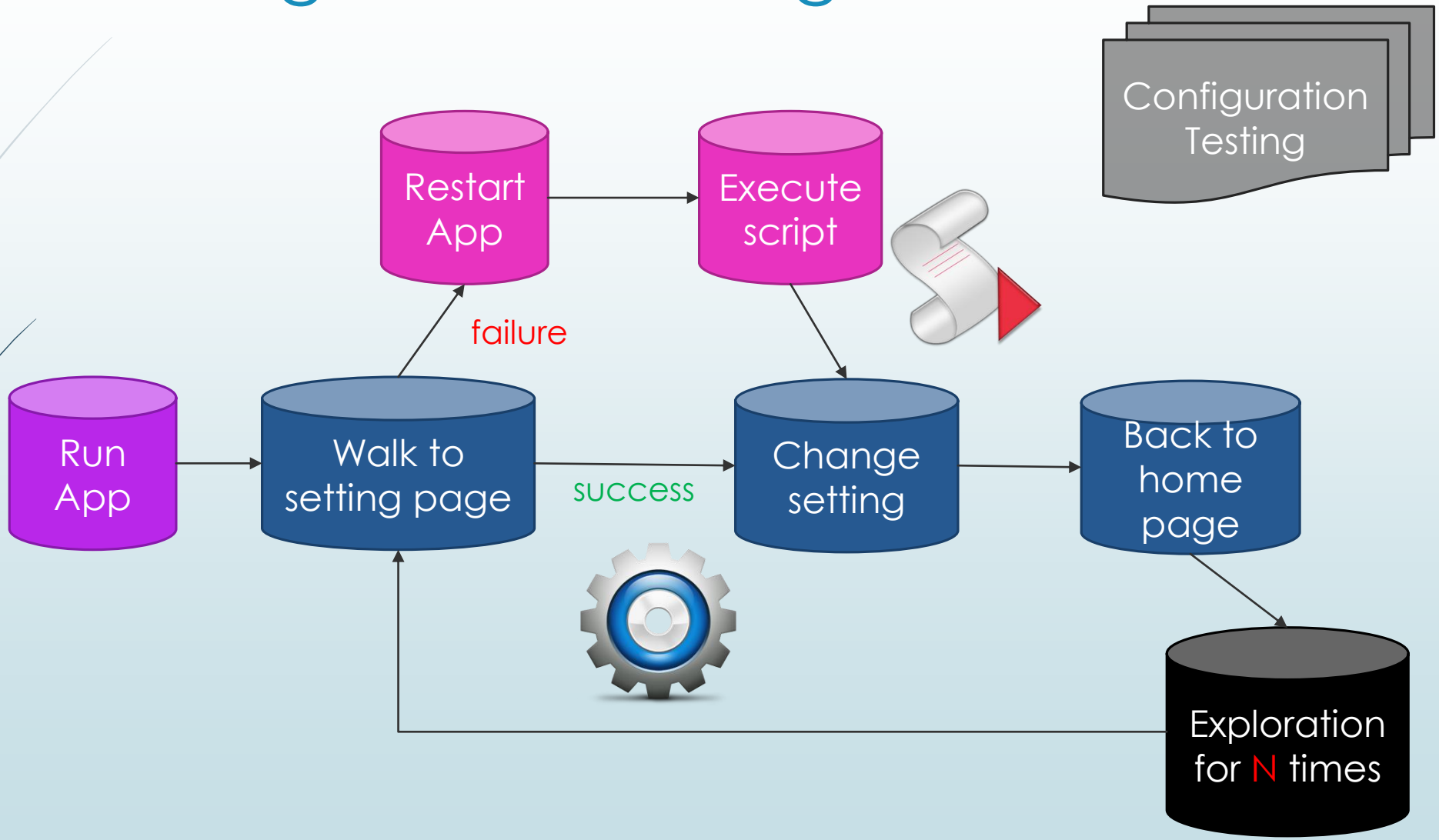
  the most direct path that the user go to a specific screen, like settings, …

Configuration Testing



- When the app is explored, the program set by parameters will decide whether to travel between the main screen and setting screen.

Configuration Testing

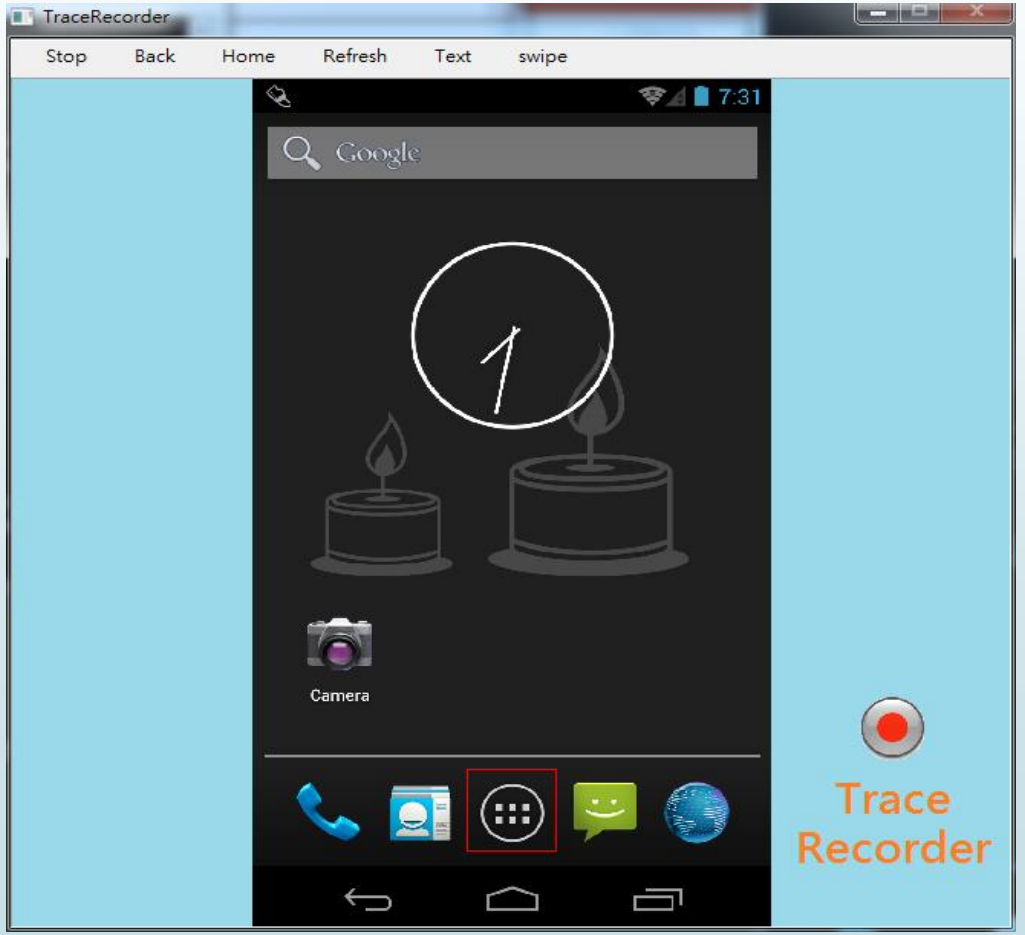


Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

TraceRecorder

- TraceRecorder, a tool to help tester easily record test traces as a script.



- The basic layout of TraceRecorder

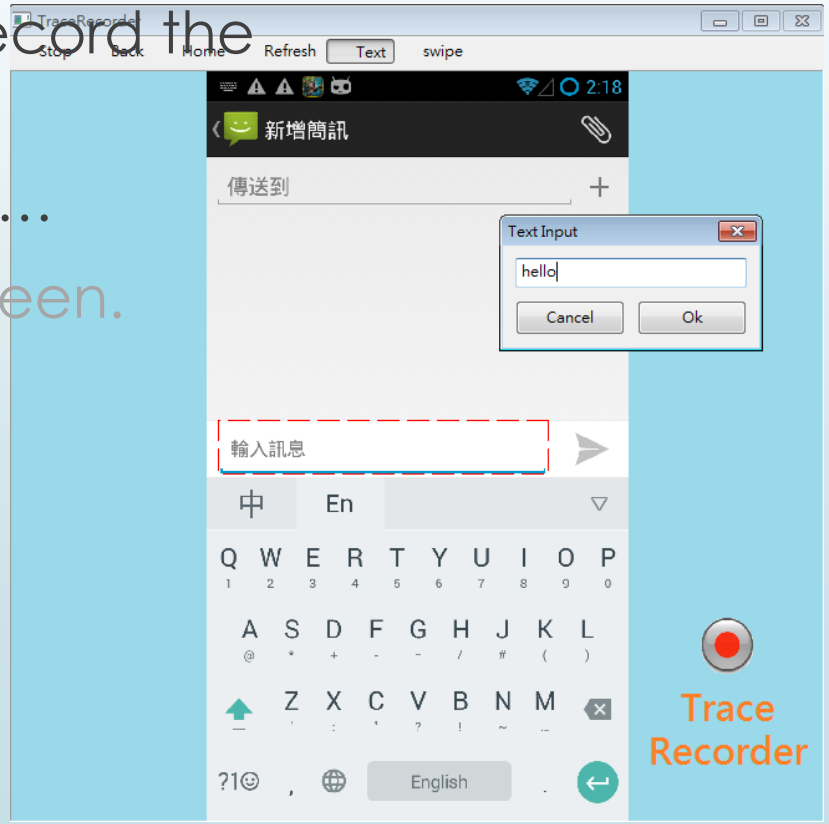
TraceRecorder

- Why make a tool just for scripts?
  - The structures of most apps are not the same.
  - In our existing techniques, writing a script consumes too much time and manpower.
  - The market is almost no similar tool.

TraceRecorder

- The functions of TraceRecorder are the followings:
  - On android devices or emulators, it can record the basic operations including:

    Click, Swipe, Text input, Home, Back, …
  - It records all the XML features on each screen.
  - It saves the steps as an easy to use script.



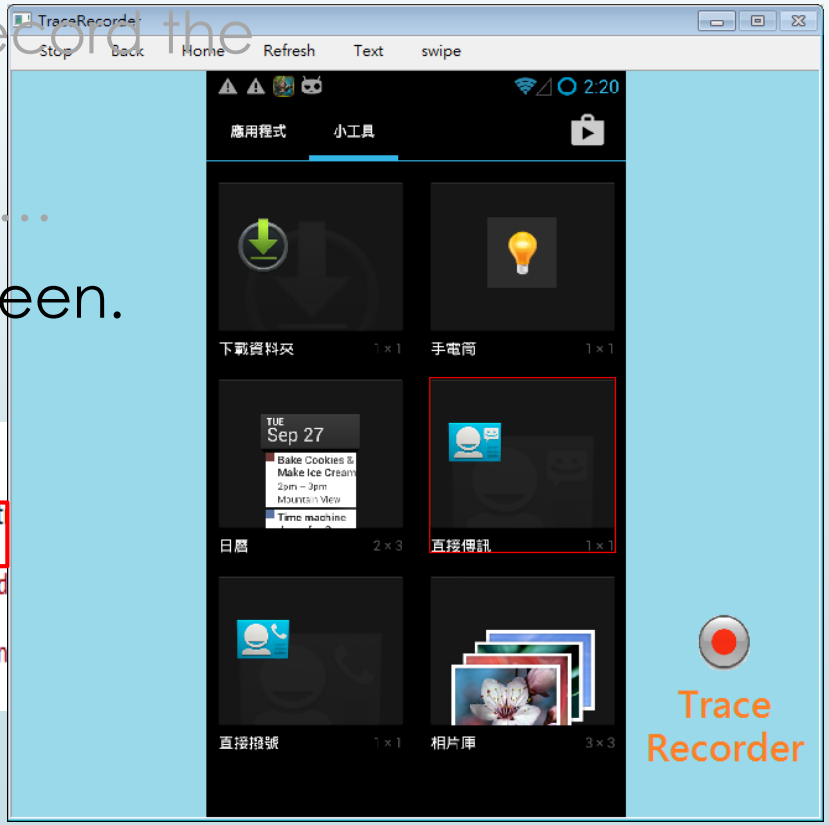
TraceRecorder

- The functions of TraceRecorder are the followings:
  - On android devices or emulators, it can record the basic operations including:

    Click, Swipe, Text input, Home, Back, …
  - It records all the XML features on each screen.
  - It saves the steps as an easy to use script.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <hierarchy rotation="0">
  - <node bounds="[0,0][720,1184]" selected="false" password="false" long-clickable="false" scrollable="false" focused="false" focusable="false" enabled="t
    package="com.xinmei365.font" class="android.widget.FrameLayout" resource-id="" text="" index="0">
    - <node bounds="[0,0][720,1184]" selected="false" password="false" long-clickable="false" scrollable="false" focused="false" focusable="false" enabled
      package="com.xinmei365.font" class="android.widget.LinearLayout" resource-id="" text="" index="0">
      - <node bounds="[0,50][720,1184]" selected="false" password="false" long-clickable="false" scrollable="false" focused="false" focusable="false" en
        package="com.xinmei365.font" class="android.widget.FrameLayout" resource-id="android:id/content" text="" index="0">
```

XML objects



TraceRecorder

- The functions of TraceRecorder are the followings:
  - On android devices or emulators, it can record the basic operations including:

    Click, Swipe, Text input, Home, Back, …
  - It records all the XML features on each screen.
  - It saves the steps as an easy to use script.

TraceRecorder

- the script can make use of the following three examples:
  - Single repeat, in order to achieve stress test.
    - e.g. taking notes in Note Pad, connecting to server repeatedly, …
  - In the automatic test, we can execute the script on the specific screen.
    - e.g. unlocking the phone lock, entering the account password to login, …
  - Combined Monkey or other algorithms, execute the script to a specific page to explore
    - e.g. setting page, the main page of the apps, …

Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

Coverage Analysis

- To verify the efficiency improvement, we propose two techniques to calculate the coverages:
  - Line Coverage
  - Atom Coverage

Line Coverage

- How to get code coverage for the black-box testing?
  - When we only have apk files downloaded from the Internet.
  - We have no Java source code.

Line Coverage

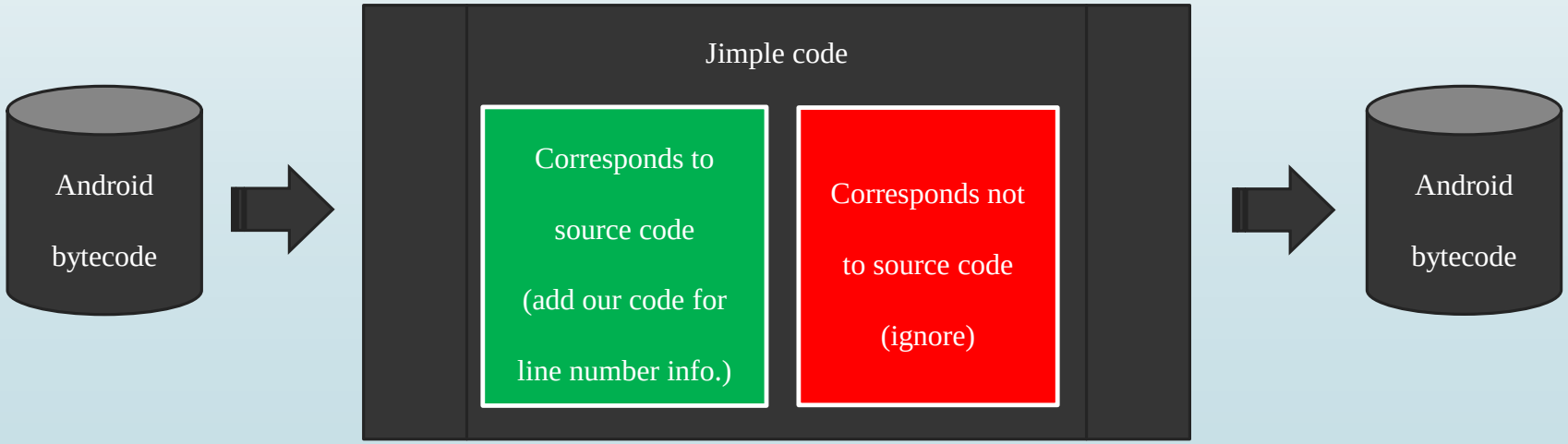
- We use a technique called "Instrumentation"
- First, Our program runs with Soot which is capable of converting Java bytecode to an intermediate representation called a Jimple program.

by the Secure Software Engineering Group at Technische Universität Darmstadt

- Originally, Soot started off as a Java optimization framework. By now, researchers and practitioners from around the world use Soot to analyze, instrument, optimize and visualize Java and Android applications.

Line Coverage

- "Instrumentation"
  - Convert Java bytecode to Jimple
  - Distinguish the source codes
  - Add our codes behind the Jimple code
  - Convert back to Apk file



Line Coverage

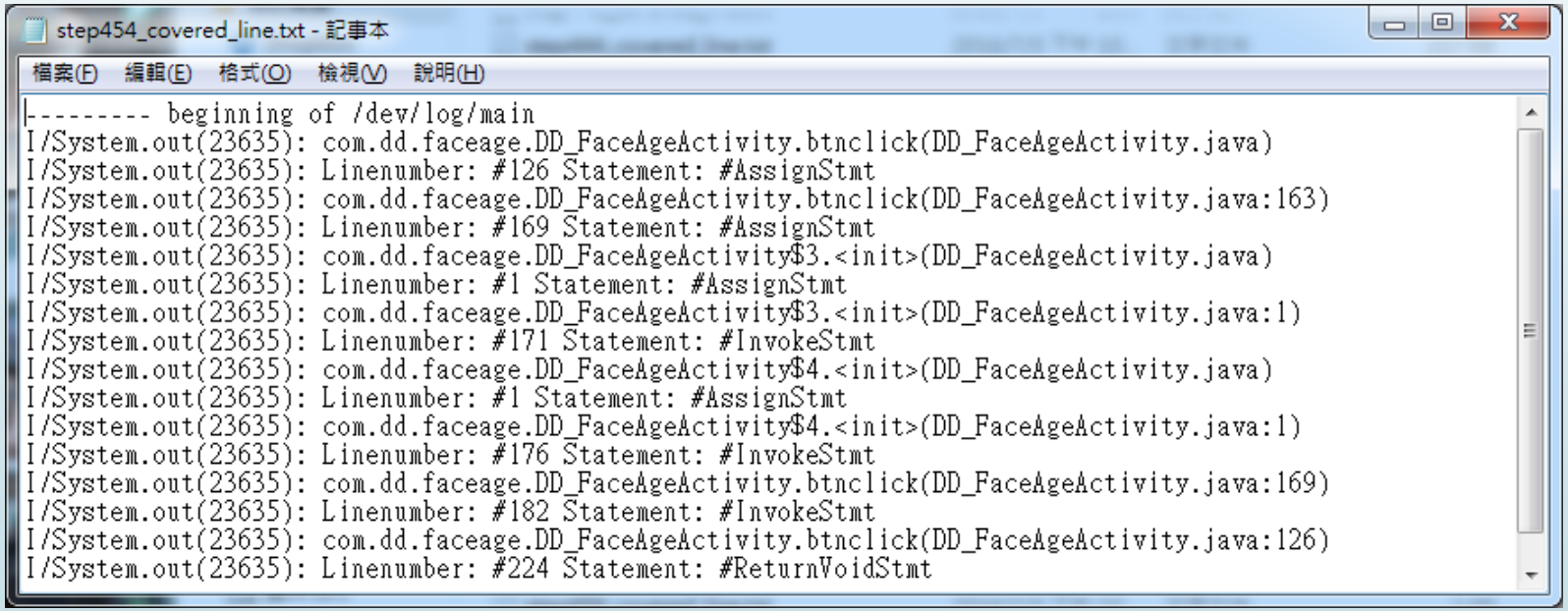
- "Instrumentation"
  - Add some code that the effect of its content is to pass line number information to Android logcat.

```
Transforming android.support.v4.view.dx...
Transforming com.google.android.gms.common.internal.ak...
Transforming android.support.v4.view.du...
line: 25 -> $r0.<android.support.v4.view.du: android.support.v4.view.dx a> = $r1
line: 25 -> specialinvoke $r0.<java.lang.Object: void <init>()>()
line: 28 -> $r2 = $r0.<android.support.v4.view.du: android.support.v4.view.dx a>
line: 28 -> interfaceinvoke $r2.<android.support.v4.view.dx: void a(android.view.View)>($r3)
line: 29 -> return
Transforming android.support.v4.view.dv...
Transforming android.support.v4.view.dy...
line: 31 -> specialinvoke $r0.<java.lang.Object: void <init>()>()
line: 158 -> return $r0
Transforming android.support.v4.view.dz...
line: 26 -> specialinvoke $r0.<android.support.v4.view.dy: void <init>()>()
line: 27 -> $r0.<android.support.v4.view.dz: android.view.WindowInsets a> = $r1
line: 28 -> return
line: 32 -> $r1 = $r0.<android.support.v4.view.dz: android.view.WindowInsets a>
line: 32 -> $i0 = virtualinvoke $r1.<android.view.WindowInsets: int getSystemWindowInsetLeft()>()
line: 77 -> $r1 = new android.support.v4.view.dz
line: 77 -> $r2 = virtualinvoke $r2.<android.view.WindowInsets: android.view.WindowInsets replaceSystem
line: 77 -> specialinvoke $r1.<android.support.v4.view.dz: void <init>(android.view.WindowInsets)>($r2)
line: 37 -> $r1 = $r0.<android.support.v4.view.dz: android.view.WindowInsets a>
line: 37 -> $i0 = virtualinvoke $r1.<android.view.WindowInsets: int getSystemWindowInsetTop()>()
line: 42 -> $r1 = $r0.<android.support.v4.view.dz: android.view.WindowInsets a>
line: 42 -> $i0 = virtualinvoke $r1.<android.view.WindowInsets: int getSystemWindowInsetRight()>()
line: 47 -> $r1 = $r0.<android.support.v4.view.dz: android.view.WindowInsets a>
line: 47 -> $i0 = virtualinvoke $r1.<android.view.WindowInsets: int getSystemWindowInsetBottom()>()
line: 116 -> r1 = $r0.<android.support.v4.view.dz: android.view.WindowInsets a>
Writing APK to: sootOutput\com.baviux.voicechanger.apk
```

- Instrumentation

Line Coverage

- "Instrumentation"
  - After instrumenting, we can dump the information of covered lines from Android logcat



- Information from Android logcat

Coverage Analysis

- To verify the efficiency improvement, we propose two techniques to calculate the coverages:
  - Line Coverage
  - Atom Coverage

Atom Coverage

- In order to analyze the relationship between code coverage and settings, we propose the following method for Atom Coverage of Configuration:

  1. Collect all configuration checkboxes c1, …, cn
  2. Given a set of traces t1, …, tm

  Configuration atom pair coverage =

  (sum (TTcjk + TFcjk + FTcjk + FFcjk))/(2n(n-1))

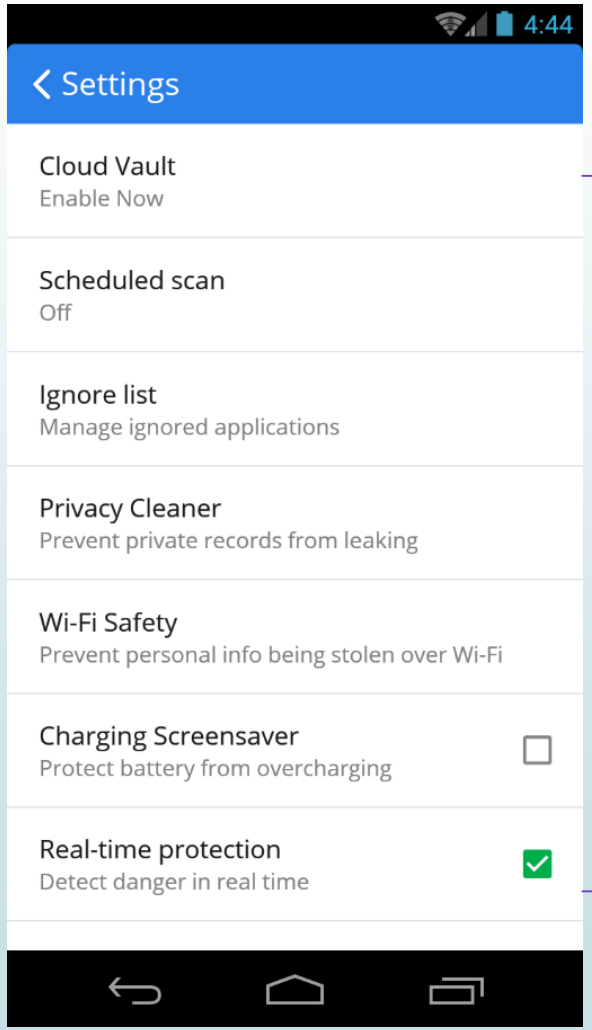
  TTcjk is 1 if cj is True and ck is True once in one of t1, …, tm; 0 otherwise.

  TFcjk is 1 if cj is True and ck is False once in one of t1, …, tm; 0 otherwise.

  FTcjk is 1 if cj is False and ck is True once in one of t1, …, tm; 0 otherwise.

  FFcjk is 1 if cj is False and ck is False once in one of t1, …, tm; 0 otherwise.

Atom Coverage



- Collect all configuration checkboxes c1, …, cn
  (contain child nodes)

Total = 2n(n-1)
(4 combinations between every 2 checkboxes)

Atom pair coverage =
the combinations we reached / Total

Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

Experiment

- In experiment, we present two experiments to show:
  - The effect of instrumentation
  - The efficiency improvement of Configuration Testing

Environment

- Laptop: CJS JS-153CR
- Operating System: Windows 7 Home Premium (64bits)
- CPU: i5 3210M dual-core 2.5GHz (Turbo to 3.1GHz)
- RAM: 8GB DDR3-1333
- Storage: 128GB Plextor M5 SSD
- Graphics card: nVIDIA GT650M

- We need to install:
  1. Python3
  2. Android SDK
  3. JDK

- Mobile phone: Samsung S3 (model GT-i9300)
- Android Version: 4.3.1
- CPU: 1.4 GHz quad-core
- RAM: 1 GB , 16 GB ROM
- Resolution: 720 x 1280 pixels

Experiment

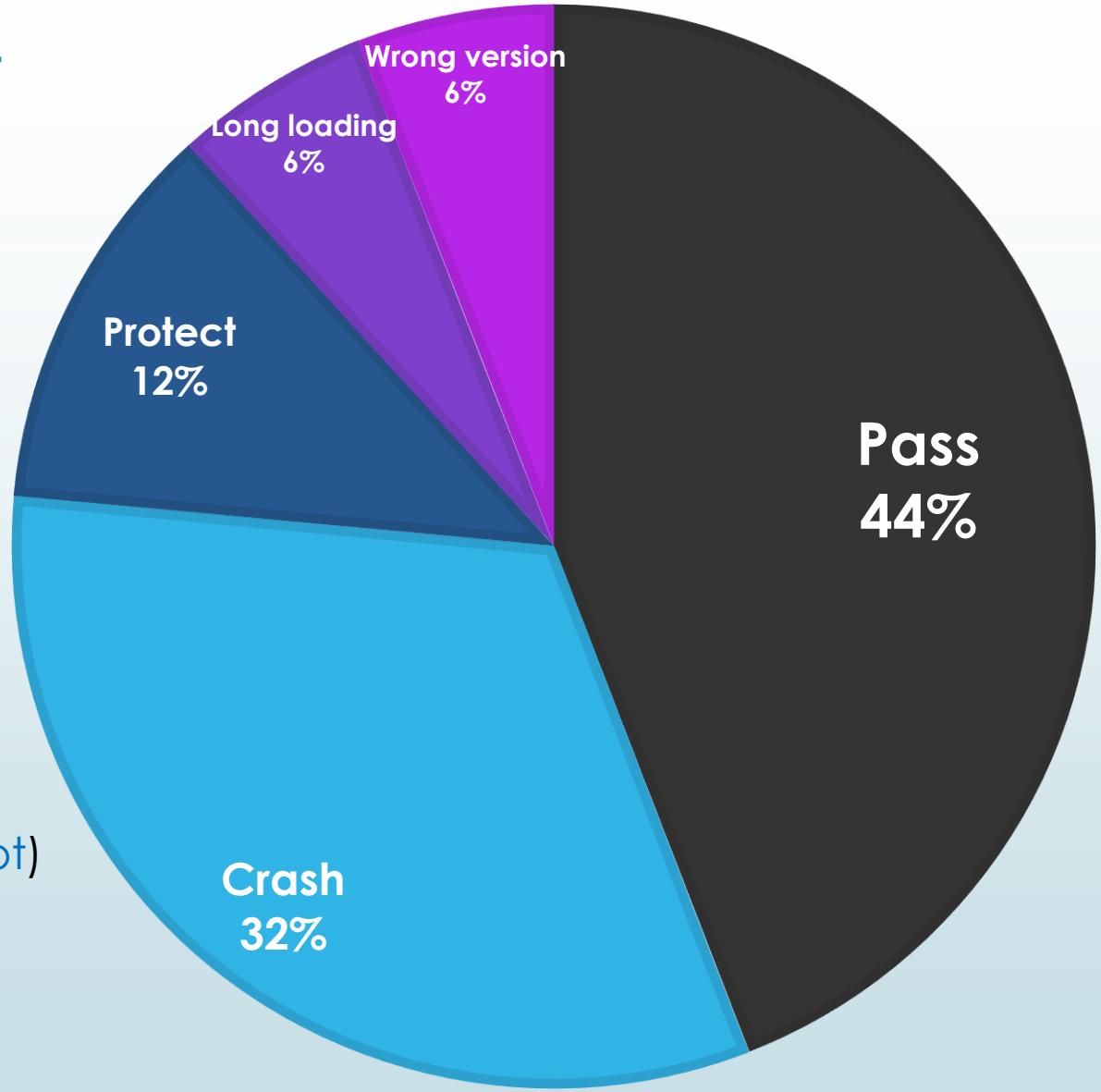
- In the first experiment, we download and test one hundred the most famous applications (including 35 games) on the Google play store.

Experiment

LINE Bubble 2	CM Launcher	Arcade fishing	KKBOX	World of Tanks Blitz
LINE Puzzle TanTan	WeChat	神偷奶爸：奔跑小小兵	Yahoo Mail	字體管家
一國兩國三國誌	B612 - Selfie from the heart	How old do I look?	神魔之塔	Doodle Jump
LINE	Power	靠霸三國	成語大挑戰 (繁體版)	BeeTalk
Dubsmash	aillis (原LINE camera)	玩美	91mai x 全家館	列王的紛爭
Clean Master	Peppa在超市	Camera360「相機360」	Skype	Yummy Gummy
aa	Candy Crush Saga	PPS影音HD	歡唱	迷宮之王
天天动听播放器	APUS 桌面	Peel Smart Remote	手机铃声下载	Implosion 聚爆
CM Security	Terra Battle	追追漫画	萬萬沒想到	魔方賽跑
LINE PLAY 打造完美小屋	Google 翻譯	我就是要換主題	特效變聲器	Plants vs. Zombies
Instagram	富豪娛樂城	Subway Surfers	Audiko 鈴聲	嬰兒遊戲Peppa
1010!	DUAL!	113助手 - 免費點數	綜合所得稅結算申報稅額試算	文件管理器 (File Manager)
1+1猜猜猜	QR Droid Private™ (繁體)	酷我音乐HD	別踩白塊兒（正版）	WhatsApp Messenger
Magic Touch	虫族獵人	藍色光濾波器	CM Browser	卡車司機3D：Offroads
美顏相機	How old do you look like ?	網路第四台	人臉年齡測試	Paktor (拍拖)
加速達人	BEAT MP3 2.0 - 节奏游戏	美图秀秀	LINE 跑跑薑餅人	愛字體(全能字體專家)
金山電池醫生	半價美食、住宿券·GOMAJI團購網	Hola桌面——小而強大	Yahoo新聞- 即時多元、全台參與	Yahoo購物中心
单手划划	Whoscall 來電辨識 & 簡訊過濾, 反詐騙, 象卡來	风行视频HD	小影	中華電信客服
藍色光濾波器 - 夜間模式	D5影视台	Zigbang	Carousell 旋轉拍賣 - 你拍我買！	觸寶輸入法國際版 TouchPal Keyboard
戰爭世界(震撼開戰)	LINE DECO	Photo Grid -相片組合	QR Code Reader	超亮手電筒

Experiment

- The results of Instrumentation (the remaining 73 apps after Soot)

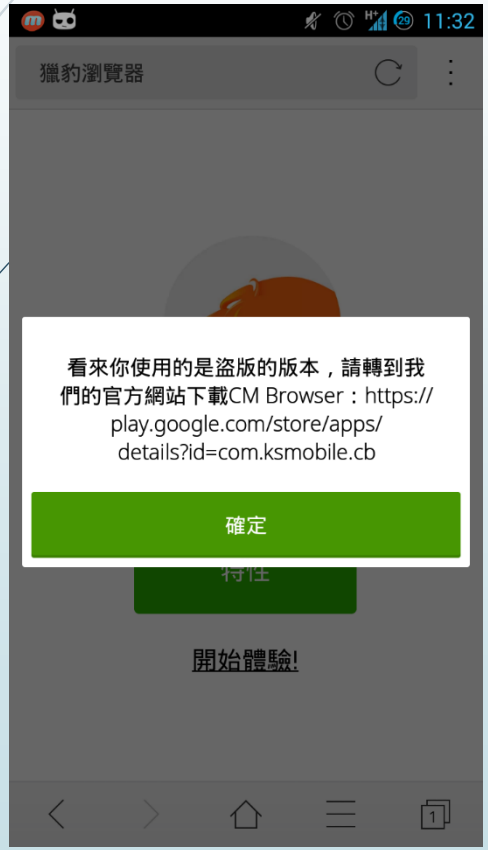


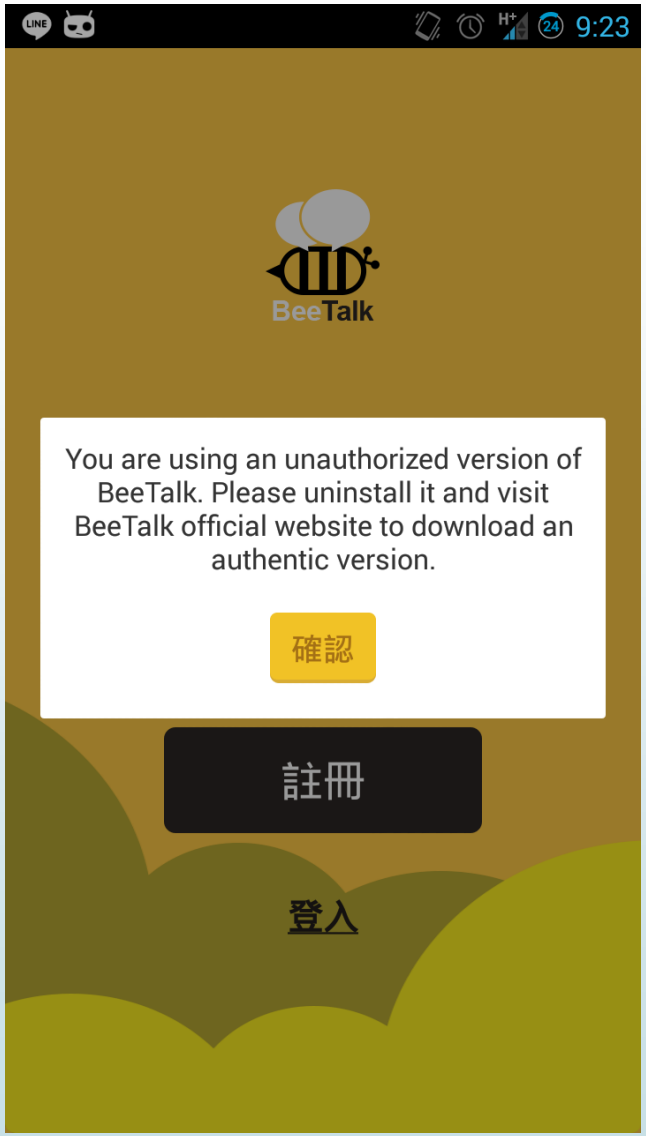
Experiment

- Instrumentation fails on many APKs from Google Play.
- The reason is that some APKs implemented protection code to defend against alteration to their bytecode.
- This is reasonable especially when an application may need to connect to the application's server for accounting purpose.

Experiment

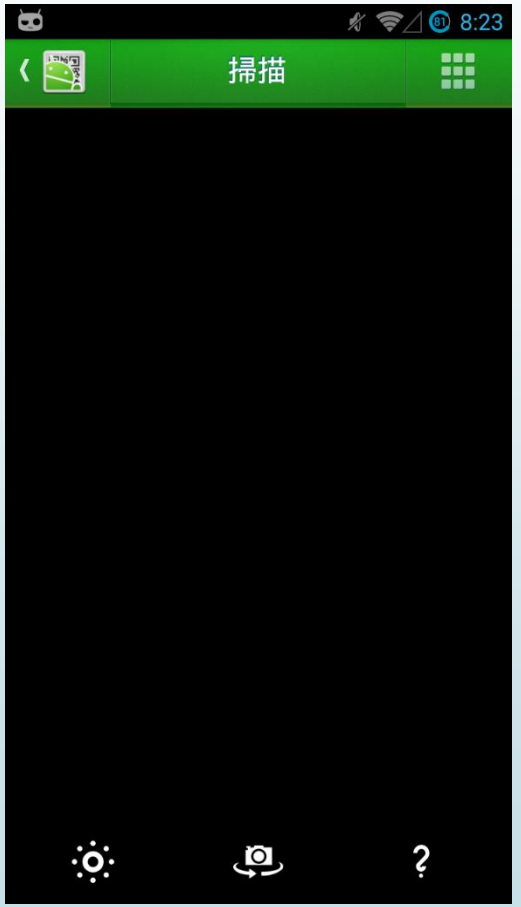
- Failure Cause - Protect

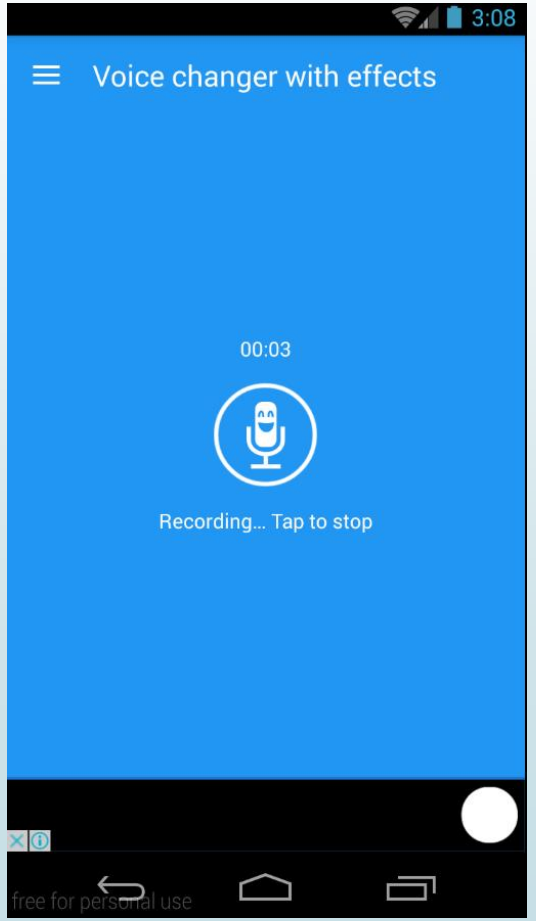




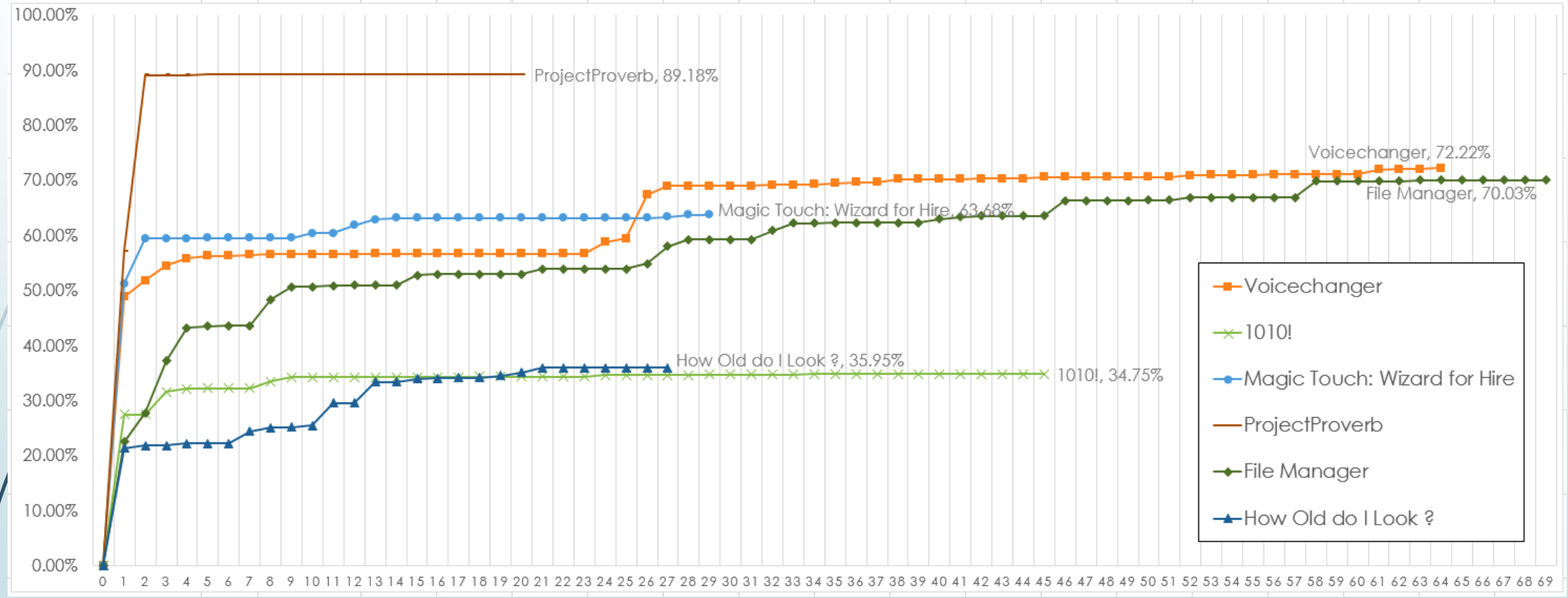
Experiment

- Failure Cause – Crush & Long loading



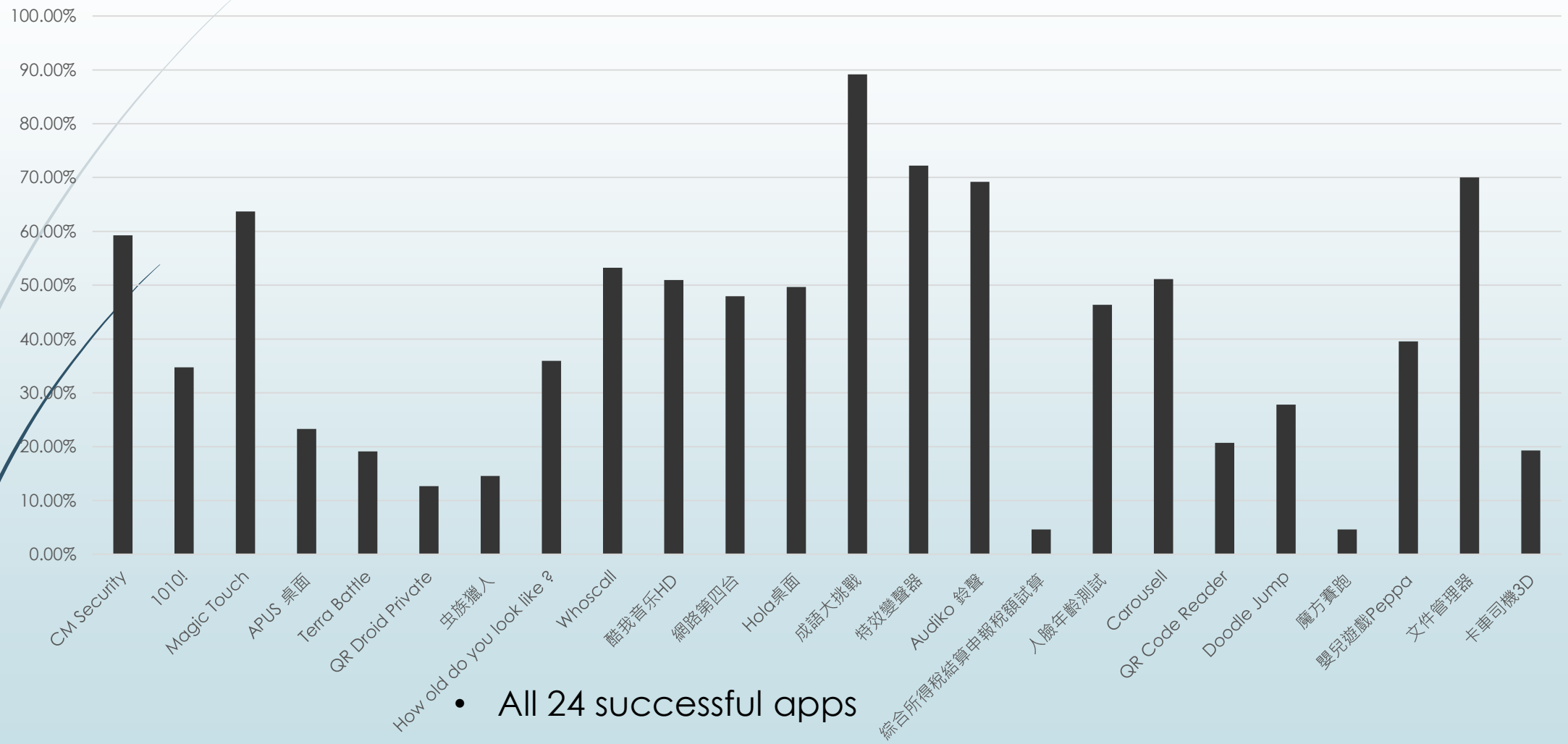


Experiment



- 6 successful apps

Experiment



- All 24 successful apps

Experiment

- In the second experiment, we choose five apps to test under our framework

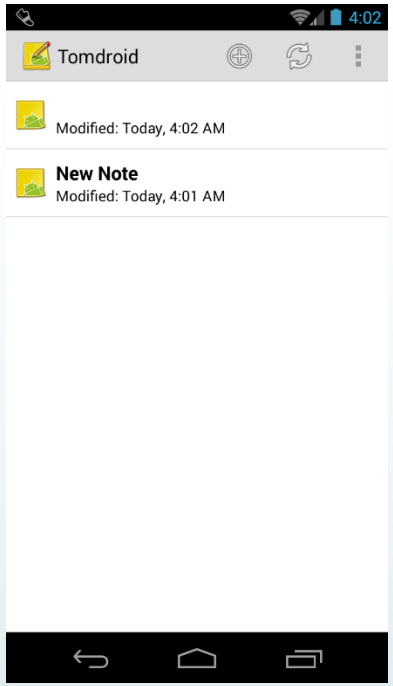
(Monkey Testing, TraceRecorder, Configuration Testing)

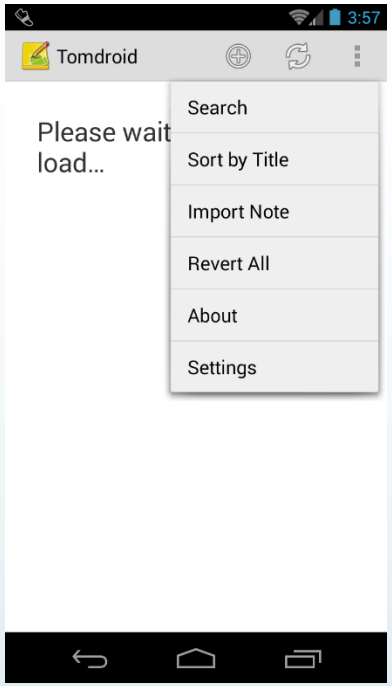
App Name	# of Installations	Rating	# of Ratings	Size
Tomdroid	10,000 - 50,000	3.9	761	1.0M
File Manager	50,000,000 - 100,000,000	4.4	829,166	2.7M
字體管家	10,000,000 - 50,000,000	4.2	558,068	6.5M
CM Security	100,000,000 - 500,000,000	4.7	18,910,135	8.3M
中華電信客服	1,000,000 - 5,000,000	3.8	11,301	6.9M

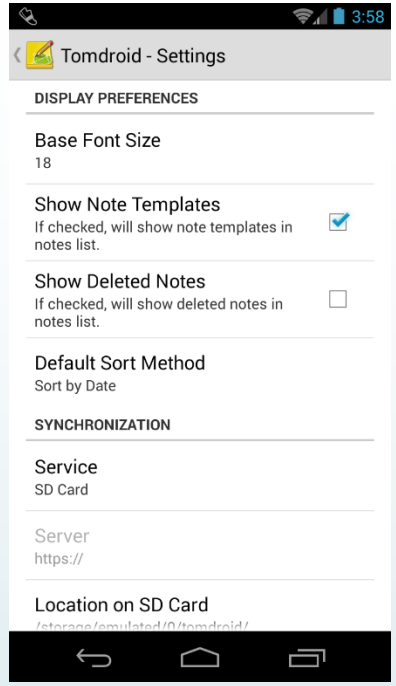
Update: 2016/7/13

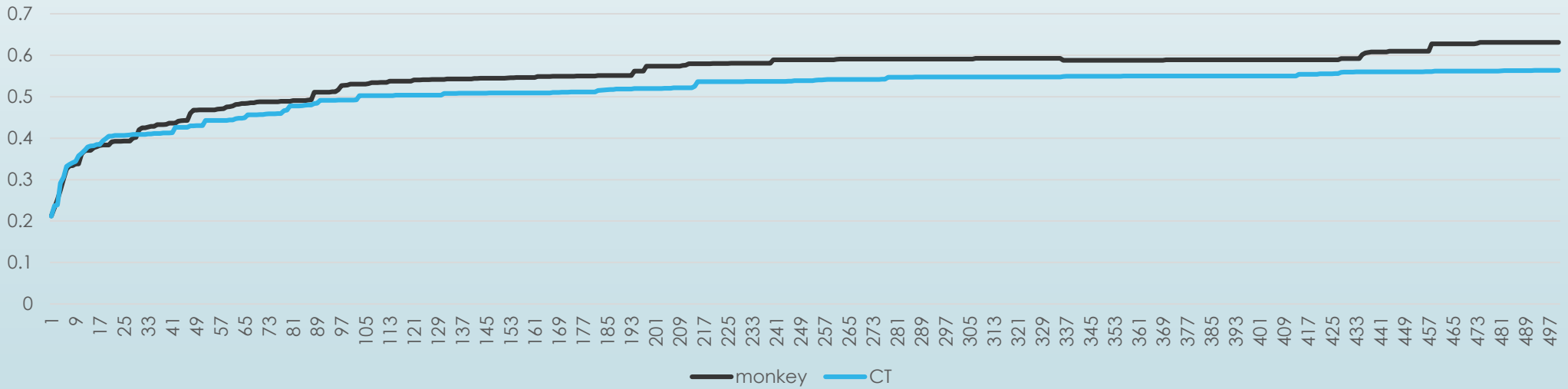
Experiment

- Tomdroid
- Simple
- Few app settings
- Walking to setting screen deliberately is not necessary





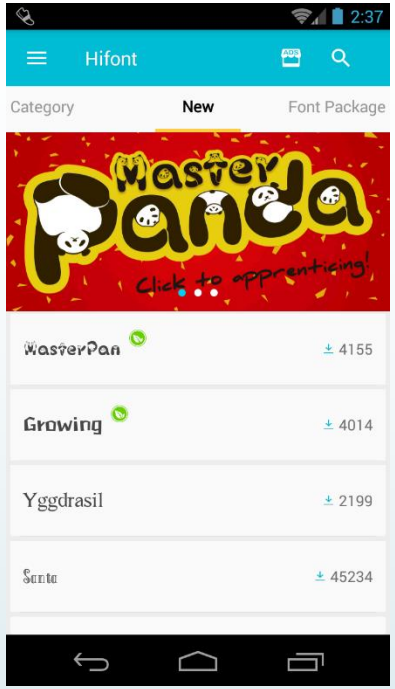


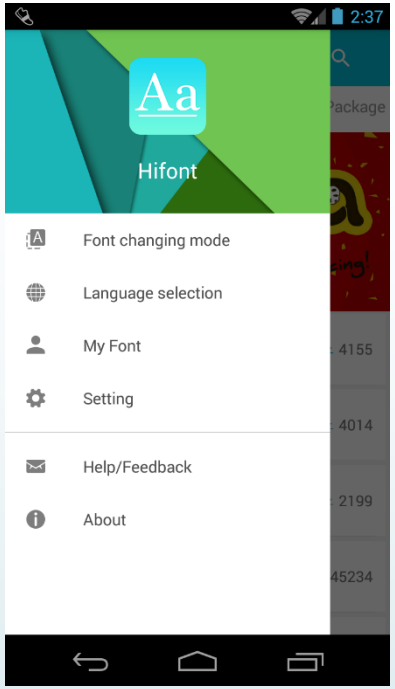


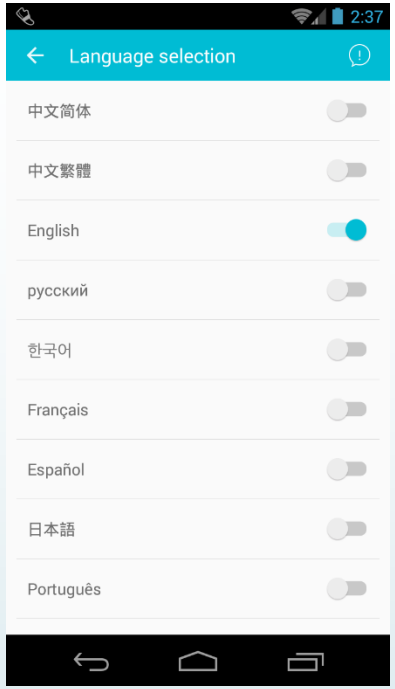
Experiment

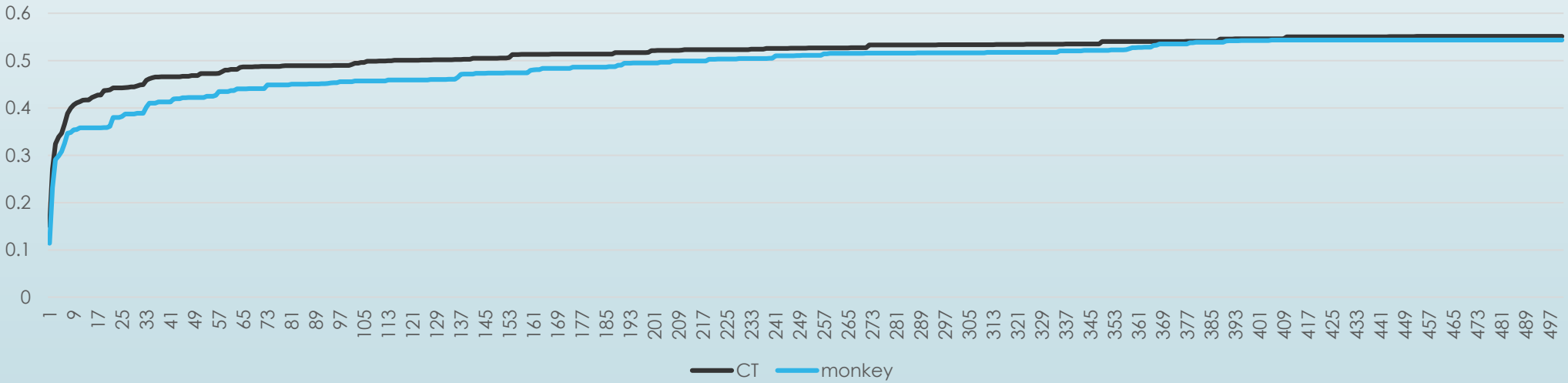
- 字體管家
- Complicated
- Many app settings
- For coverage, monkey 500 steps equal to CT 312 steps

(60% efficiency improvement)



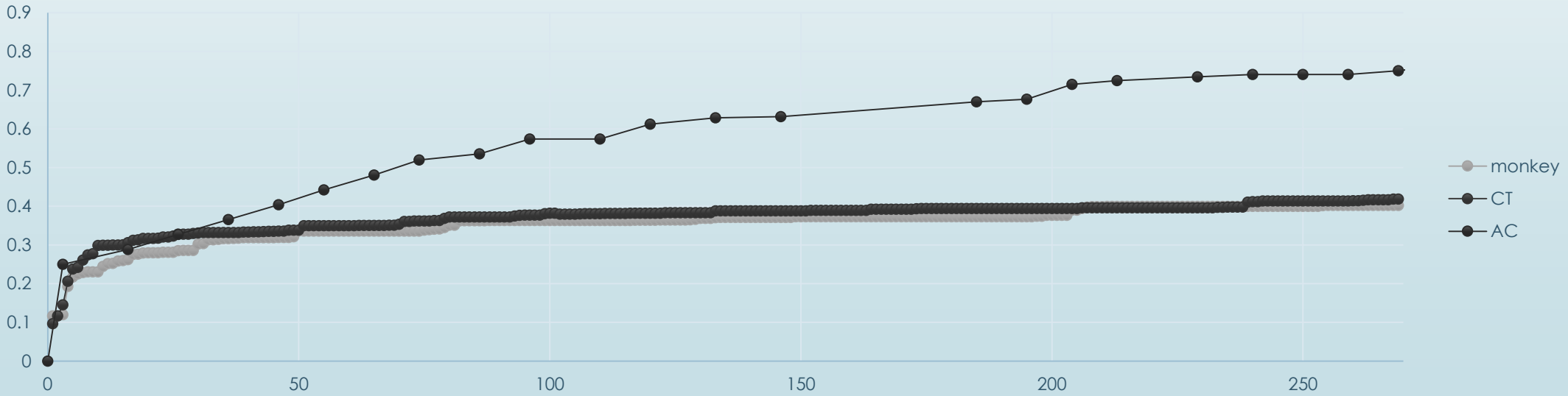






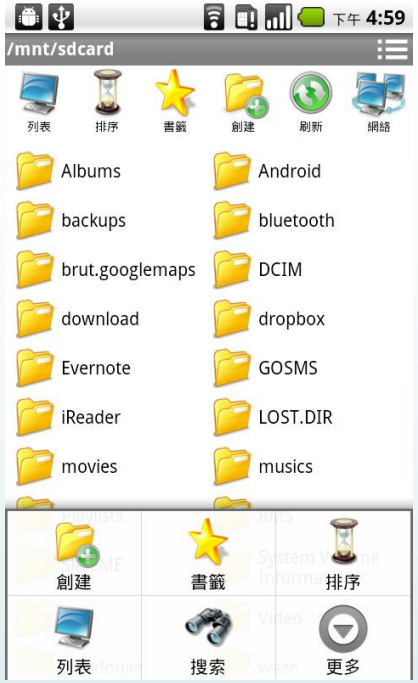
Experiment

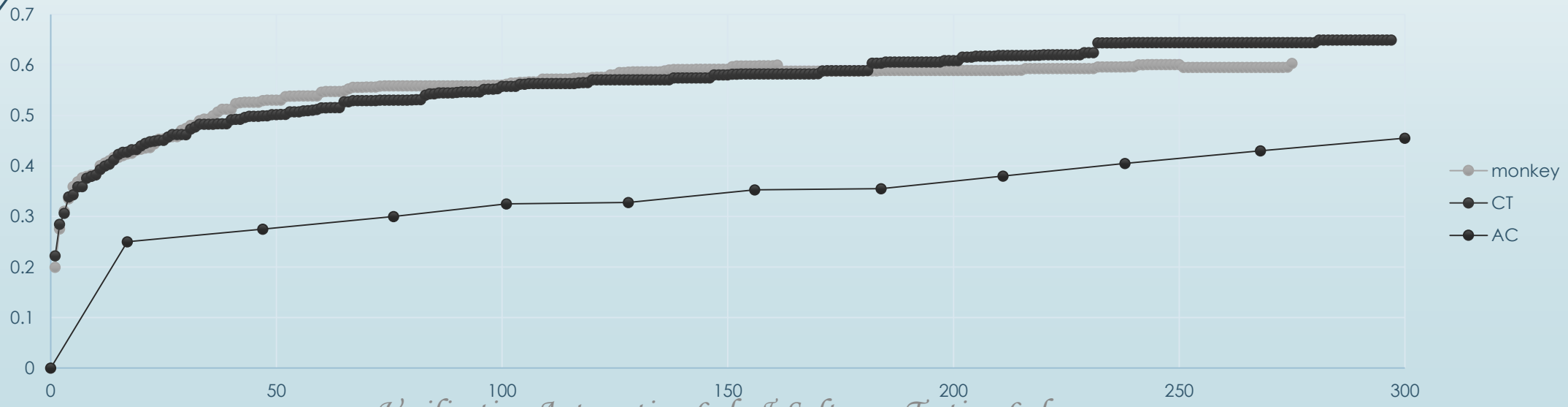
- 中華電信客服
- Complicated
- Few app settings
- Settings do not affect the results



Experiment

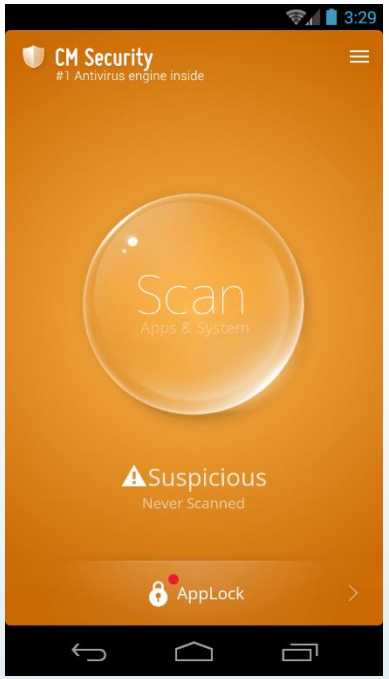
- File Manager
- Very complicated
- Many app settings
- After step:160, CT is still growing as AC line

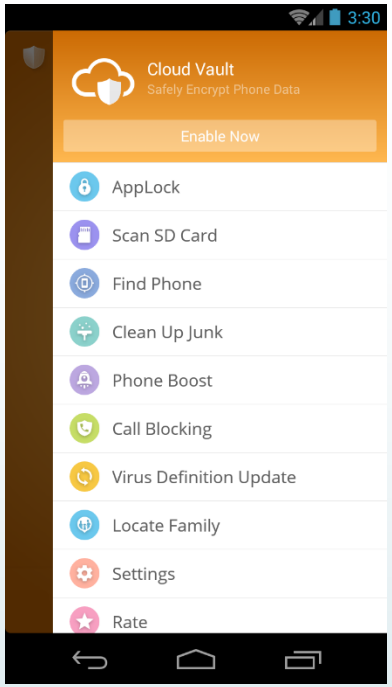


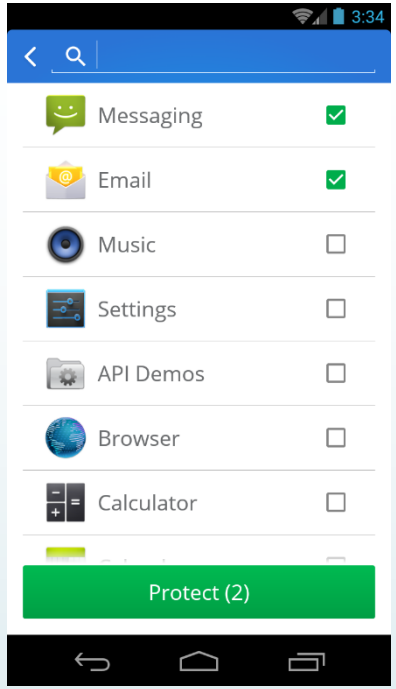


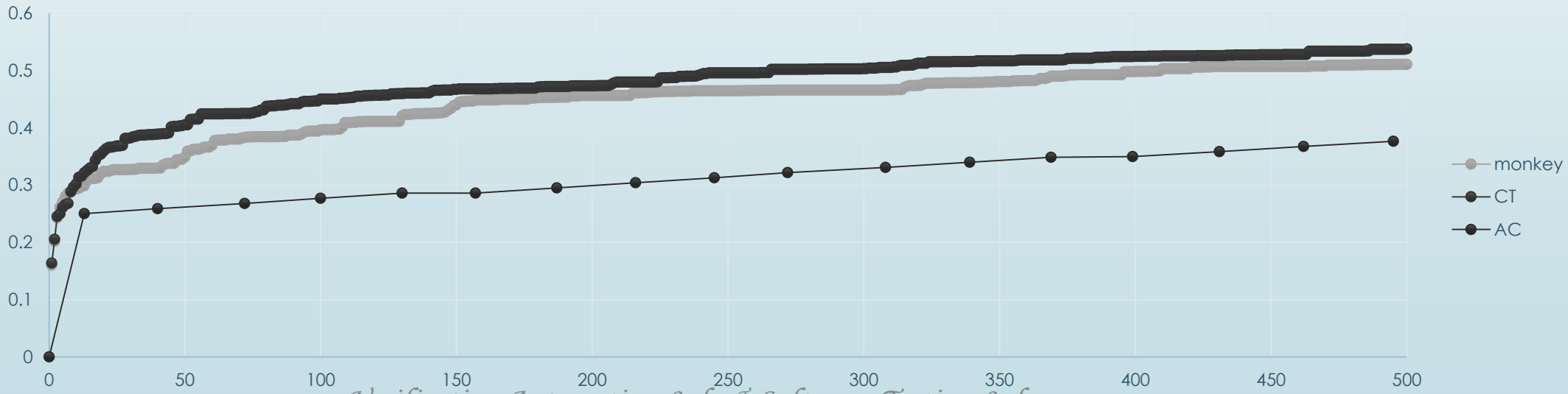
Experiment

- **CM Security**
- Complicated
- Many app settings
- CT has better efficiency (and more similar to AC line)









Outline

- Introduction
- Proposed framework
- TraceRecorder
- Coverage Analysis
- Experiment
- Conclusion

Conclusion

- When the apps have lots of settings parameters, our framework can explore it more efficiently.
- Especially those complicated apps, we can provide more rapid exploration, or more extensive exploration.