한국정보과학회 프로그래밍언어연구회 여름학교 프로그램 (SIGPL)

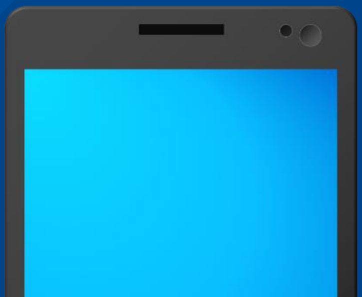
Challenges of Automated Model-based GUI Testing for Android Apps

2017.08.10

Young-Min Baek, Doo-Hwan Bae

Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea

{ymbaek, bae}@se.kaist.ac.kr



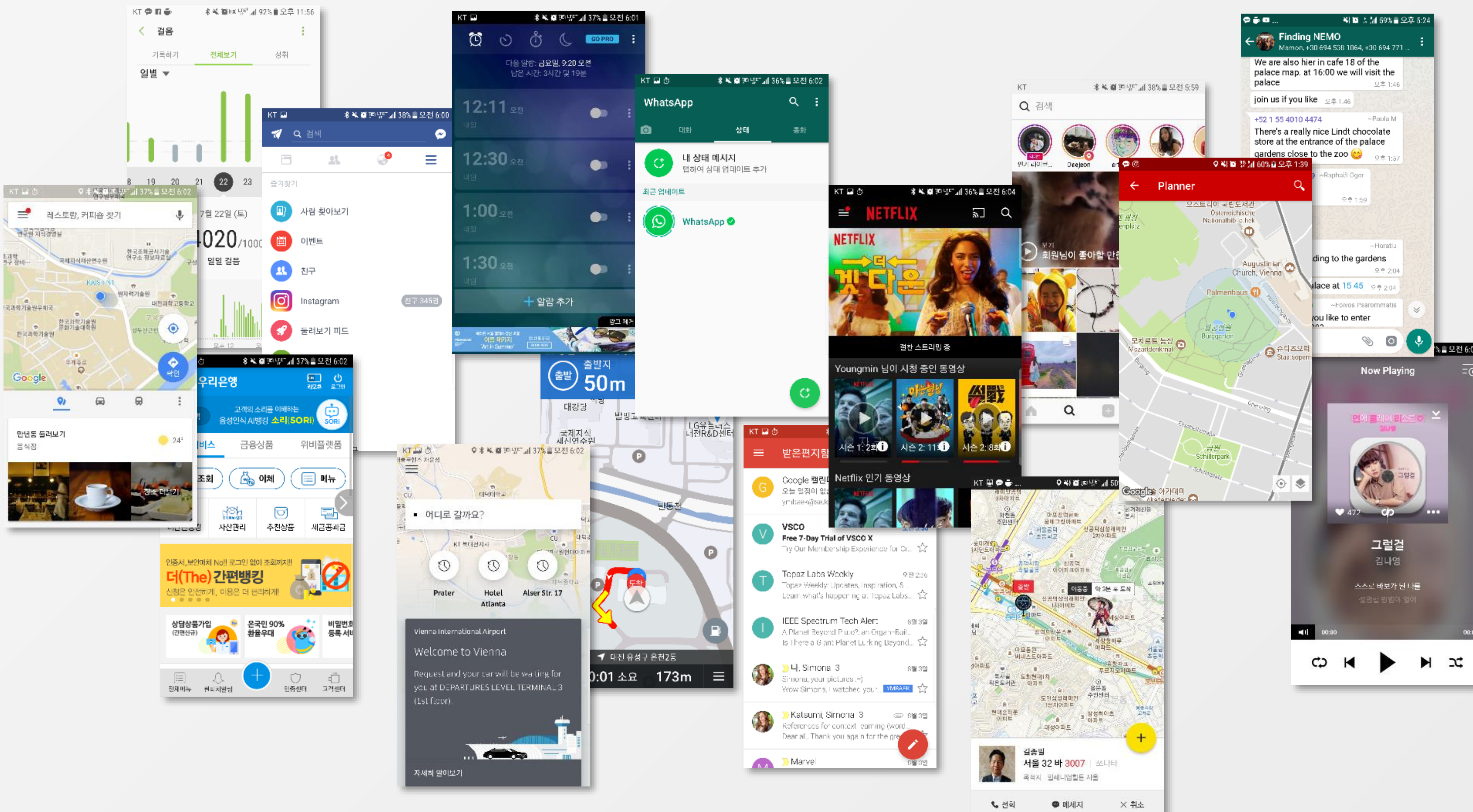
Outline

1. Introduction
2. Graphical User Interface (GUI) Testing
   for Mobile Apps
3. Challenges of Model-based Automated GUI Testing
   for Mobile Apps
4. Our Empirical Study
   Automated Model-based GUI Testing using
   Multi-level GUI Comparison Criteria (ASE '16)
5. Conclusions

Introduction

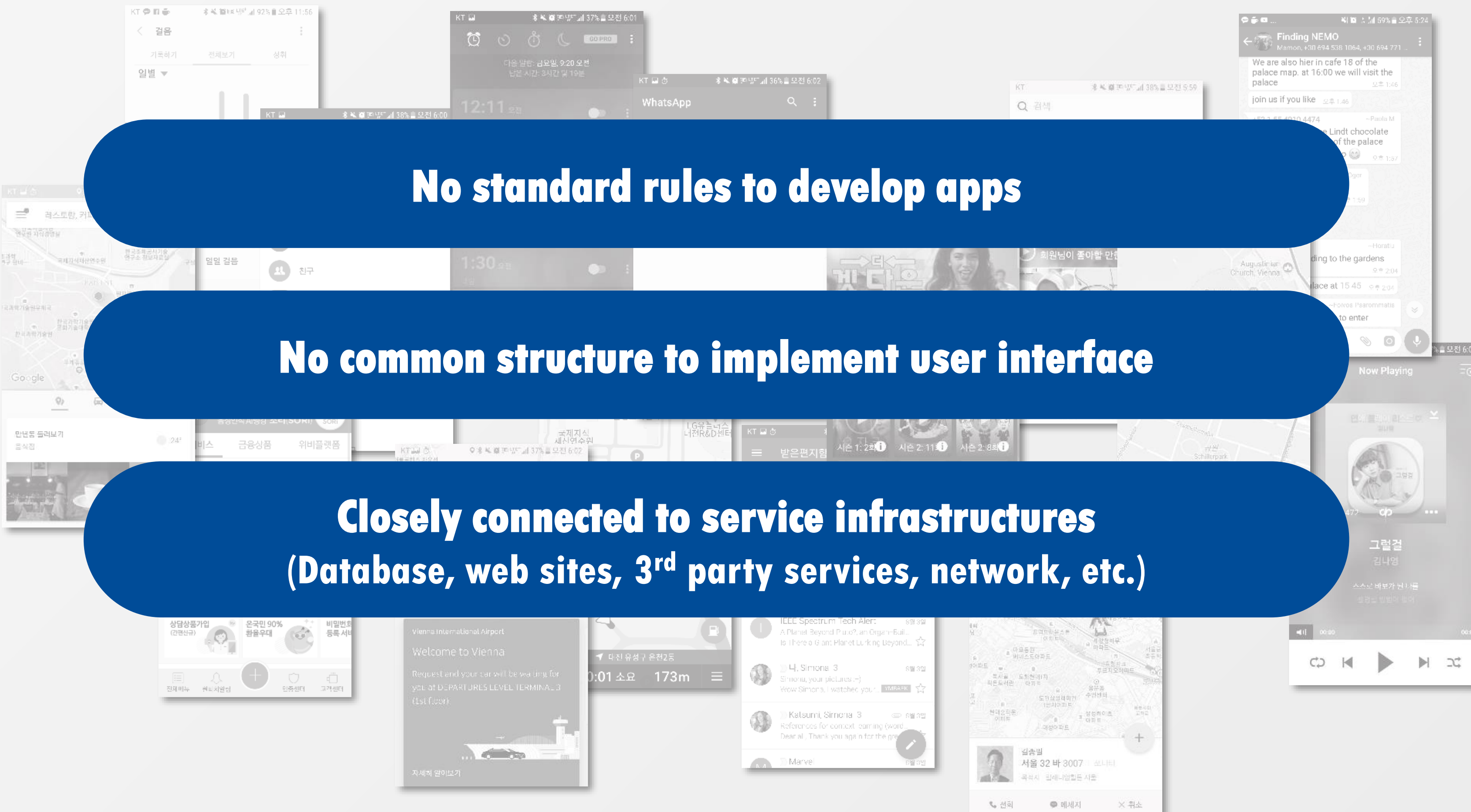
The World of Mobile Applications

- **Overwhelming variety of mobile applications**



The World of Mobile Applications

- **Overwhelming variety of mobile applications**



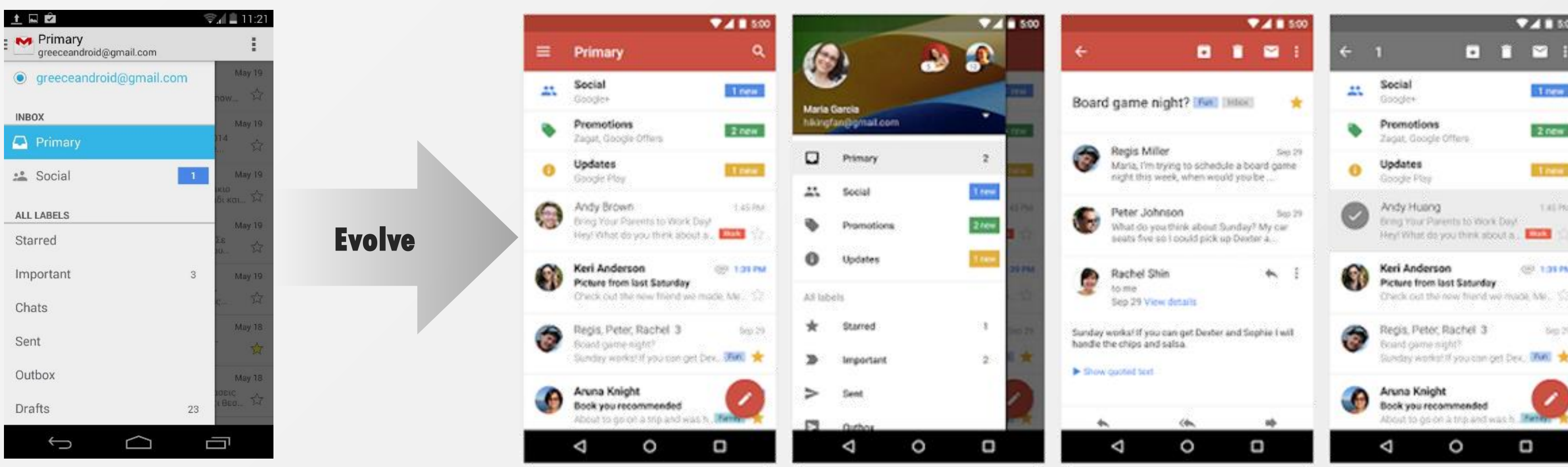
No standard rules to develop apps

No common structure to implement user interface

Closely connected to service infrastructures
(Database, web sites, 3rd party services, network, etc.)

Evolving Mobile Apps

- **Evolving features to improve user experience**
  - Better performance
  - Shallower depth to access screens
  - Simpler graphical design
  - Modern look-and-feel



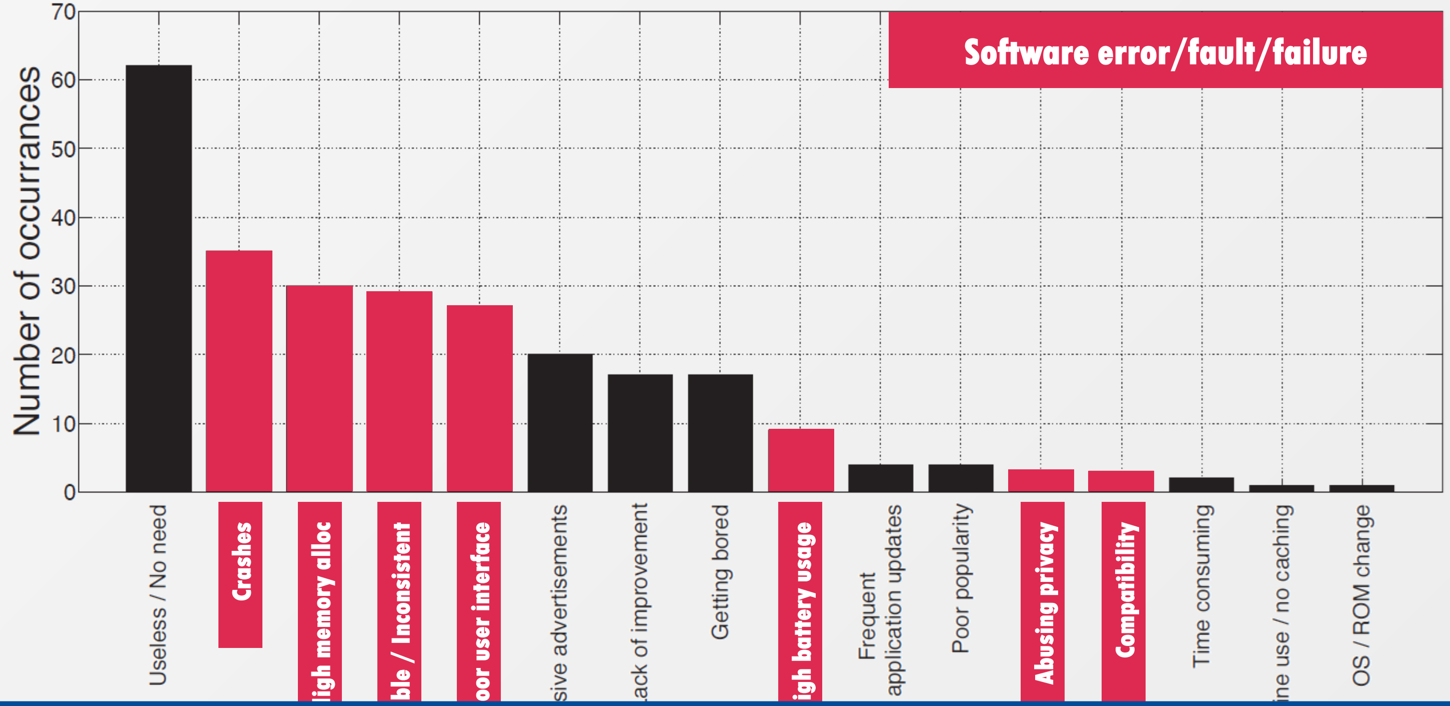
Old gmail app

Current gmail app
with better UI, improved security

http://www.greeceandroid.gr/images/articles/apps/gmail-for-android-updated-to-4-8/gmail-slide-bar-old-1.jpg
https://scdn.androidcommunity.com/wp-content/uploads/2015/06/Gmail-app-for-Android.png

However, Mobile App Users Are…

- **Volatile and easy to leave your app**
  - Reasons to uninstall apps[1]

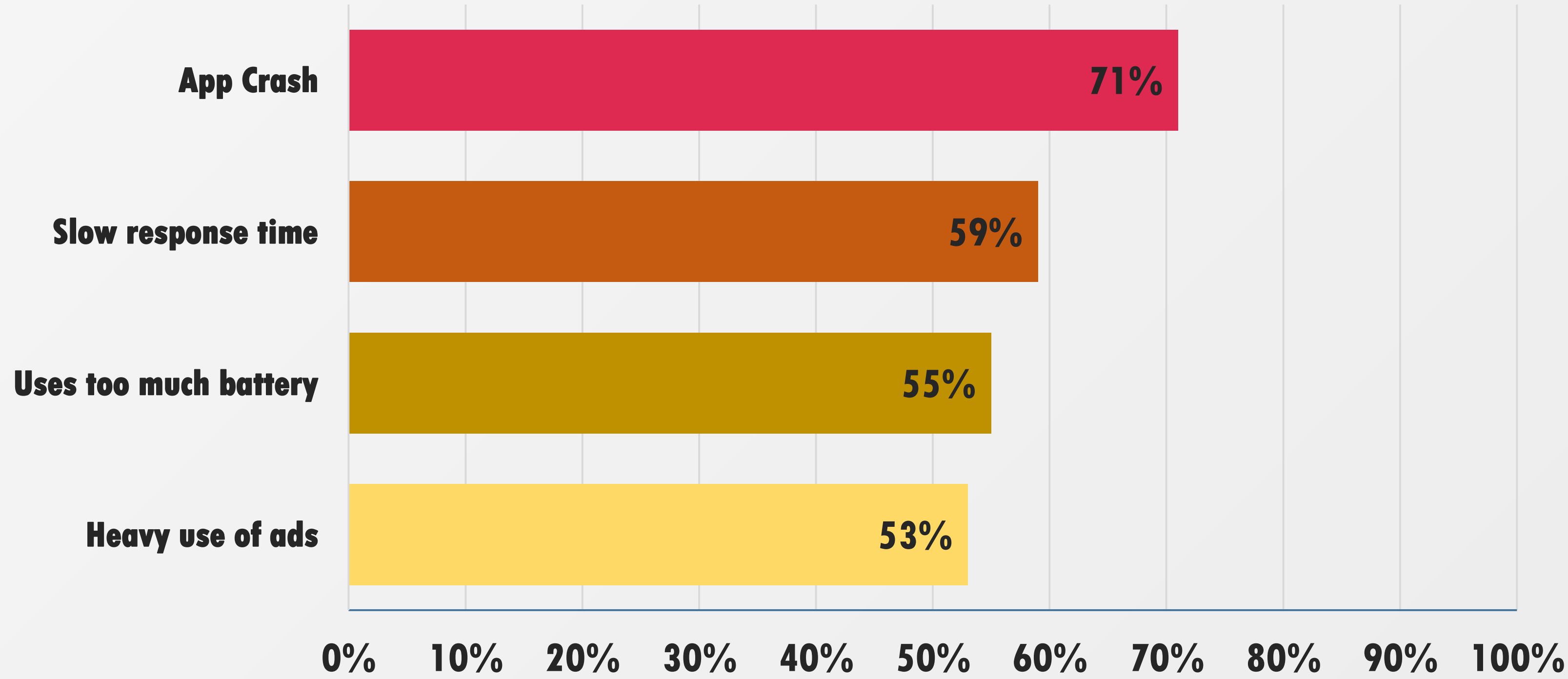


There is a need to predict, detect, solve faults in your app.

[1] Selim Ickin et al., "Why Do Users Install and Delete Apps? A Survey Study," ICSOB '17

Need of Mobile App Testing

- A report by *Usamp* stated the percentage of those deleting an app for specific reasons[1]



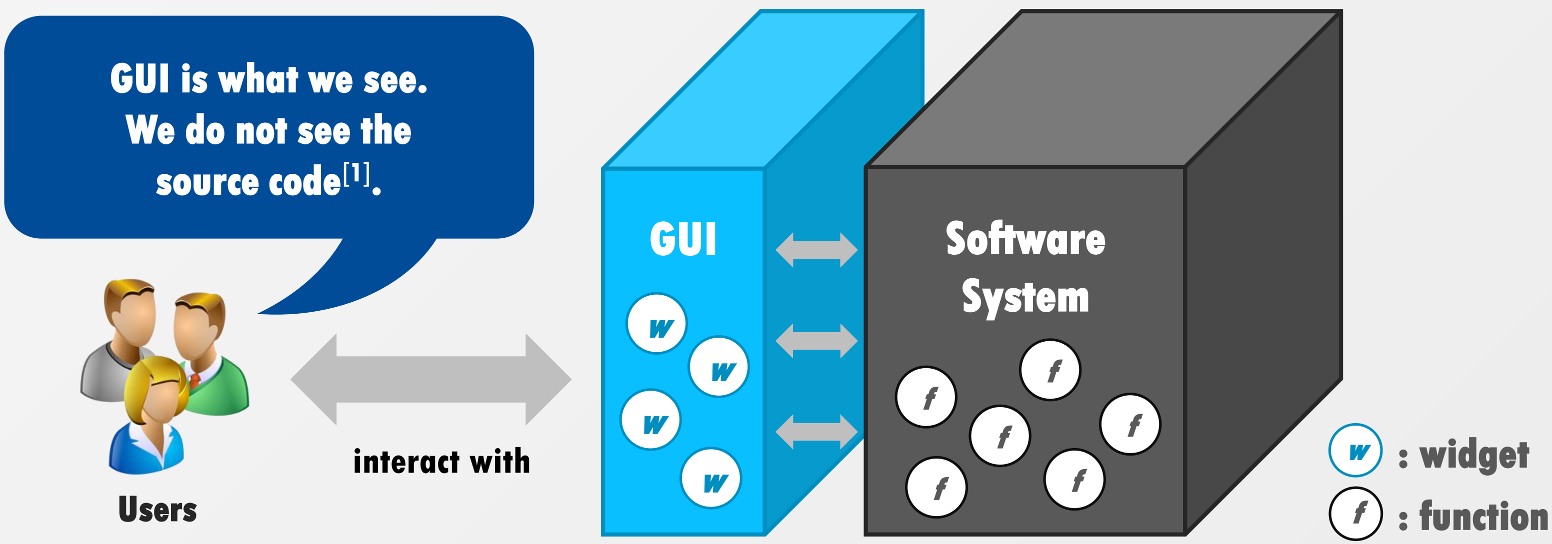
[1] Chupamobile, "Why Your Mobile App is Getting Deleted (And How to Avoid It),"
http://www.chupamobile.com/blog/2014/05/13/why-your-mobile-app-is-getting-deleted-and-how-to-avoid-it/

GUI Testing for Mobile Apps

(Graphical User Interface Testing)

Graphical User Interface (GUI)

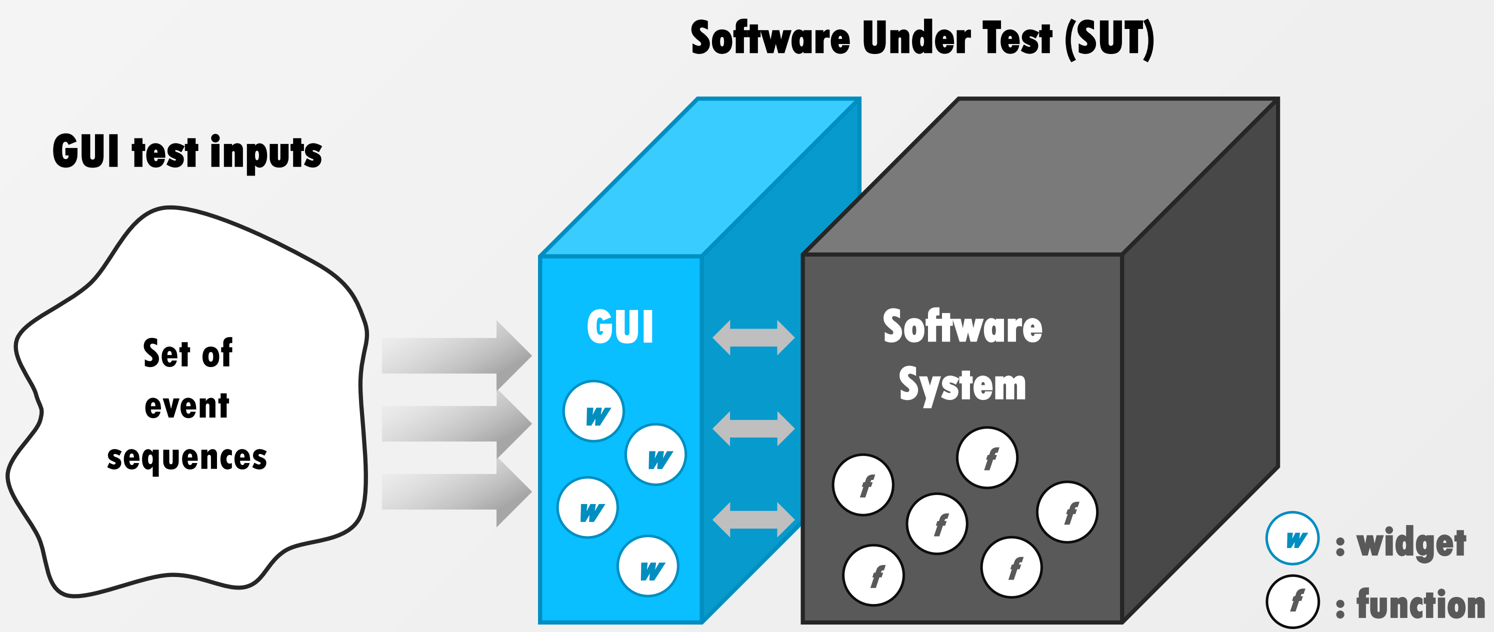
- **GUIs are event-driven components to interact with users.**
  - GUIs enable users to execute functionality via widgets such as buttons, text fields, etc.
  - Users perform actions (events) such as clicking, long-clicking, keyboard typing on the widgets.



[1] Guru99, "Complete Guide for GUI Testing," http://www.guru99.com/gui-testing.html

What is GUI Testing?

- **GUI testing is a process that detects if an application is functionally correct by using its GUIs[1].**
  - To ensure trouble-free use and implementation, from improper output and small bugs to complete system crashes[2]



[1] Alessandro Marchetto, Fondazione Bruno Kessler, "GUI-based Testing
[2] Techopedia, "Graphical User Interface Testing (GUI Testing),"
https://www.techopedia.com/definition/29846/graphical-user-interface-testing-gui-testing

Why Is GUI Testing Important?

- **Think as a user, not a tester**
  - It is the user interface (UI) of the application, which decides that a user is going to use the app further or not[1].

Unit testing

- Unit testing relies on automated tests written by developers.
- Each test targets individual units of source code or a narrow aspect of application behavior.

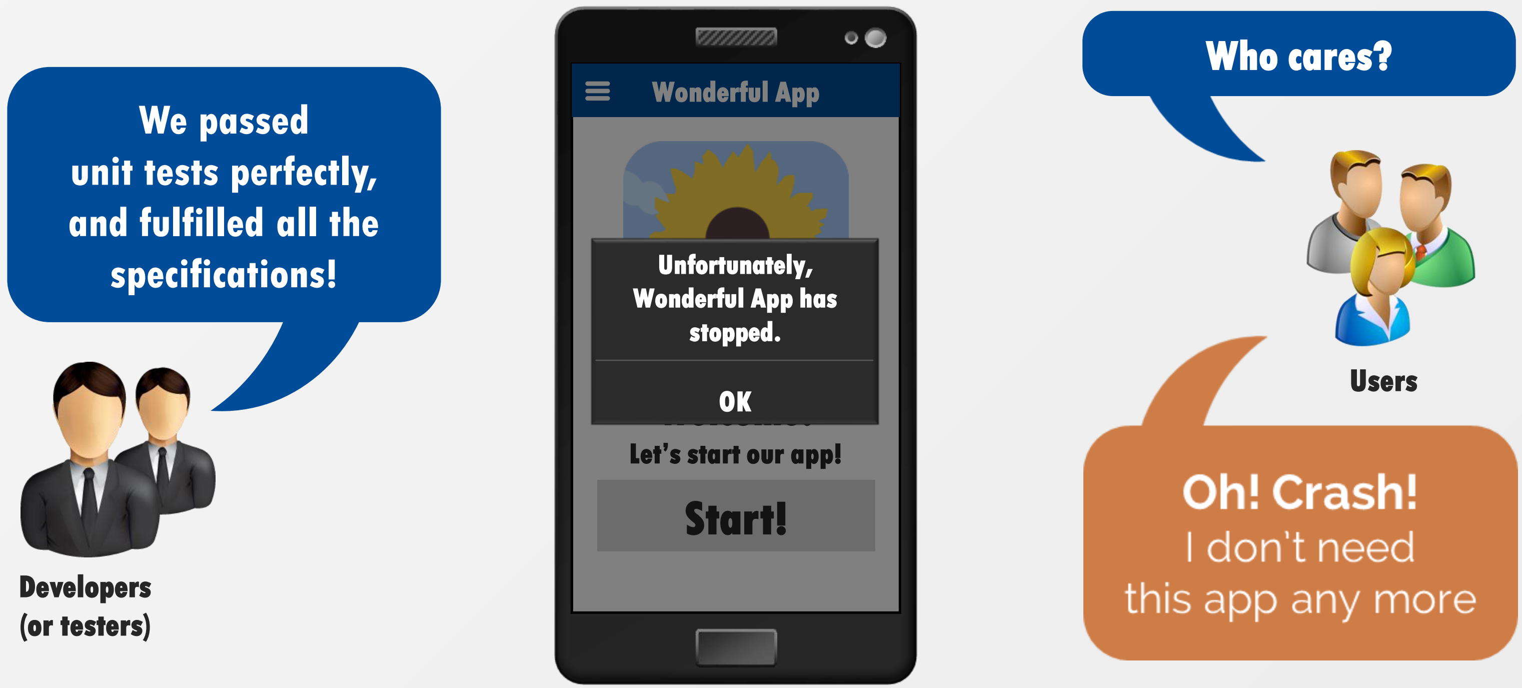
GUI testing

- Functional testing is performed by QA personnel or through automated UI testing framework.
- GUI testing performs the test processes like a user.

[1] Testomato, "What Is UI Testing and Why Is It Important?“ https://blog.testomato.com/what-is-ui-testing/

Why Is GUI Testing Important?

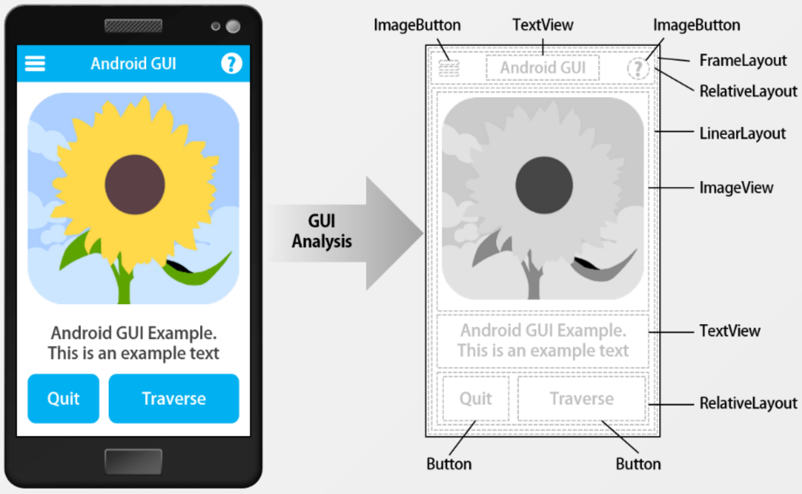
- **Think as a user, not a tester**
  - It is the user interface (UI) of the application, which decides that a user is going to use the app further or not[1].



[1] Testomato, "What Is UI Testing and Why Is It Important?" https://blog.testomato.com/what-is-ui-testing/

GUI of Mobile Apps (Android)

- An Android GUI is a hierarchical, graphical front-end to an Android app displayed on the foreground of the screen.
  - GUIs accept input events and produce graphical outputs.
    - An Android GUI hierarchically consists of specific types of graphical objects called **widgets**; each widget has a fixed set of **properties**; each property has discrete **values** during the execution of the GUI.

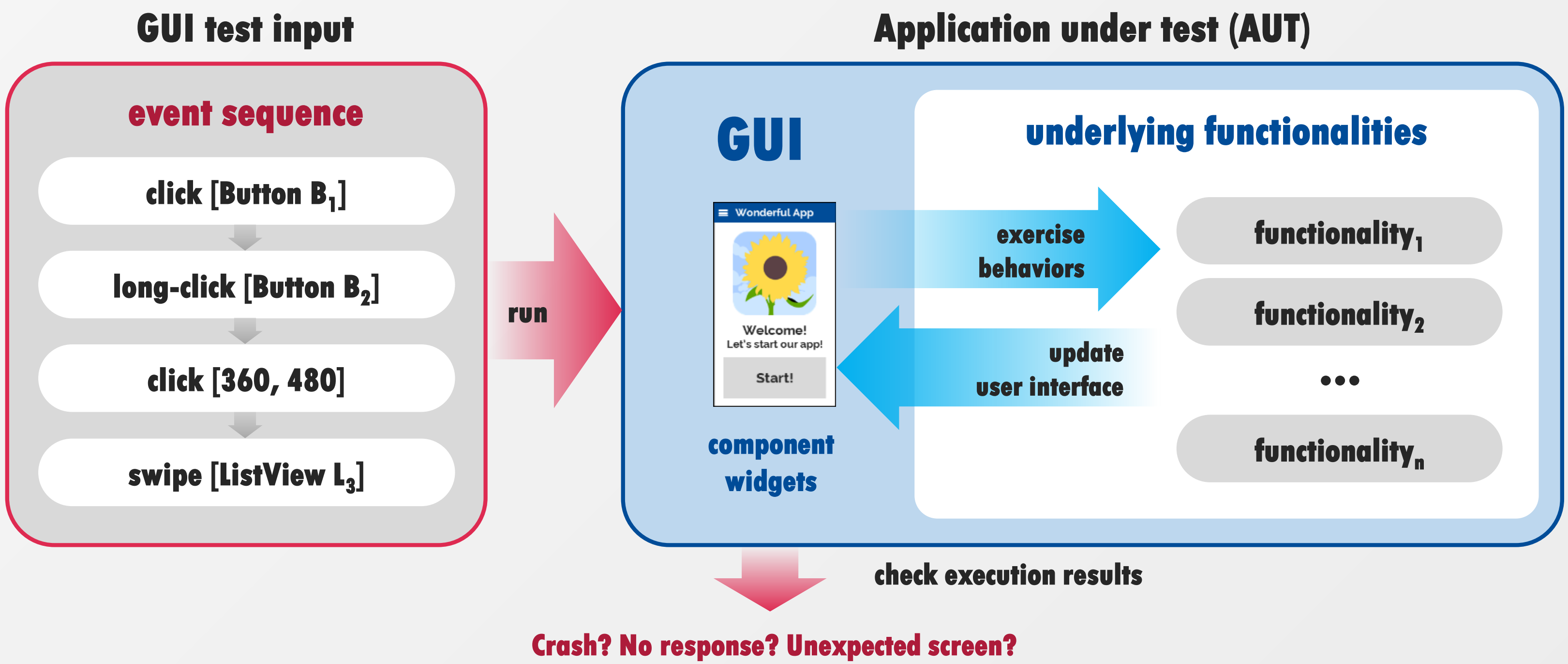


A GUI on Android screen

Hierarchical widget structure

GUI Testing for Android Apps

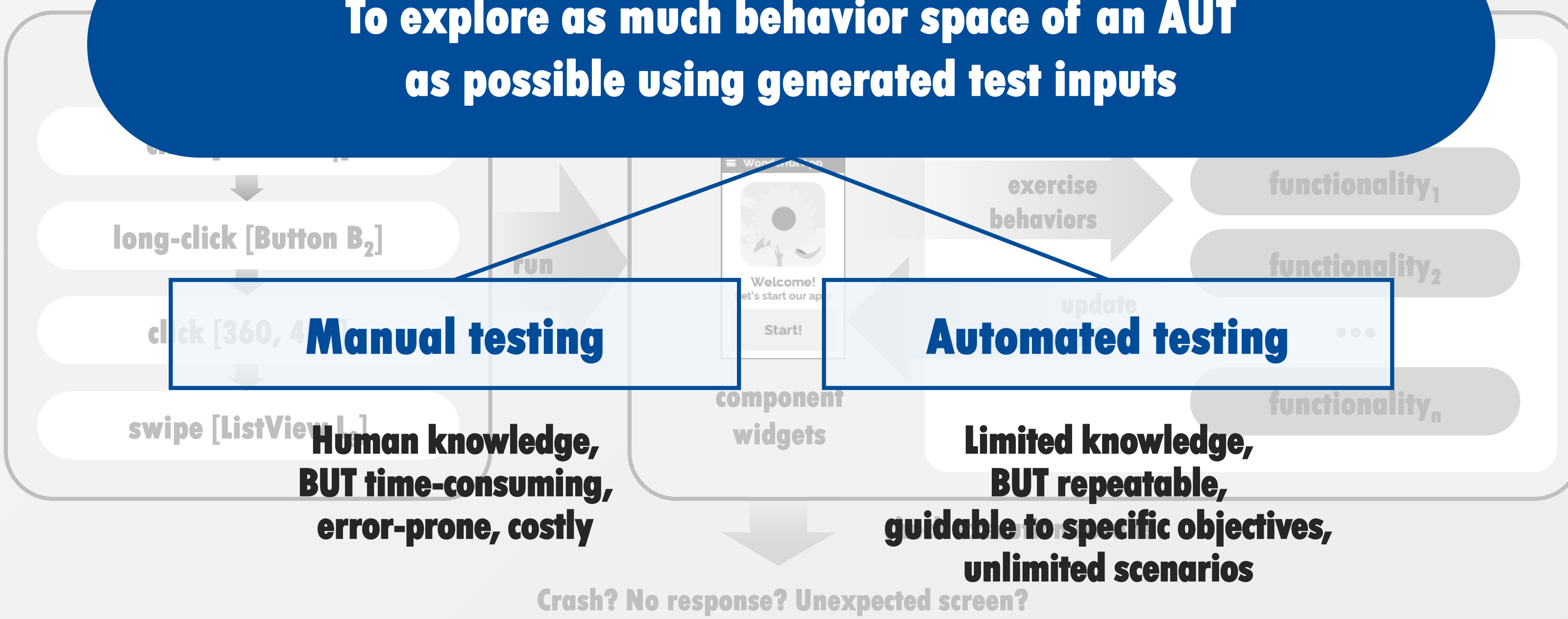
- **GUI testing ensures that the application returns the correct UI output in response to a sequence of user / system actions.**
  - Fault detection, behavior observation, robustness testing



GUI Testing for Android Apps

- **GUI testing ensures that the application returns the correct UI output in response to a sequence of user / system actions.**
  - Fault detection, behavior observation, robustness testing

To explore as much behavior space of an AUT as possible using generated test inputs



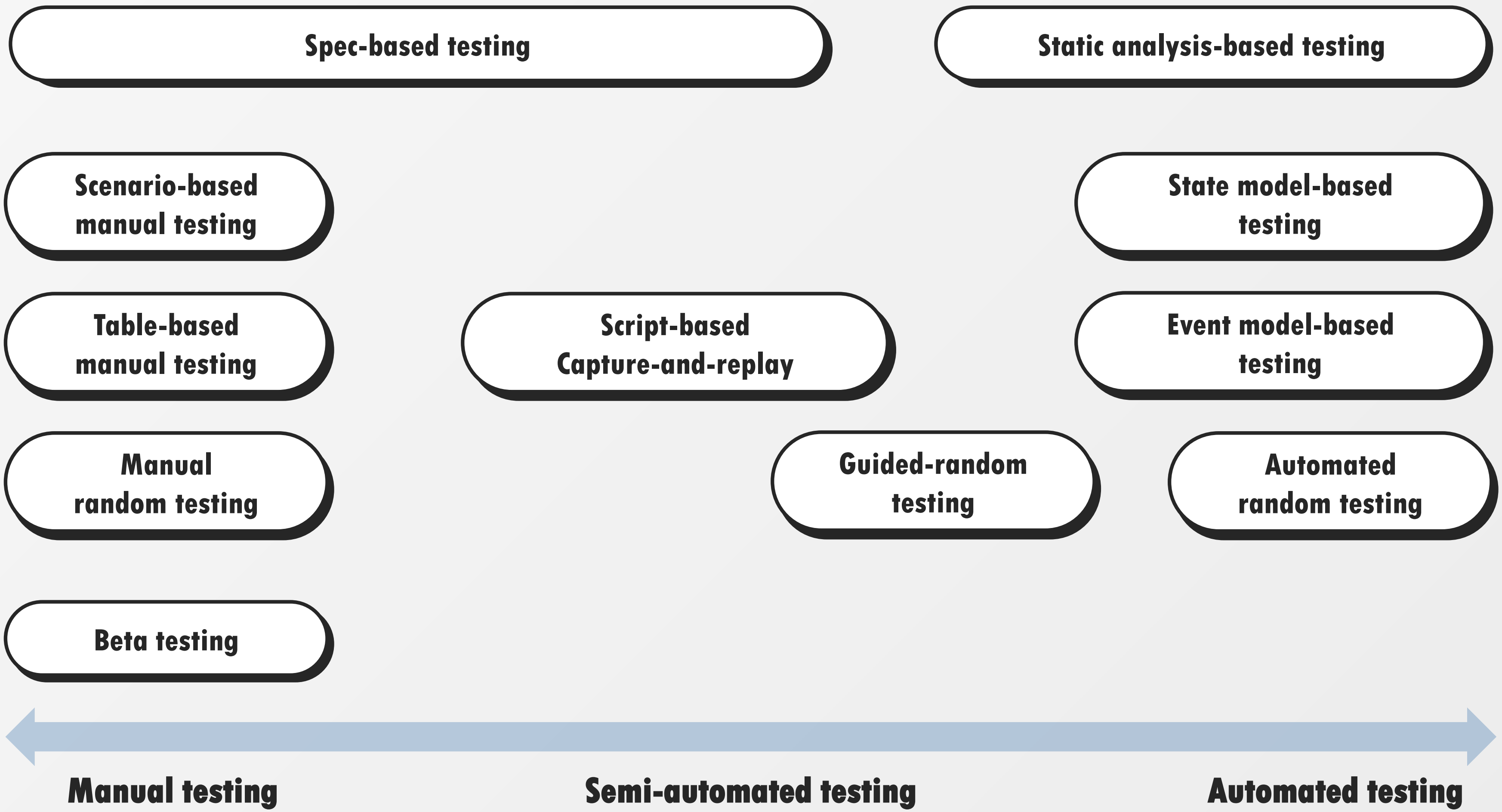
Manual testing

Human knowledge,
BUT time-consuming,
error-prone, costly

Automated testing

Limited knowledge,
BUT repeatable,
guidable to specific objectives,
unlimited scenarios

Existing GUI Testing Methods for Android Apps



GUI Testing Approaches

- **Generating GUI test cases**
  - A test input contains sequences of GUI events, completed with concrete inputs and expected oracles.

Black-box

- Internal structure is not known to the tester.
- Static information (source code, dependencies, relationships) is not used.

Grey-box
(Black-box + White-box)

- Internal structure is partially known to the tester.
- Testing is performed at the user, or black-box level.

White-box

- Internal structure, design, implementation is known to the tester.
- Generally, programming and implementation knowledge is required.

Methods to generate GUI test inputs

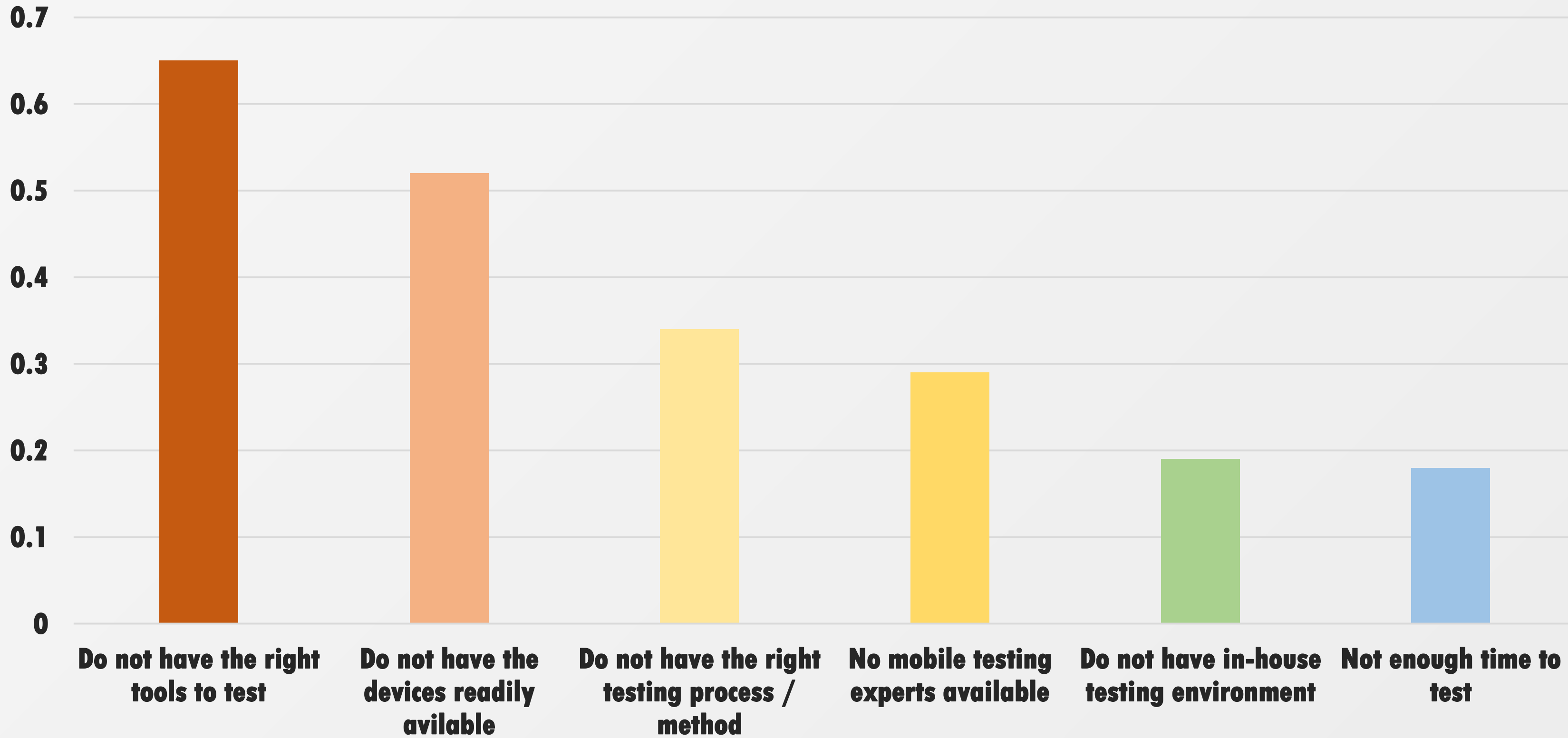
What Errors Can Occur in (Android) GUI?

- **Examples of GUI errors**
  - Incorrect functioning, including crash errors
  - Missing commands
  - Incorrect GUI screenshots or states
  - Absence of mandatory UI components
  - Incorrect default values for fields or UI objects
  - Data validation errors
  - Incorrect error handling
  - Compatibility among different smart devices
    (Positioning of GUI elements for different screen resolution)
  - Poor usability

Challenges of
(Model-based) GUI Testing for Mobile Apps

Overview of Challenges for Mobile App Testing

- **Challenges to mobile testing**[1,2]

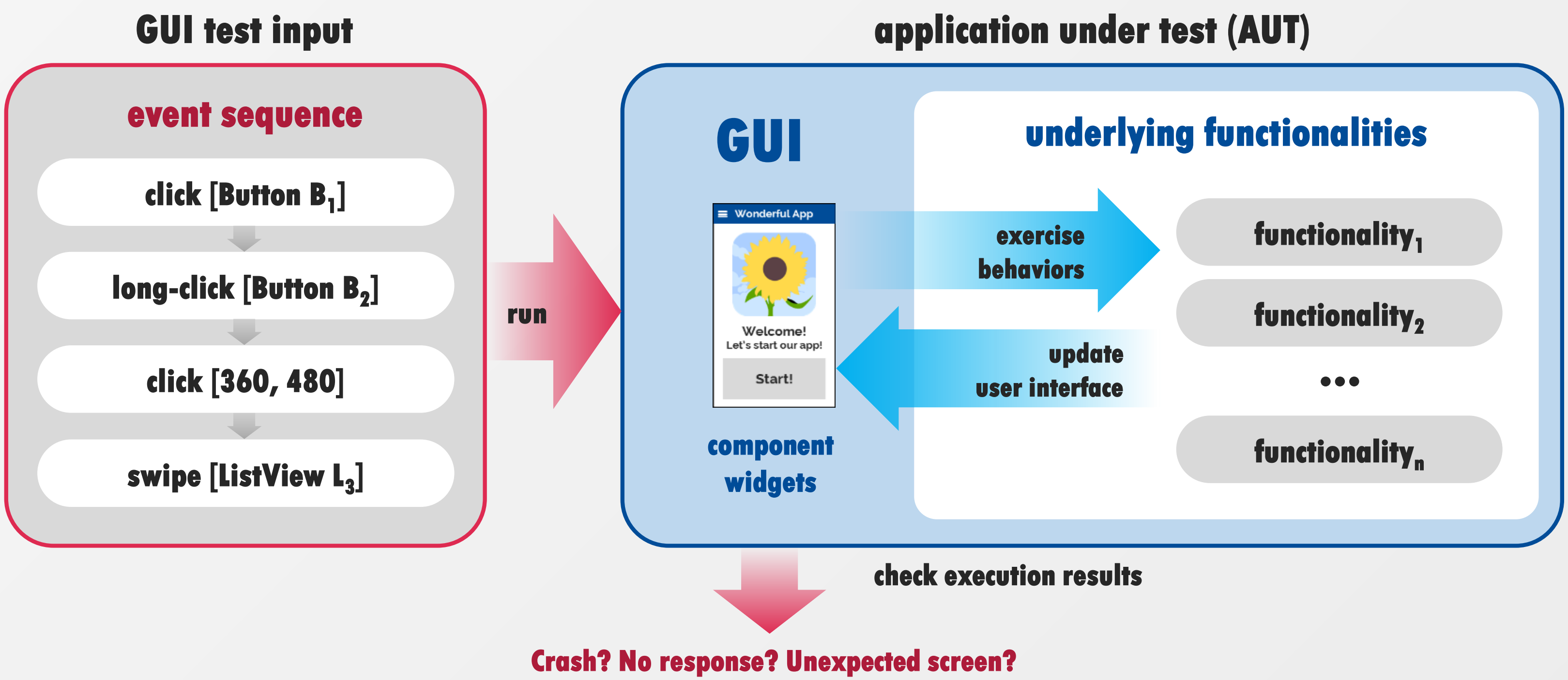


[1] Capgemini, "World Quality Report 2013-14 Mobile Testing Pull Out," http://www.capgemini.com/resources/world-quality-report-2013-14-mobile-testing-pull-out
[2] Software Testing Help, "5 Mobile Testing Challenges and Solutions," http://www.softwaretestinghelp.com/5-mobile-testing-challenges-and-solutions/

SELAB KAIST

Model-based GUI Testing (1/2)

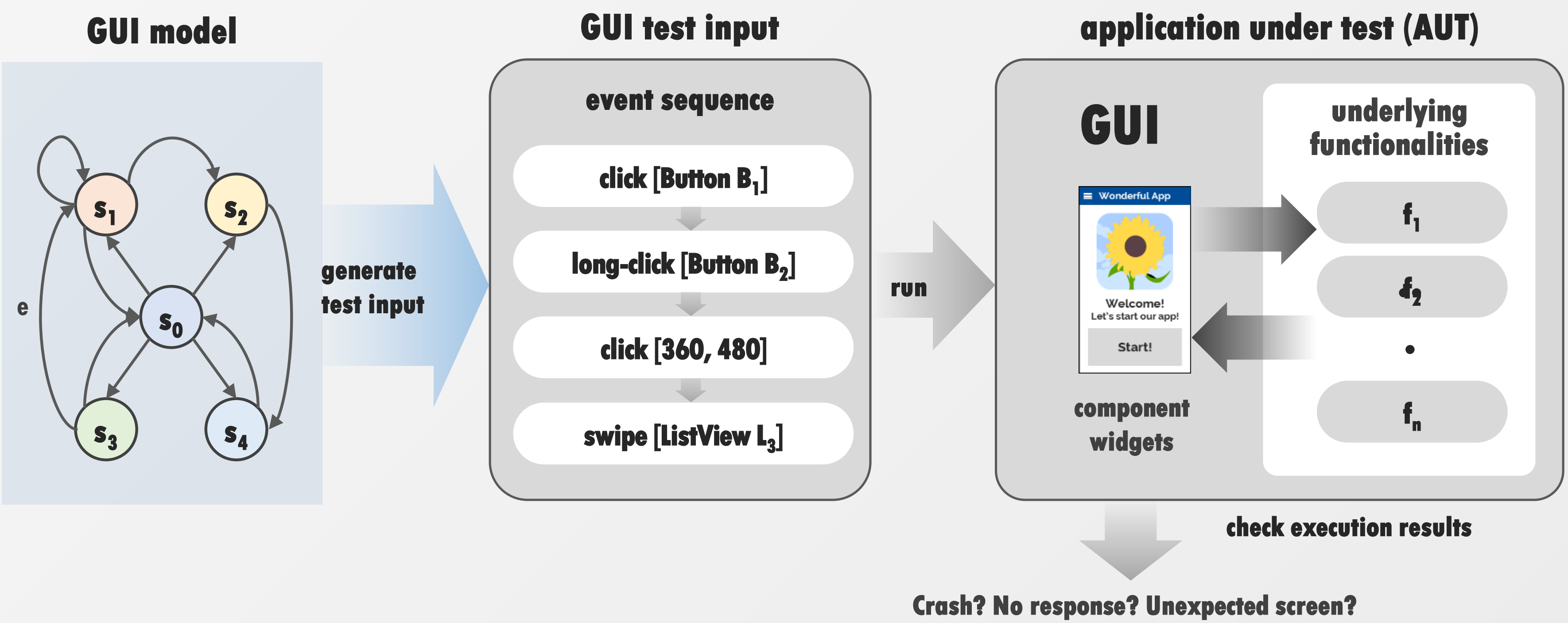
- **Generic GUI testing framework**
  - GUI testing exercises the behavior space of an application under test (AUT) as a user.



Reference

Model-based GUI Testing (2/2)

- **Model-based GUI testing**
  - Utilize a model to guide test input generation and limit the search space for the systematic test[1]



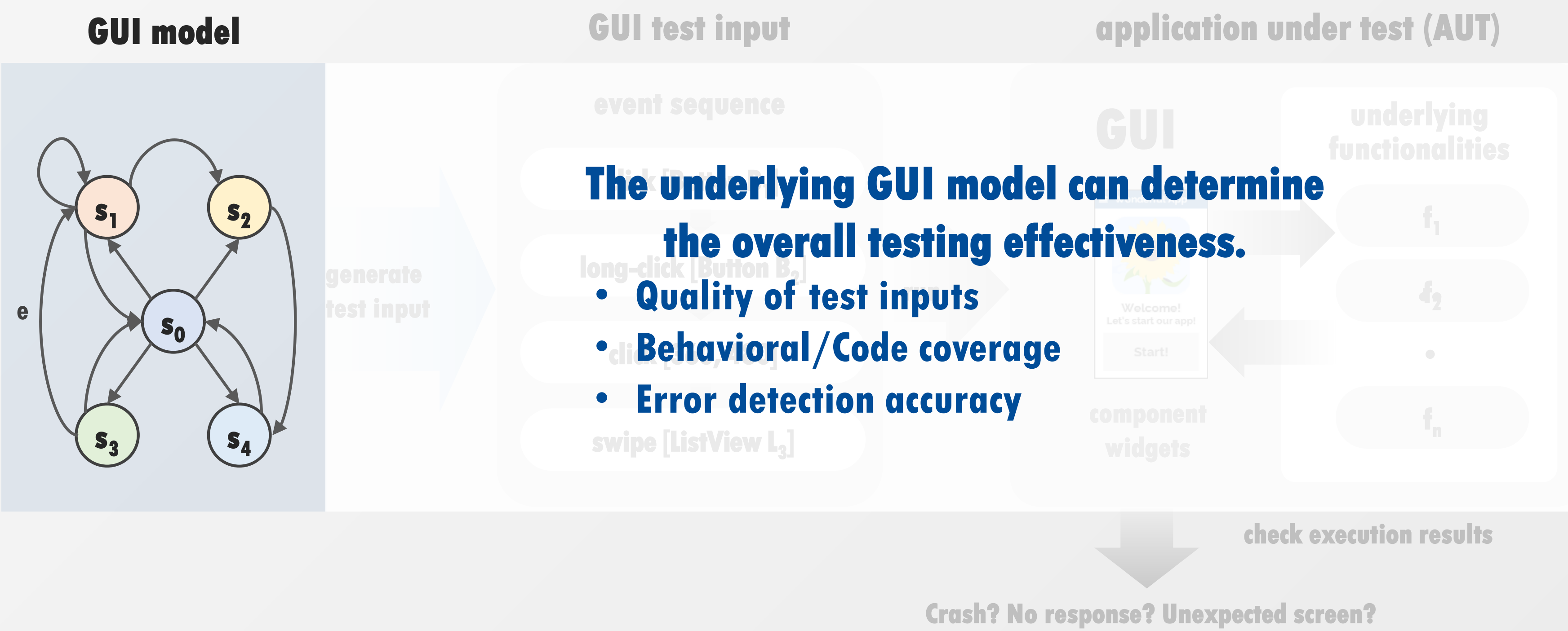
[1] S. R. Choudhary, et al., "Automated Test Input Generation for Android: Are We There Yet?," ASE '15 Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

Model-based GUI Testing (2/2)

Model-based GUI testing

- Utilize a model to guide test input generation and limit the search space for the systematic test[1]

GUI model

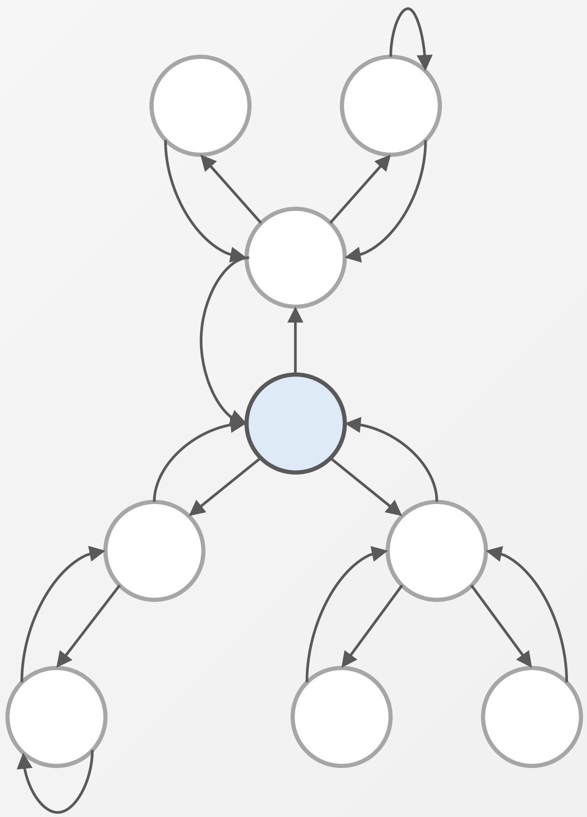


The underlying GUI model can determine the overall testing effectiveness.

- Quality of test inputs
- Behavioral/Code coverage
- Error detection accuracy

[1] S. R. Choudhary, et al., "Automated Test Input Generation for Android: Are We There Yet?," ASE '15 Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

How to Build a GUI Model?

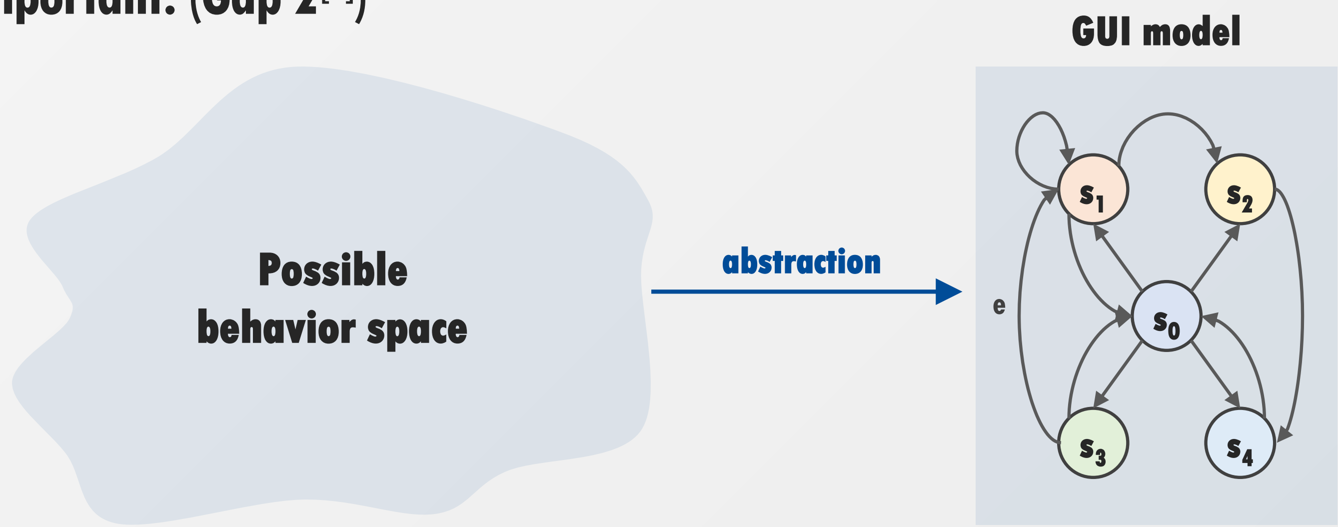


- **Manual-based**
  - Build a GUI model based on the specification by experts
- **Program Analysis**
  - Build a GUI model statically from app source code
- **Random-based**
  - Build a GUI model from random app executions
- **GUI Ripping**
  - Learn and build a GUI model dynamically from interactive app executions and their traces

[1] S. R. Choudhary, et al., "Automated Test Input Generation for Android: Are We There Yet?," ASE '15 Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

The Gap in Automated GUI Model Generation

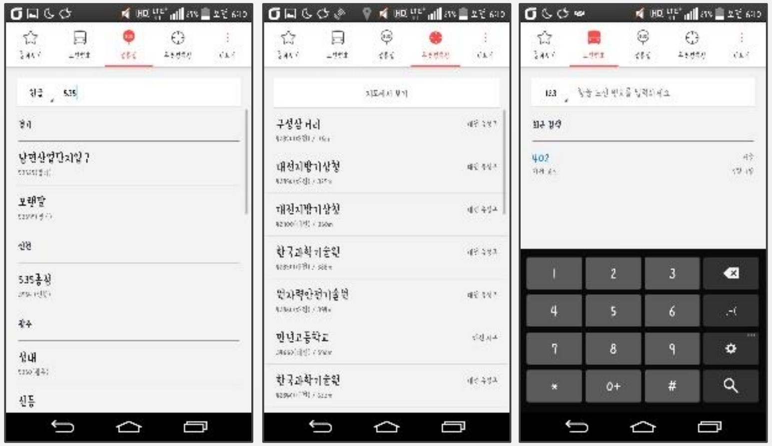
- **How much do we have to abstract an AUT's behaviors?**
  - Because most Android apps do not have their own GUI models (specifications) beforehand, we often have to build a GUI model through reverse-engineering (GUI ripping).[1]
  - Reaching a sufficient coverage in a reasonable time for model extraction is important. (Gap 2[2])



[1] D. Amalfitano, et al. "Using gui ripping for automated testing of android applications". In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, 2012. ACM.
[2] P. Aho, M. Suarez, A. Memon, and T. Kanstrén, "Making GUI Testing Practical: Bridging the Gaps," Information Technology – New Generations (ITNG), 2015.

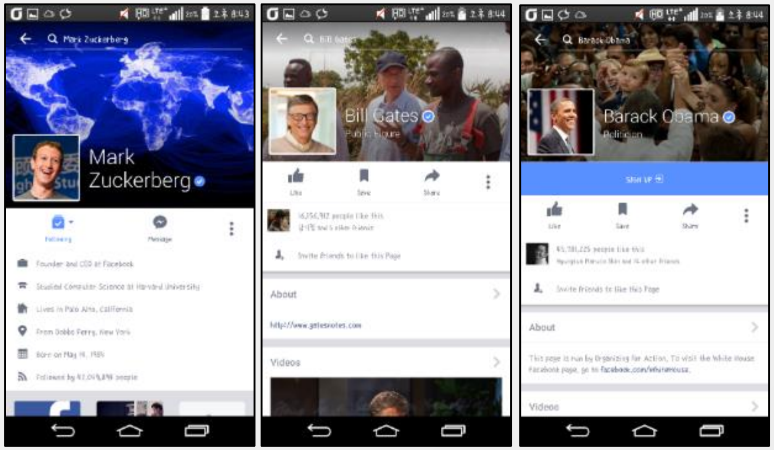
Challenges: Fancy but Volatile Apps

- **Characteristics of recent mobile apps (1/2)**
  - Recent apps have a number of dynamic pages, non-deterministic (context-sensitive) GUIs.



Multiple dynamic pages of a single screen (Seoulbus app)

ViewPager widget is frequently used
to provide multiple view pages in a single activity.

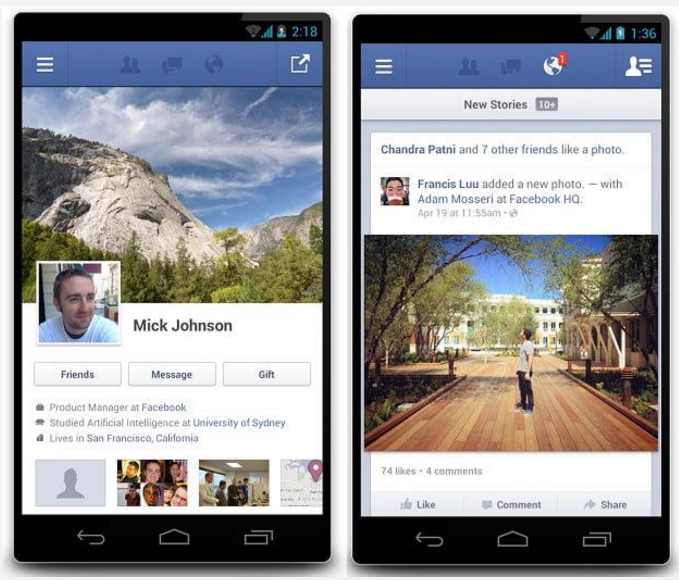


Context/Data-sensitive GUIs (Facebook app)

Timeline and personal newspeed are totally dependent
on user's and facebook friends' data or context.

Challenges: Fancy but Volatile Apps

- **Characteristics of recent mobile apps (2/2)**
  - Practical testing tools are needed to apply testing techniques into industrial app development.
    - "That's why we are still doing manual or random-based testing"



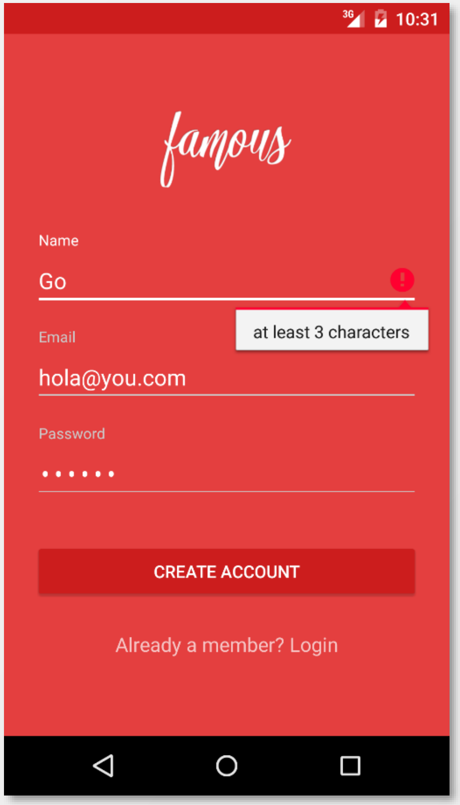
Facebook Android app

- How to define the behavior space?
- How to model this complicated app?
- How to classify app's bugs/faults?
- How to deal with unpredictable data?
- Until when to finish testing?

*https://icdn2.digitaltrends.com/image/facebook-android-2-0-625x1000.jpg

Challenges: Test Input Generation (1/2)

- **Infinite number of possible combinations**
  - Difficulties in identifying meaningful test input combinations
- **Sophisticated GUI test inputs**
  - Inter-related, synchronized, inter-dependent
  - Test inputs requiring personal information/data to access certain screens
    - E.g., text input for sign-up: Valid ID, password, e-mail are required



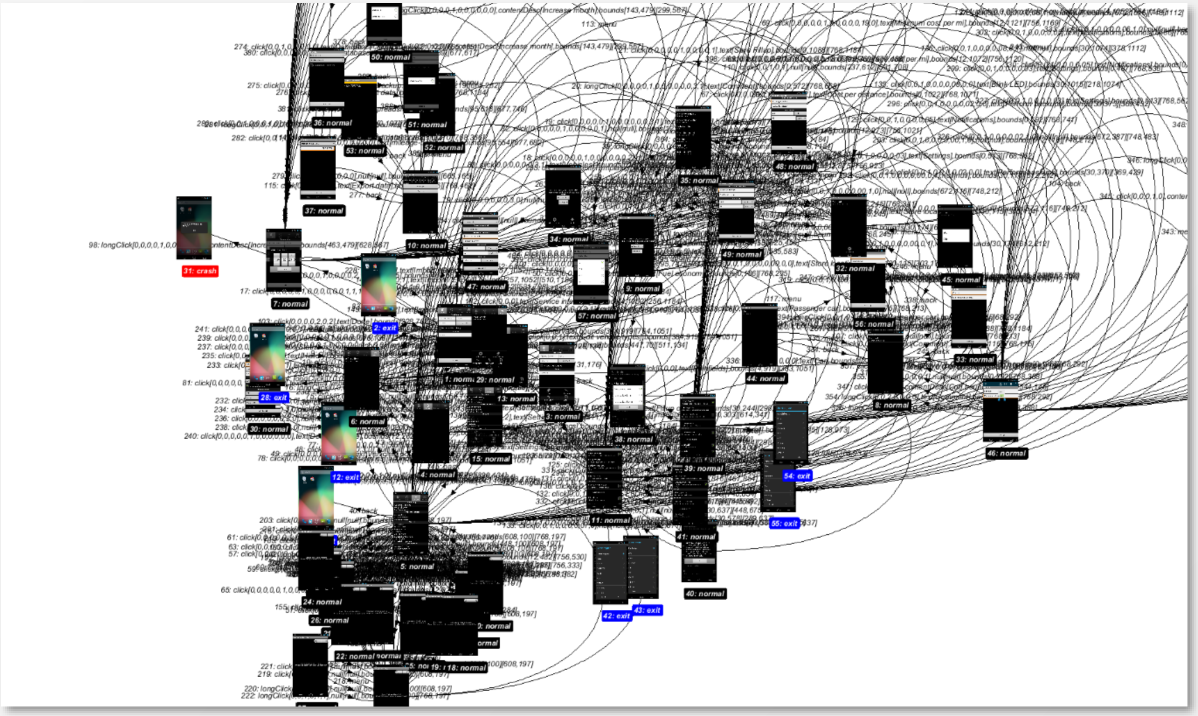
https://sourcey.com/beautiful-android-logn-and-signup-screens-with-material-design/screenshot-signup.png

Challenges: Test Input Generation (2/2)

- **Stateful GUIs & Context-dependent behavior**
  - A single GUI state can contain lots of contextual information, and test results may not be reproducible if the context is not met.

Challenges: Model Generation

- **UI state explosion problem**
  - Even a simple app can contain a large number of UI states
    - Combinatorial explosion due to a number of branching and choices
  - Test input selection, prioritization, pruning is required.



Challenges: GUI Testing Process

- **Test results analysis**
  - Conventional code-based coverage cannot be adequate.
    - GUIs are implemented in terms of event-based system, hence, the abstraction level is different with respect to the conventional system code.
    - So, mapping between GUI events and system code cannot be easy.
  - Coverage criteria for adequacy evaluation
    - Types: code coverage, state coverage, event coverage
      - ✓ Event coverage: All events of the GUI need to be executed at least once
        - → Event-pair coverage, Event-triple coverage
      - ✓ State coverage: All states of the GUI need to be exercised at least once
      - ✓ Functionality coverage: Using a functional point of view
    - The coverage criteria are not commonly used
      - ✓ Difficult to compare with each other

Challenges: Test Effectiveness & Performance

- **A comparative study among GUI testing methods is required, but it is not easy to conduct the experiment fairly.**
  - Overview of existing test input generation tools for Android [1]

Name	Available	Instrumentation		Events		Exploration strategy	Needs source code	Testing strategy
		Platform	App	UI	System			
Monkey [23]	✓	×	×	✓	×	Random	×	Black-box
Dynodroid [17]	✓	✓	×	✓	✓	Random	×	Black-box
DroidFuzzer [35]	✓	×	×	×	×	Random	×	Black-box
IntentFuzzer [28]	✓	×	×	×	×	Random	×	White-box
Null IntentFuzzer [24]	✓	×	×	×	×	Random	×	Black-box
GUIRipper [1]	✓[a]	×	✓	✓	×	Model-based	×	Black-box
ORBIT [34]	×	×	×	✓	×	Model-based	✓	Grey-box
A^3E -Depth-first [5]	✓	×	✓	✓	×	Model-based	×	Black-box
SwiftHand [7]	✓	×	✓	✓	×	Model-based	×	Black-box
PUMA [12]	✓	×	✓	✓	×	Model-based	×	Black-box
A^3E -Targeted [5]	×	×	✓	✓	×	Systematic	×	Grey-box
EvoDroid [18]	×	×	✓	✓	×	Systematic	×	White-box
ACTEve [3]	✓	✓	✓	✓	✓	Systematic	✓	White-box
JPF-Android [31]	✓	×	×	✓	×	Systematic	✓	White-box

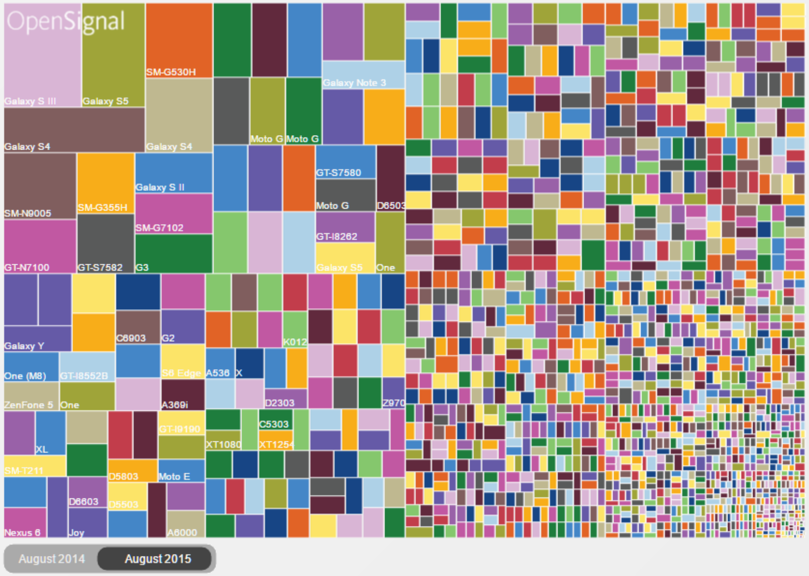
[1] S. R. Choudhary, et al., "Automated Test Input Generation for Android: Are We There Yet?," ASE '15 Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

SELAB KAIST

Challenges: Compatibility & Fragmentation (1/2)

Android **fragmentation**

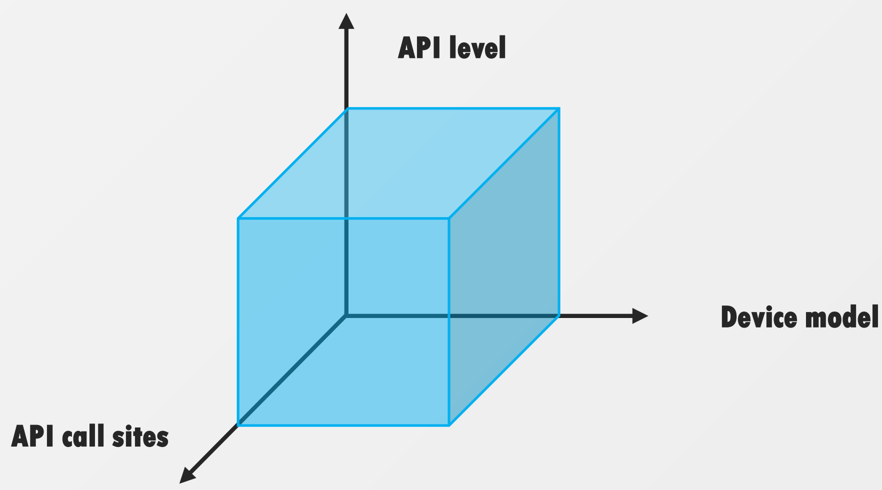
- Fragmentation has been a contentious issue in Android[1]
  - Different version of operating systems
  - Different version of smart devices, device types, resolutions
  - Different types of mobile apps: Native, hybrid, web



[1] XDA Developers, "The Sorry State of Android Fragmentation: An Example to Understand Developers' Plight," https://www.xda-developers.com/the-sorry-state-of-android-fragmentation/

Challenges: Compatibility & Fragmentation (2/2)

- **Fragmentation in smart platforms due to fast-evolving mobile platform system** [1]
  - App developers need to consider diversified screen sized for their user interface to be developed.
  - However, an application can behave differently across OSs and devices.



[1] Lili Wei et al., "Taming Android Fragmentation: Characterizing and Detecting Compatibility Issues for Android Apps," Automated Software Engineering (ASE) 2016, Singapore.

Challenges: Non-functional Requirements

- **How to test non-functional requirements[1]**
  - Usability, Accessibility
  - Responsiveness, Performance
  - Reliability, Security
  - Modifiability
  - Maintainability

- **A new validation method for non-functional requirements at GUI level is required.**
  - Since GUI is what user only sees, nonfunctional requirements should be satisfied at GUI level as well.

[1] http://www.dummies.com/web-design-development/mobile-apps/basics-of-nonfunctional-requirements-for-ios-apps/

Other Challenges / Required Features

- Performance & Scalability
- More shift left testing[2]
- Network virtualization testing[2] / Network bypass[1]
- Cloud testing[2]
- Emergence of big data[2]
- Continuous testing as a part of continuous integration[2]
- Parallel testing[2]
- Regression testing
- Test oracle generation
- Industrial standards[1]

[1] The Official 360Logica Blog, "Challenges faced in Mobile App Testing,"
http://www.360logica.com/blog/challenges-faced-in-mobile-app-testing/
[2] Guy Arieli, "11 Challenges for Mobile Testing in 2016," Experitest Blog, 2016.
https://experitest.com/blog/blog-cat/11-challenges-for-mobile-testing-in-2016-blog/

Our Empirical Study

Y. M. Baek, D. H. Bae,

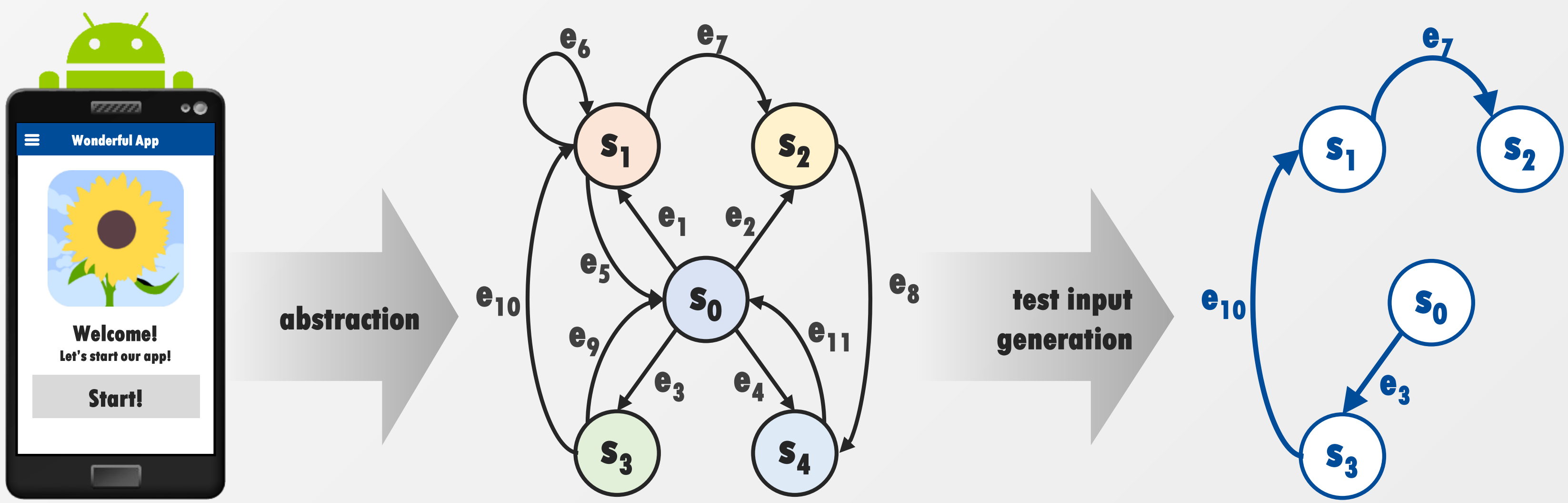
"Automated Model-based Android GUI Testing using Multi-level GUI Comparison Criteria," ASE '16

Automated Model-based Android GUI Testing
using Multi-level GUI Comparison Criteria

Introduction

How Can We Model an Android App?

- **State-based models for Model-Based GUI Testing (MBGT)** *,**
  - Model AUT's GUI states and transitions between the states
  - Generate test inputs, which consist of sequences of events, considering stateful GUI states of the model



application under test (AUT)

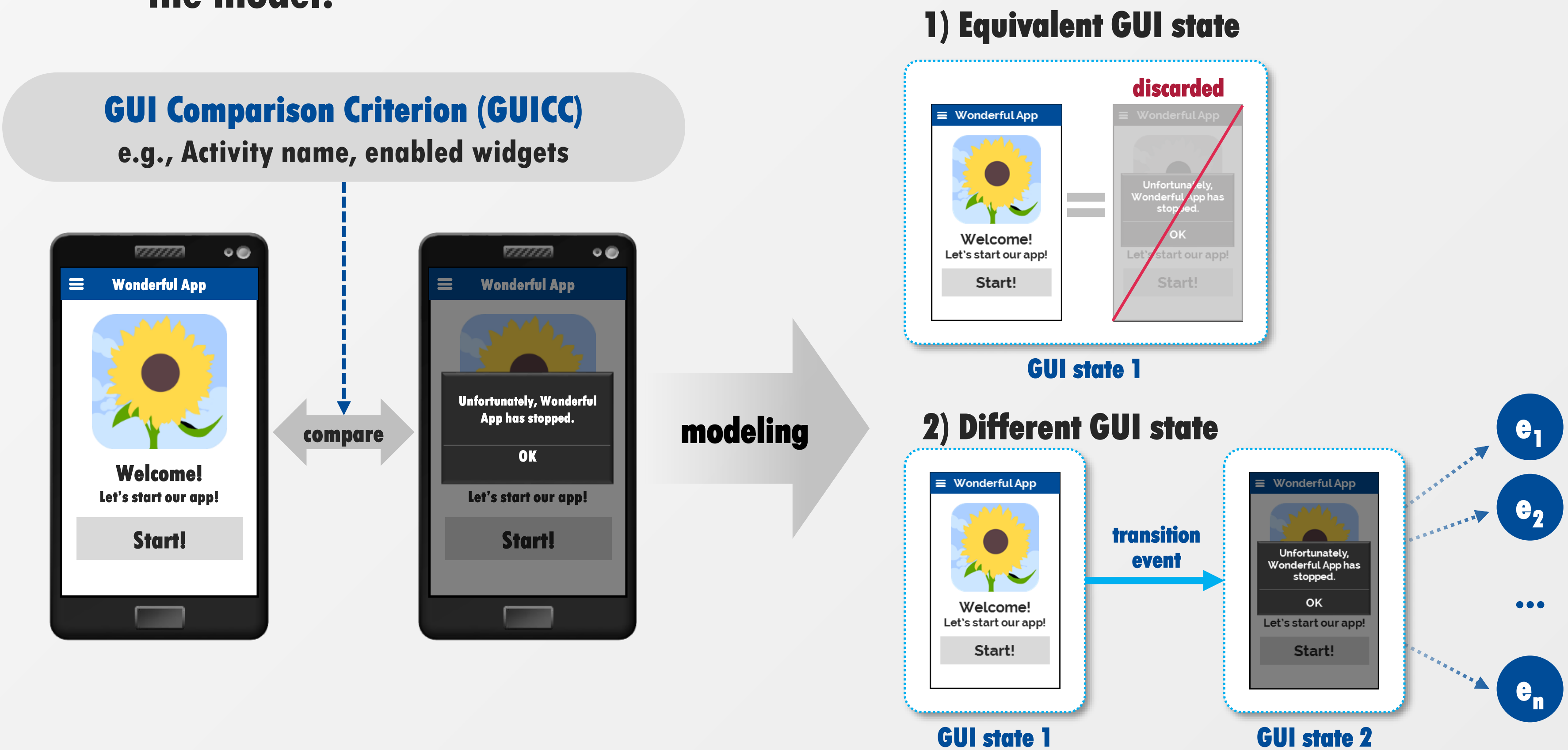
GUI model
5 states, 11 transitions

test input
$e_3 \rightarrow e_{10} \rightarrow e_7$

* D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated Model-Based Testing of Mobile Apps," IEEE Software, vol. 32, issue 5, pp 53-59, 2015.
** T. Azim and I. Neamtiu, "Targeted and Depth-first Exploration for Systematic Testing of Android Apps," OOPSLA 2013.

Define GUI States of a GUI Model

- **A GUI Comparison Criterion (GUICC)**
  - A GUICC distinguishes the equivalence/difference between GUI states to update the model.

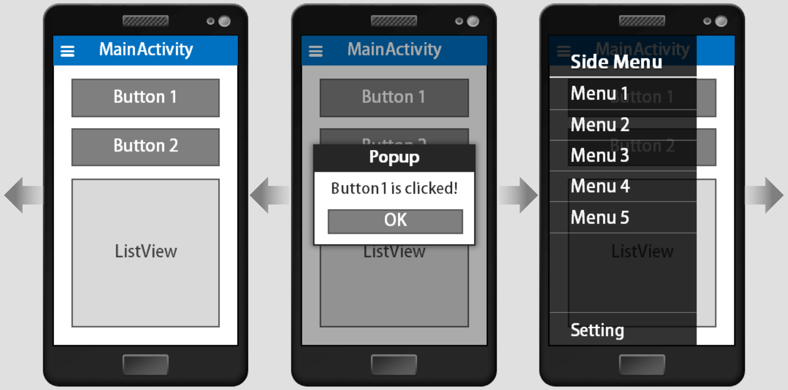


Influence of GUICC on Automated MBGT

- **GUICC determines the effectiveness of MBGT techniques.**

Weak GUICC for Android

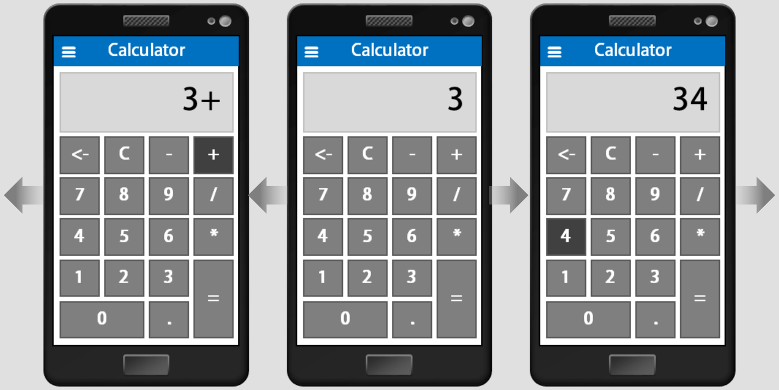
Activity name



MainActivity MainActivity MainActivity

Strong GUICC for Android

Observable graphical change



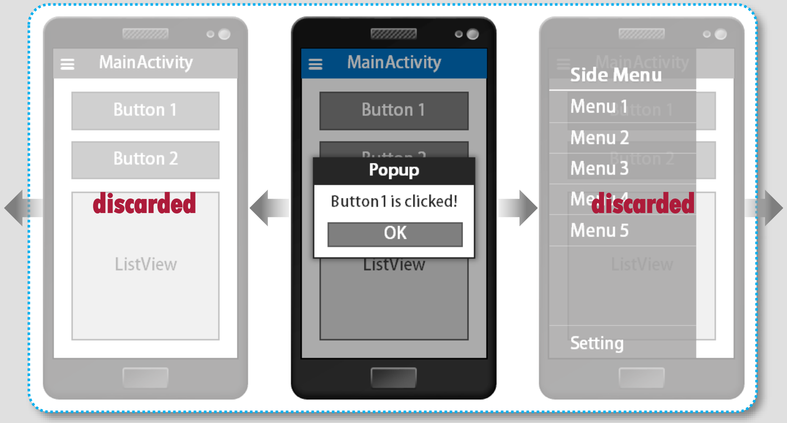
* P. Aho, M. Suarez, A. Memon, and T. Kanstrén, "Making GUI Testing Practical: Bridging the Gaps," Information Technology – New Generations (ITNG), 2015.

Influence of GUICC on Automated MBGT

- **GUICC determines the effectiveness of MBGT techniques.**

Weak GUICC for Android

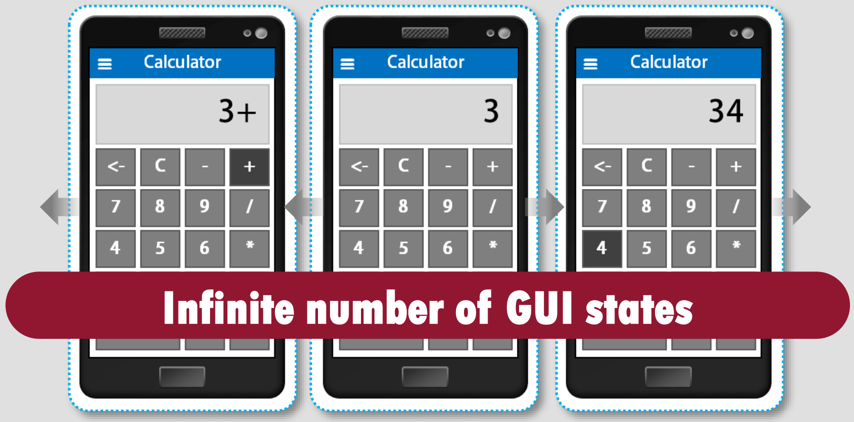
Activity name



state 1 (MainActivity)

Strong GUICC for Android

Observable graphical change



state 1 **state 0** **state 2**

MBGT techniques should carefully consider the GUICC for model generation

SELAB KAIST

* P. Aho, M. Suarez, A. Memon, and T. Kanstrén, "Making GUI Testing Practical: Bridging the Gaps," Information Technology – New Generations (ITNG), 2015.

GUI Comparison Criteria (GUICC) for Android

Used GUICC by existing MBGT tools for Android apps

1st Author	Venue/Year	Tool	Type of model	GUICC
M. L. Vasquez	MSR/2015	*MonkeyLab*	statistical language model	Activity
L. Ravindranath	MobiSys/2014	*VanarSena*	OCR-based EFG	positions of texts
E. Nijkamp	Github/2014	*SuperMonkey*	state-based graph	Activity
S. Hao	MobiSys/2014	*PUMA*	state-based graph	cosine-similarity with a threshold
D. Amalfitano	ASE/2012	*AndroidRipper*	GUI tree	composition of conditions
				Activity
				enabled GUI elements
				widget values (id, properties)
P. Tonella	ICSE/2014	*Ngram-MBT*	statistical language model	values of class attributes
W. Yang	FASE/2013	*ORBIT*	finite state machine	observable state (structural)
C. S. Jensen	ISSTA/2013	*Collider*	finite state machine	set of event handlers
R. Mahmood	FSE/2014	*EvoDroid*	interface model/ call graph model	composition of widgets

1. They have not clearly explained why those GUICC were selected.
2. They utilize only a fixed single GUICC for model generation, no matter how AUTs behave.

Goal of This Research

Motivation

- Dynamic and volatile behaviors of Android GUIs make accurate GUI modeling more challenging.
- However, existing MBGT techniques for Android:
  - are focusing on improving exploration strategies and test input generation, while GUICC are regarded as unimportant.
  - have defined their own GUI comparison criteria for model generation

Goal

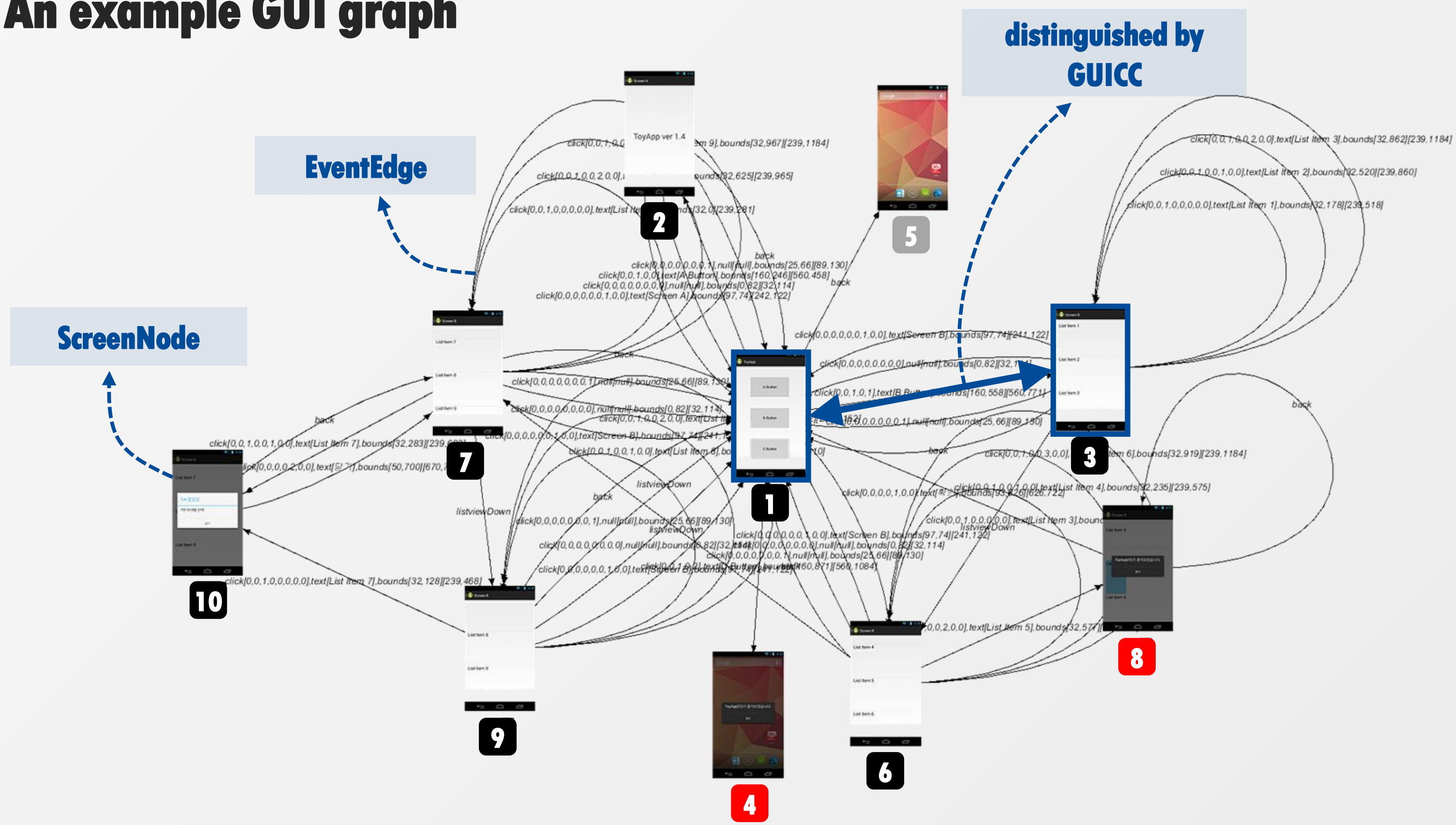
- To conduct empirical experiments **to identify the influence of GUICC on the effectiveness** of automated model-based GUI testing.

Automated Model-based Android GUI Testing using Multi-level GUI Comparison Criteria

GUI Graph

GUI Graph

- **An example GUI graph**



s **Normal state** s **Terminated state** s **Error state**

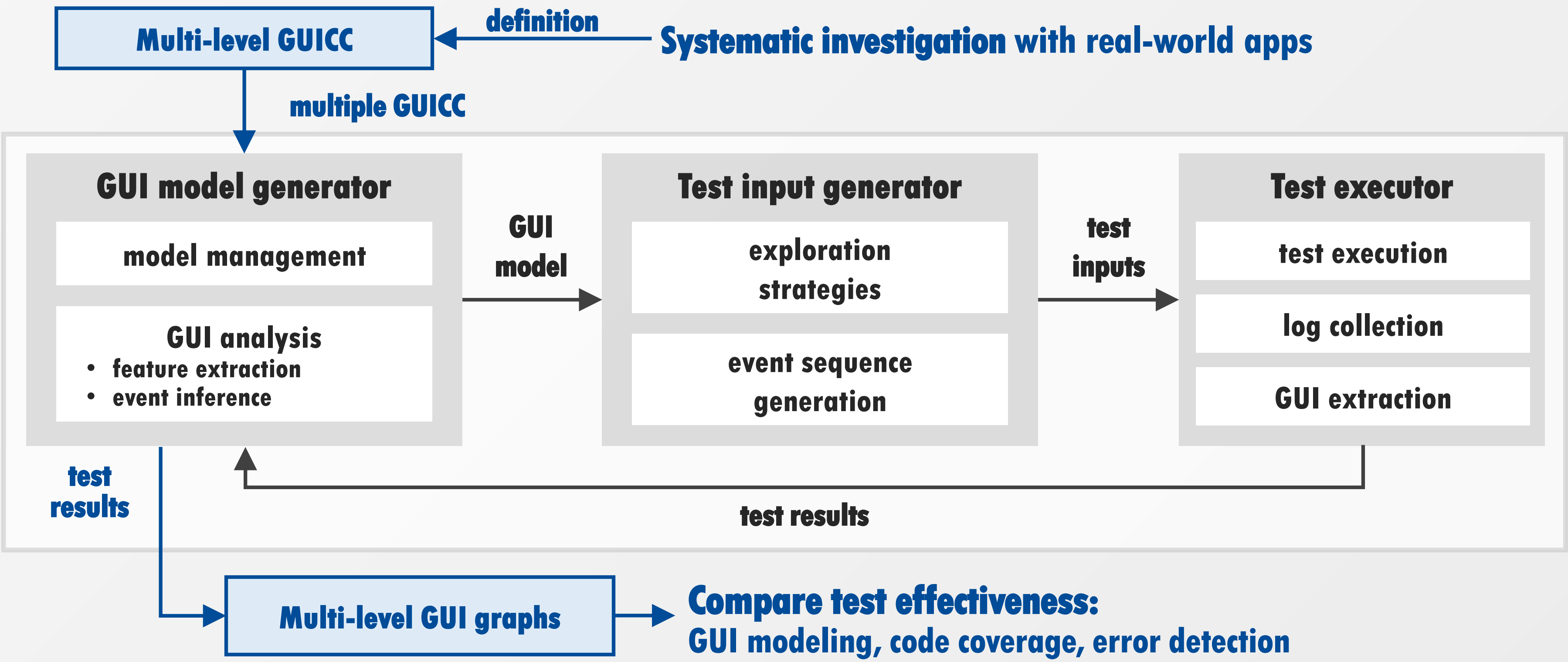
* The generated GUI models are visualized by *GraphStream-Project*, http://graphstream-project.org/

Our Approach

- Overall Approach
- Multi-level GUI Comparison Criteria
- Automated GUI Testing Framework

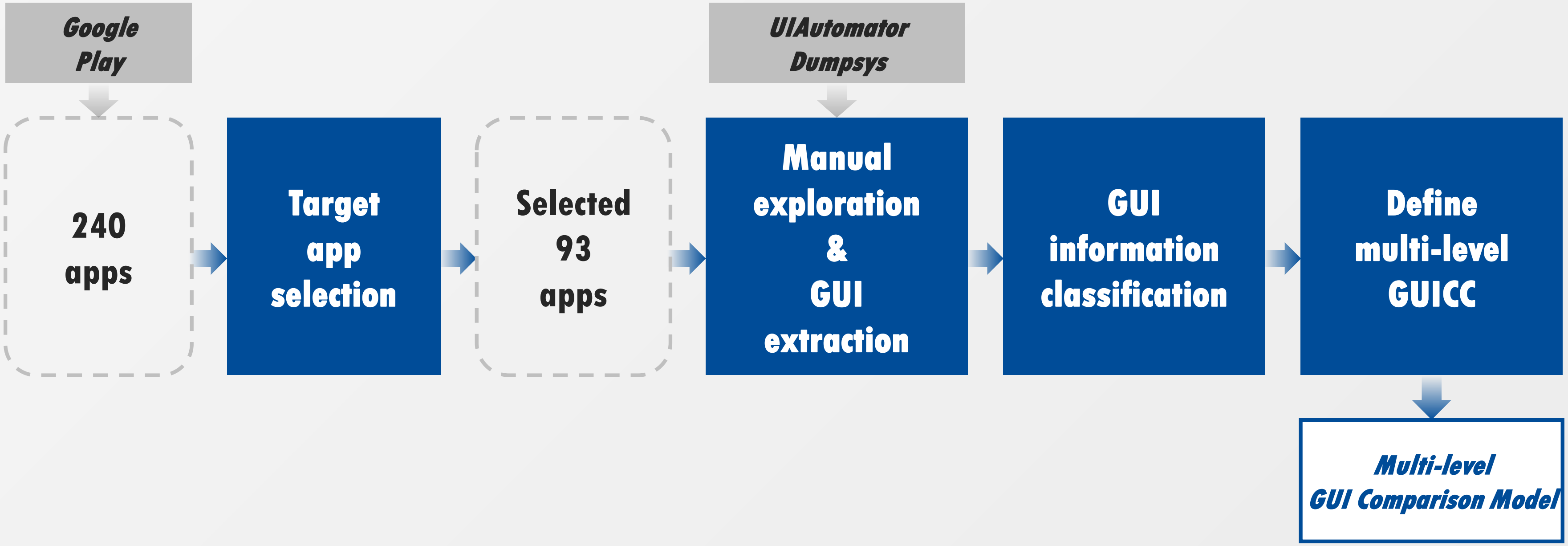
Our Approach

- Automated model-based Android GUI testing using Multi-level GUI Comparison Criteria (Multi-level GUICC)



Design Multi-level GUICC

- **Overview of the investigation on the behaviors of commercial Android applications**
  - The *multi-level GUI comparison technique* was designed based on a semi-automated investigation with 93 real-world Android commercial apps registered in Google Play.



Investigation on GUIs of Android Apps

Step 1. Target app selection

- Collect the 20 most popular apps in 12 categories (Total 240 apps)
  - Exclude <Game> category because they are usually not native apps
  - Exclude apps that were downloaded fewer than 10,000 times
- Finally select 93 target apps

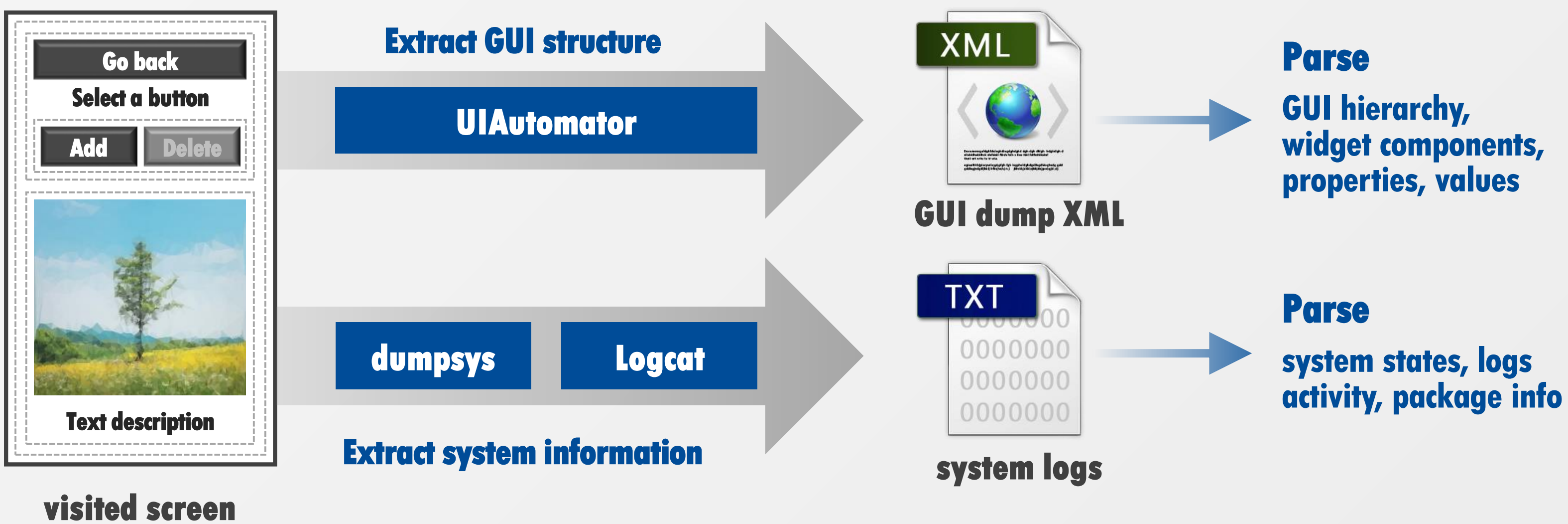
Category	# of apps	Category	# of apps
Book	13	Shopping	9
Business	12	Social	7
Communication	10	Transportation	13
Medical	9	Weather	10
Music	10		

Selected 93 apps for the investigation

Investigation on GUIs of Android Apps

Step 2. Manual exploration

- Manually visit 5-10 main screens of the apps in an end user's view
- Examine the constituent widgets in GUIs of the main screens
  - Use `UIAutomator` tool to analyze the GUI components of the screens
  - Extract system information via `dumpsys` system diagnostics & `Logcat`

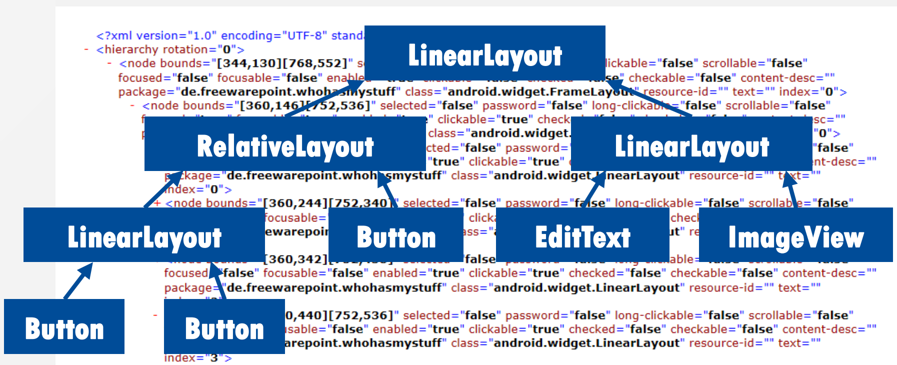


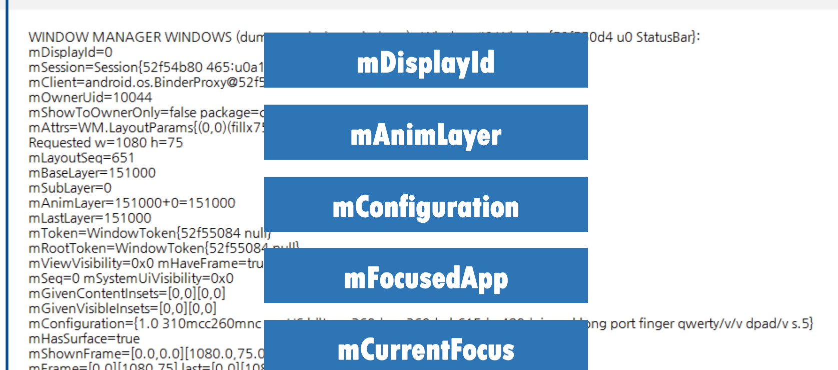
Investigation on GUIs of Android Apps

Step 2. Manual exploration

- Manually visit 5-10 main screens of the apps in an end user's view
- Examine the constituent widgets in GUIs of the main screens
  - Use `UIAutomator` tool to analyze the GUI components of the screens
  - Extract system information via `dumpsys` system diagnostics & `Logcat`

UIAutomator dump	Dumpsys
adb shell /system/bin/uiautomator/ dump /data/local/tmp/uidump.xml	adb shell dumpsys window windows > /data/local/tmp/windowdump.txt

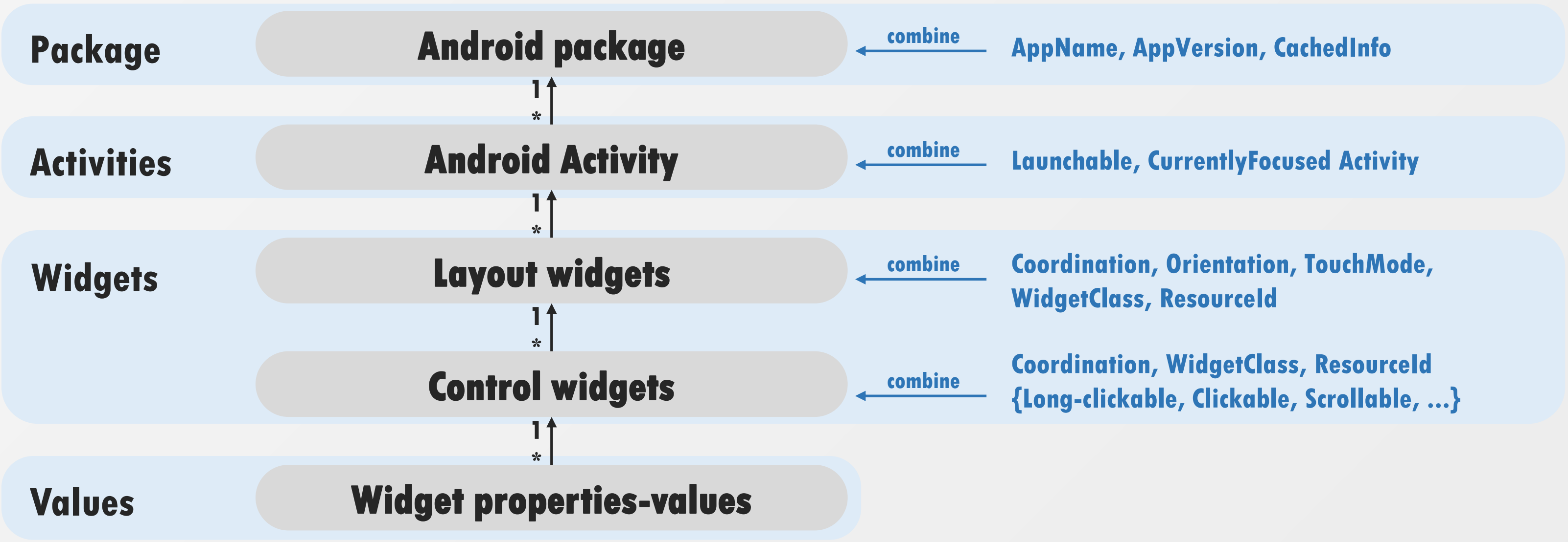




Investigation on GUIs of Android Apps

Step 3. Classification of GUI information

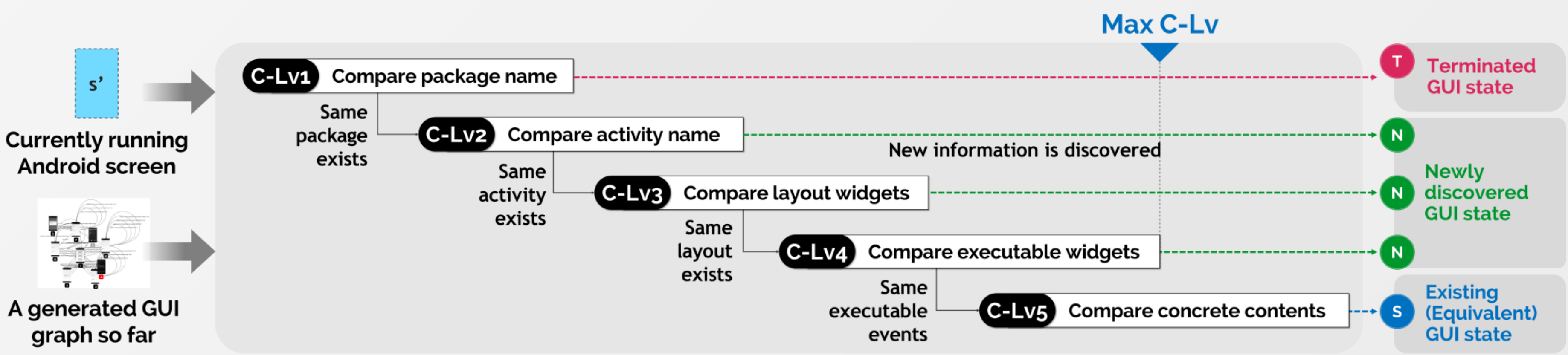
- Find hierarchical relationships from extracted GUI information
- Filter out redundant GUI information that highly depends on the device or the execution environment (e.g., coordinates)
- Merge some GUI information into a single property



Investigation on GUIs of Android Apps

Step 4. Definition of GUICC model

- Design a multi-level GUI comparison model that contains hierarchical relationships among GUI information
  - Define 5 comparison levels (C-Lv)
  - Define 3 types of outputs according to the comparison result
    - T: Terminated state, S: Same state, N: New state

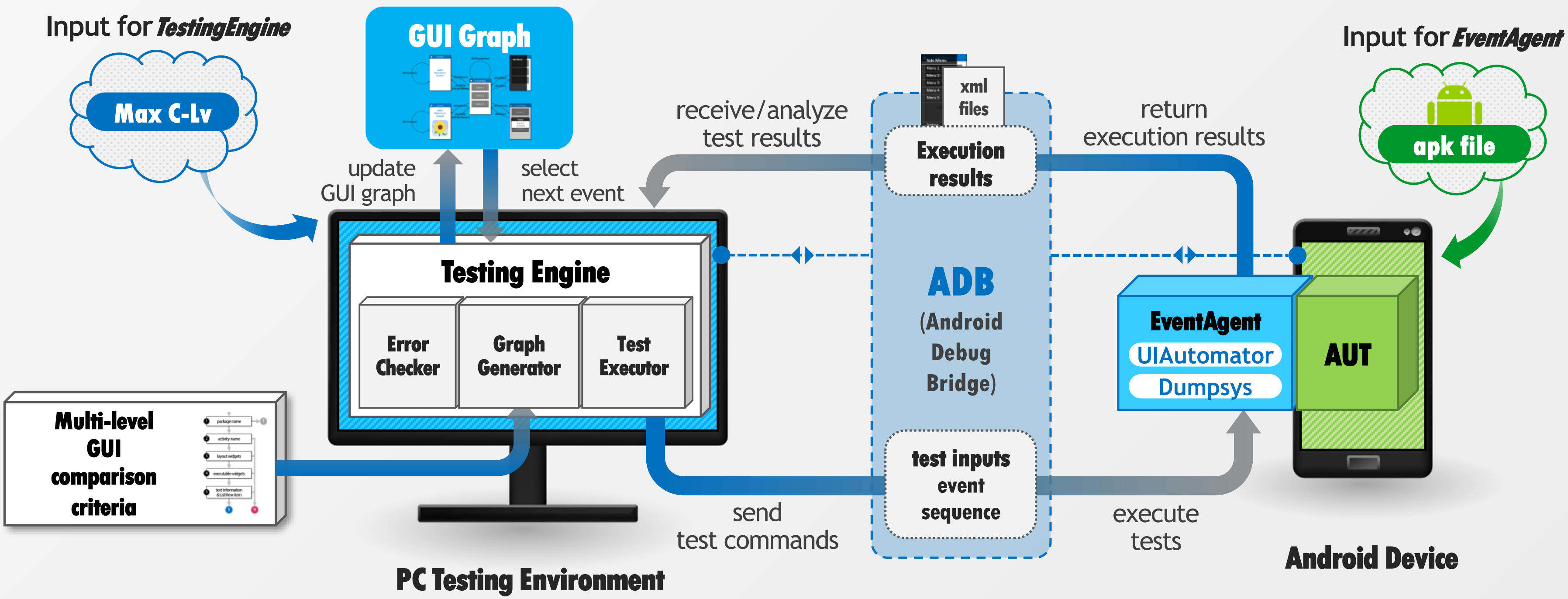


A GUI comparison model using multi-level GUICC for Android apps

Testing Framework with Multi-level GUICC

Automated Model Learner for Android

- Traverse AUT's behavior space and build a GUI graph based on the execution traces
- Generate test inputs based on the graph generated so far



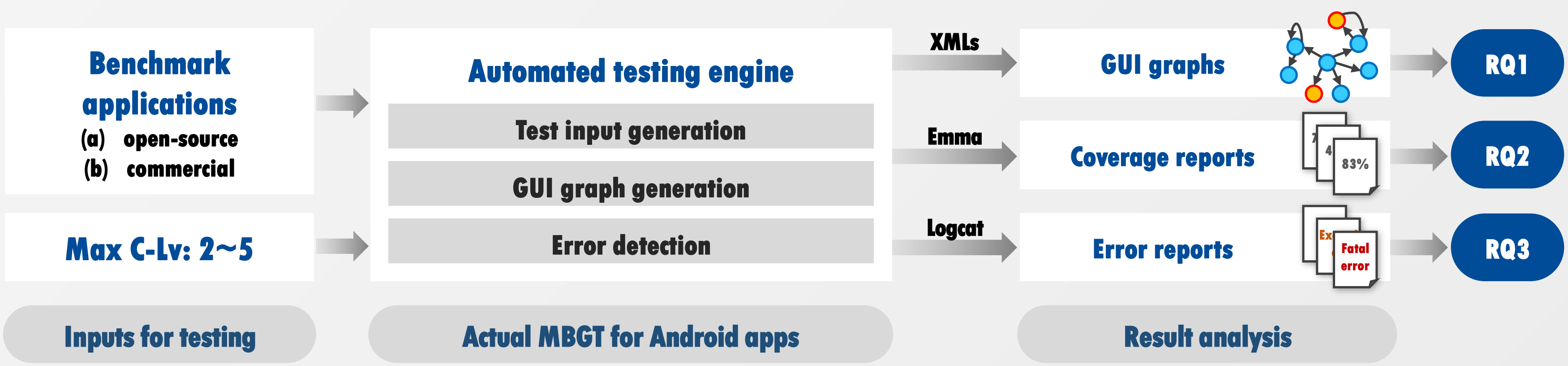
SELAB KAIST

Automated Model-based Android GUI Testing using Multi-level GUI Comparison Criteria

Empirical Study

Research Questions

- **Evaluating the influence of GUICC on the effectiveness of automated model-based GUI testing for Android**
  - **RQ1:** How does the GUICC affect the **behavior modeling**?
    - (a) GUI graph generation of open-source apps
    - (b) GUI graph generation of commercial apps
  - **RQ2:** Does the GUICC affect the **code coverage**?
  - **RQ3:** Does the GUICC affect the **error detection ability**?



SELAB KAIST

Experimental Setup

Benchmark Android apps: open-source & commercial apps

- Commercial Android apps were used to assess the feasibility of our testing framework for real-world apps

Open-source benchmark apps*		
No	**Application package**	**LOC**
1	*org.jtb.alogcat*	1.5K
2	*com.example.anycut*	1.1K
3	*com.evancharlton.mileage*	4.6K
4	*cri.sanity*	8.1K
5	*ori.jessies.dalvikexplorer*	2.2K
6	*i4nc4mp.myLock*	1.4K
7	*com.bwx.bequick*	6.3K
8	*com.nloko.android.syncmypix*	7.2K
9	*net.mandaria.tippytipper*	1.9K
10	*de.freewarepoint.whohasmystuff*	1.1K

Commercial benchmark apps		
No	**Application name**	**Download**
1	*Google Translate*	300,000K
2	*Advanced Task Killer*	70,000K
3	*Alarm Clock Xtreme Free*	30,000K
4	*GPS Status & Toolbox*	30,000K
5	*Music Folder Player Free*	3,000K
6	*Wifi Matic*	3,000K
7	*VNC Viewer*	3,000K
8	*Unified Remote*	3,000K
9	*Clipper*	750K
10	*Life Time Alarm Clock*	300K

* The open-source Android apps were collected from *F-Droid* (https://f-droid.org)

Experimental Configuration

- **Test input generation algorithm: fixed**
  - For only assessing the influence of GUICC, test input generation was performed with the same algorithm.
- **Exploration strategy: BFS (Breadth-first-search)**
  - In order to exercise much behavior during the same amount of time, our framework implements BFS strategy as a default.
- **Knowledge of the source code of apps: Black-box**
  - Our framework do not require the detailed knowledge of the underlying source code of an AUT.
  - Our framework only needs (1) an APK file and (2) a specific C-Lv.

Automated Model-based Android GUI Testing
using Multi-level GUI Comparison Criteria

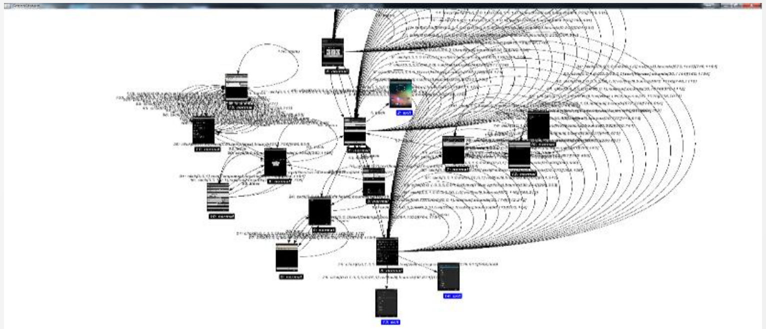
Results

GUI Graph Generation with Our Framework

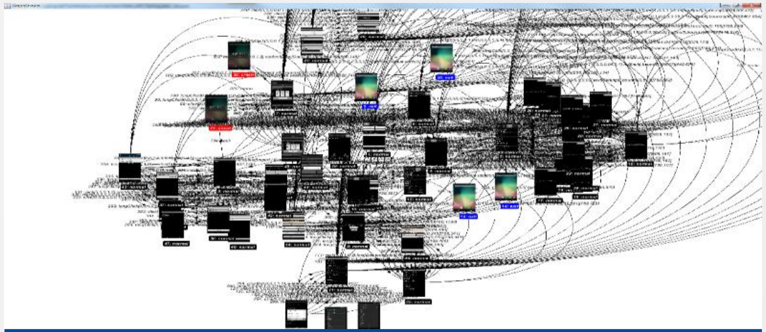
- **Automated GUI crawling and model-based test input generation using multi-level GUICC**

GUI Graph Generation with Our Framework

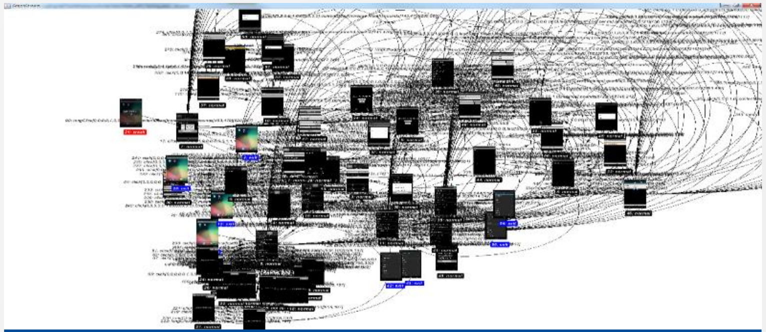
- **Multi-level GUI graph generation by manipulating C-Lvs***



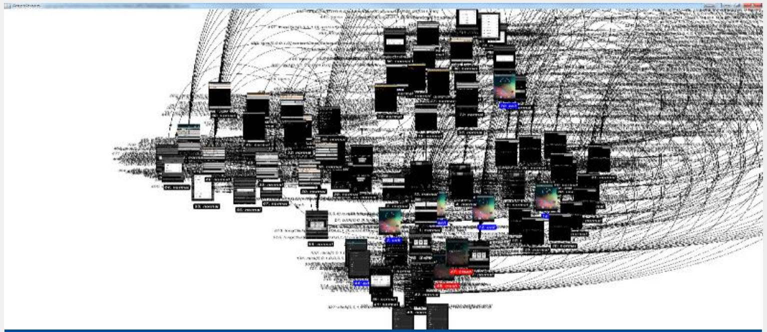
[C-Lv 2] 16 ScreenNodes, 117 EventEdges



[C-Lv 3] 48 ScreenNodes, 385 EventEdges



[C-Lv 4] 57 ScreenNodes, 401 EventEdges



[C-Lv 5] 77 ScreenNodes, 563 EventEdges

* Example GUI graphs are the modeling results of *Mileage app (com.evancharlton.mileage)*

RQ1: Evaluation on GUI Modeling by GUICC

Generated GUI graphs by GUICC (Max C-Lv)

- A. Open-source benchmark Android apps
  - Number of EventEdges (#EE) indicates the number of exercised test inputs

No	Package name	Activity-based		Proposed comparison steps					
		C-Lv2		C-Lv3		C-Lv4		C-Lv5	
		#SN	#EE	#SN	#EE	#SN	#EE	#SN	#EE
1	*org.jtb.alogcat*	5	45	8	66	15	247	76	269
2	*com.example.anycut*	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	*com.evancharlton.mileage*	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	*cri.sanity*	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
5	*ori.jessies.dalvikexplorer*	16	178	29	285	30	301	S/E	S/E
6	*i4nc4mp.myLock*	2	24	5	51	5	51	10	101
7	*com.bwx.bequick*	2	7	36	200	60	250	71	351
8	*com.nloko.android.syncmypix*	4	11	17	81	20	96	20	115
9	*net.mandaria.tippytipper*	4	29	11	65	13	102	19	175
10	*de.freewarepoint.whohasmystuff*	7	37	15	106	24	143	26	180

1. More GUI states were modeled.
2. More test inputs were inferred and exercised.

**C-Lv: level of comparison, #SN: number of ScreenNodes, #EE: number of EventEdges*

RQ1: Evaluation on GUI Modeling by GUICC

Generated GUI graphs by GUICC (Max C-Lv)

- A. Open-source benchmark Android apps
  - Number of EventEdges (#EE) indicates the number of exercised test inputs

No	Package name	Activity-based		Proposed comparison steps					
		C-Lv2		C-Lv3		C-Lv4		C-Lv5	
		#SN	#EE	#SN	#EE	#SN	#EE	#SN	#EE
1	*org.jtb.alogcat*	5	45	8	66	15	247	76	269
2	*com.example.anycut*	8	33	8	33	8	33	9	42
3	*com.evancharlton.mileage*	16	117	48	385	69	532	81	618
4	*cri.sanity*	1	4	1	4	2	7	145	922
5	*ori.jessies.dalvikexplorer*	16	178	29	285	30	301	S/E	S/E
6	*i4nc4mp.myLock*	2	24	5	51	5	51	10	101
7	*com.bwx.bequick*	2							
8	*com.nloko.android.syncmypix*	4							
9	*net.mandaria.tippytipper*	4							
10	*de.freewarepoint.whohasmystuff*	7							

State Explosion (S/E)
DalvikExplorer has continuously changing TextView
for the real-time monitoring of Dalvik VM

**For DalvikExplorer, C-Lv4 could be
the best GUICC for behavior modeling.**

RQ2: Evaluation on Code Coverage by GUICC

Achieved code coverage by GUICC (Max C-Lv)

- C-Lv shows the minimum comparison level to achieve the maximum coverage **M**

No	Package name	Class coverage			Method coverage			Block coverage			Statement coverage		
		A	C-Lv	M	A	C-Lv	M	A	C-Lv	M	A	C-Lv	M
1	*org.jtb.alogcat*	51%	4	69%	46%	4	65%	42%	5	60%	39%	5	56%
2	*com.example.anycut*	27%	4	86%	23%	4	69%	18%	5	56%	19%	4	55%
3	*com.evancharlton.mileage*	28%	5	59%	22%	5	43%	19%	5	36%	18%	5	33%
4	*cri.sanity*		n/a			n/a			n/a			n/a	
5	*ori.jessies.dalvikexplorer*	71%	4	73%	65%	4	70%	60%	4	67%	57%	4	64%
6	*i4nc4mp.myLock*	16%	3	16%	11%	4	12%	11%	4	12%	10%	4	11%
7	*com.bwx.bequick*	43%	4	51%	24%	5	39%	22%	5	38%	21%	5	39%
8	*com.nloko.android.syncmypix*	22%	4	50%	10%	4	24%	5%	4	15%	6%	4	17%
9	*net.mandaria.tippytipper*	70%	5	93%	42%	5	65%	37%	5	64%	36%	5	61%
10	*de.freewarepoint.whohasmystuff*	74%	5	89%	39%	5	62%	35%	5	52%	35%	4	51%
	Average	45%		65%	31%		50%	28%		44%	27%		43%

***C-Lv**: *minimum comparison level that achieves the maximum coverage,* **A**: *Activity-based,* **M**: *maximum coverage (C-Lv3~5)*

RQ2: Evaluation on Code Coverage by GUICC

Achieved code coverage by GUICC (Max C-Lv)

- C-Lv shows the minimum comparison level to achieve the maximum coverage M

Activity-based testing achieved lower code coverage than testing with other higher levels of GUICC

No	Package name	Class coverage			Method coverage			Block coverage			Statement coverage		
		A	C-Lv	M	A	C-Lv	M	A	C-Lv	M	A	C-Lv	M
1	*org.jtb.alogcat*	51%	18%	69%	46%	19%	65%	42%	18%	60%	39%	17%	56%
2	*com.example.anycut*	27%	59%	86%	23%	36%	69%	18%	38%	56%	19%	36%	55%
3	*com.evancharlton.mileage*	28%	31%	59%	22%	21%	43%	19%	17%	36%	18%	15%	33%
4	*cri.sanity*		n/a			n/a			n/a			n/a	
5	*ori.jessies.dalvikexplorer*	71%	2%	73%	65%	5%	70%	60%	7%	67%	57%	7%	64%
6	*i4nc4mp.myLock*	16%	0%	16%	11%	1%	12%	11%	1%	12%	10%	1%	11%
7	*com.bwx.bequick*	43%	8%	51%	24%	15%	39%	22%	16%	38%	21%	18%	39%
8	*com.nloko.android.syncmypix*	22%	28%	50%	10%	15%	24%	5%	10%	15%	6%	11%	17%
9	*net.mandaria.tippytipper*	70%	23%	93%	42%	13%	65%	37%	27%	64%	36%	25%	61%
10	*de.freewarepoint.whohasmystuff*	74%	15%	89%	39%	23%	62%	35%	17%	52%	35%	16%	51%
	Average	45%	20%	65%	31%	19%	50%	28%	16%	44%	27%	16%	43%

**C-Lv: minimum comparison level that achieves the maximum coverage, A: Activity-based, M: maximum coverage (C-Lv3~5)*

RQ3: Evaluation on Error Detection Ability by GUICC

- **Detected runtime errors by GUICC (Max C-Lv)**
  - Our testing framework had detected four reproducible runtime errors in open-source benchmark apps.

No	C-Lv	Application package	Error type	Detected error log
1	*C-Lv5*	com.evancharlton.mileage	Fatal signal	**F/libc(23414):** ***Fatal signal 11 (SIGSEGV)*** at 0x9722effc (code=2), thread 23414 (harlton.mileage)
2	*C-Lv5*	cri.sanity	Fatal exception	**E/AndroidRuntime(9415):** FATAL EXCEPTION: main **E/AndroidRuntime(9415):** java.lang.RuntimeException: Unable to start activity ComponentInfo{cri.sanity/cri.sanity.screen.VibraActivity}: ***java.lang.NullPointerException***
3	*C-Lv5*	cri.sanity	Fatal exception	**E/AndroidRuntime(22158):** FATAL EXCEPTION: main **E/AndroidRuntime(22158):** java.lang.RuntimeException: Unable to start activity ComponentInfo {cri.sanity/cri.sanity.screen.VibraActivity}: **java.lang.NullPointerException**
4	*C-Lv4*	com.evancharlton.mileage	Fatal signal	**F/libc(20978):** ***Fatal signal 11 (SIGSEGV)*** at 0x971b4ffc (code=2), thread 20978 (harlton.mileage)

RQ3: Evaluation on Error Detection Ability by GUICC

- **Detected runtime errors by GUICC (Max C-Lv)**
  - Our testing framework had detected four reproducible runtime errors in open-source benchmark apps.

No	C-Lv	Application package	Error type	Detected error log
1	*C-Lv5*	com.	ignal	**F/libc(23414): *Fatal signal 11 (SIGSEGV)*** at 0x9722effc (code=2), thread 23414 (harlton.mileage)
2	*C-Lv5*	cri.s	xception	**E/AndroidRuntime(9415):** FATAL EXCEPTION: main **E/AndroidRuntime(9415):** java.lang.RuntimeException: Unable to start activity ComponentInfo{cri.sanity/cri.sanity.screen.VibraActivity}: ***java.lang.NullPointerException***
3	*C-Lv5*	cri.sanity	Fatal exception	**E/AndroidRuntime(22158):** FATAL EXCEPTION: main **E/AndroidRuntime(22158):** java.lang.RuntimeException: Unable to start activity ComponentInfo {cri.sanity/cri.sanity.screen.VibraActivity}: **java.lang.NullPointerException**
4	*C-Lv4*	com.evancharlton.mileage	Fatal signal	**F/libc(20978): *Fatal signal 11 (SIGSEGV)*** at 0x971b4ffc (code=2), thread 20978 (harlton.mileage)

From C-Lv2 to C-Lv3,
these runtime errors could not be detected
by automated testing

Summary of Experimental Results

- "Activity" is still frequently used as a simple GUICC by many tools, but **Activity-based models have to be refined**.
- Clearly **defining an appropriate GUICC** can be an easier way to improve overall testing effectiveness.
- Higher levels of GUICC are **not always optimal solutions**.

Conclusion

Conclusion

Current model-based GUI testing techniques for Android

Focus on improving test generation algorithms, exploration strategies

Use arbitrary GUI comparison criteria

This study

Multi-level GUICC
- Investigation of behaviors of Android GUIs
- Design GUI comparison model

+

Automated model-based testing framework
- GUI graph generation
- Model-based test input generation
- Error detection

→

Empirical study of the influence of GUICC
- Behavior modeling by GUICC (a) open-source apps, (b) commercial apps
- Code coverage by GUICC
- Error detection by GUICC

Future direction for Android model-based GUI testing

Automated model-based testing should **carefully/clearly define the GUICC** according to AUTs' behavior styles, prior to improvement of other algorithms

Threats to Validity (1/3)

- **Automated selection of an adequate GUICC**
  - **Current related work**
    - [Gap 2 in Pekka Aho et al., 2015*] **To refine a generated GUI model by providing inputs for multiple states of the GUI after model generation**
    - [State abstraction by Pekka Aho et al., 2014**] **To abstract away the data values by parameterizing screenshots of the GUI**
  - **Future work**
    - **Feature-based GUI model generation**
      - ✓ **Automatic feature extraction from APK file and GUI information of minimal number of main screens (uploaded on the market)**
      - ✓ **Generation of specific abstraction levels of a GUI model based on the extracted GUI features**

Threats to Validity (2/3)

Performance problems in model-based GUI testing

- Current performance
  - Modeling a GUI graph with about 25 nodes and 150 edges takes about a half an hour. → Expensive for industrial application
- Future work
  - Incremental model refinement
    - Build the most abstract GUI model first, and then incrementally refine some parts of the model into more concrete models.
  - GUI model clustering
    - Build multi-level GUI models parallel, and then cluster them into a single model of an AUT
    - Generate more sophisticated test inputs using the clustered model

Threats to Validity (3/3)

Other threats to validity

- Time and memory consumption problems
  - Online model-learning (GUI ripping) requires non-trivial testing time and memory.
- Reliable on test assistant tools
  - Our testing framework uses UIAutomator, Dumpsys to analyze GUI states and test input generation automatically.
- Quality of F-Droid apps
  - Many open-source Android apps in F-Droid are not following the latest design trends,

Summary

- **GUI testing is necessary.**
  - Graphical User Interface (GUI) decides that a user is going to use the app further or not.
  - GUI testing is a process that detects if an application is functionally correct by using its GUIs.
- **We should understand the challenges of automated model-based GUI testing for mobile apps.**
  - Due to various characteristics of recent mobile apps, GUI testing becomes more difficult.
  - In particular, model-based techniques should understand these challenges and they must be carefully addressed for the practical application.
- **Our empirical study (ASE'16) shows the importance of GUI model generation and provides future research directions.**

한국정보과학회 프로그래밍언어연구회 여름학교 프로그램 (SIGPL)

Challenges of Automated Model-based GUI Testing for Android Apps

Thank You.

Young-Min Baek, Doo-Hwan Bae

{ymbaek, bae}@se.kaist.ac.kr
http://se.kaist.ac.kr

Publication

- **Final publication copy of this paper**
  - Young-Min Baek, Doo-Hwan Bae, "**Automated Model-Based Android GUI Testing using Multi-level GUI Comparison Criteria,**" In Automated Software Engineering (ASE), 2016 31th IEEE/ACM International Conference on,
  - URL: http://dl.acm.org/citation.cfm?doid=2970276.2970313
  - DOI: 10.1145/2970276.2970313

SE LAB KAIST

Appendix

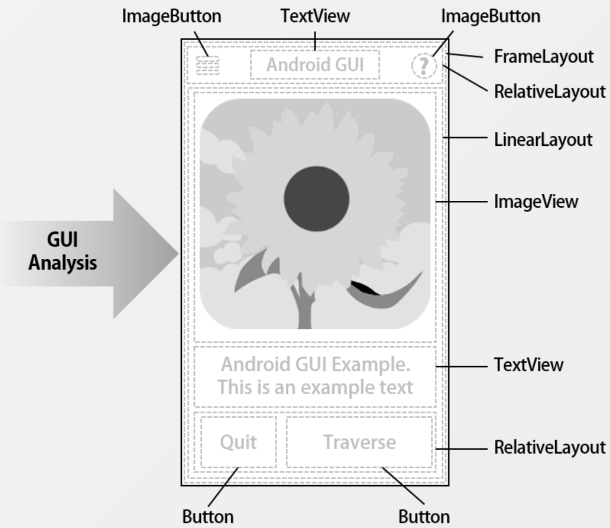
A. Android GUI
B. Example of Dynamic/Volatile Android GUIs
C. Investigated Android Apps
D. Comparison of Widget Compositions using UIAutomator
E. Comparison Examples
F. Exploration Strategies of Our Framework
G. Exploration Strategy: Breadth-first-search (BFS)
H. Exploration Strategy: Depth-first-search (DFS)

Android GUI

Definition

- An **Android GUI** is a hierarchical, graphical front-end to an Android application displayed on the foreground of the Android device screen that accepts input events from a finite set of events and produces graphical output according to the inputs.
- An Android GUI hierarchically consists of specific types of graphical objects called **widgets**; each widget has a fixed set of **properties**; each property has discrete **values** at any time during the execution of the GUI.

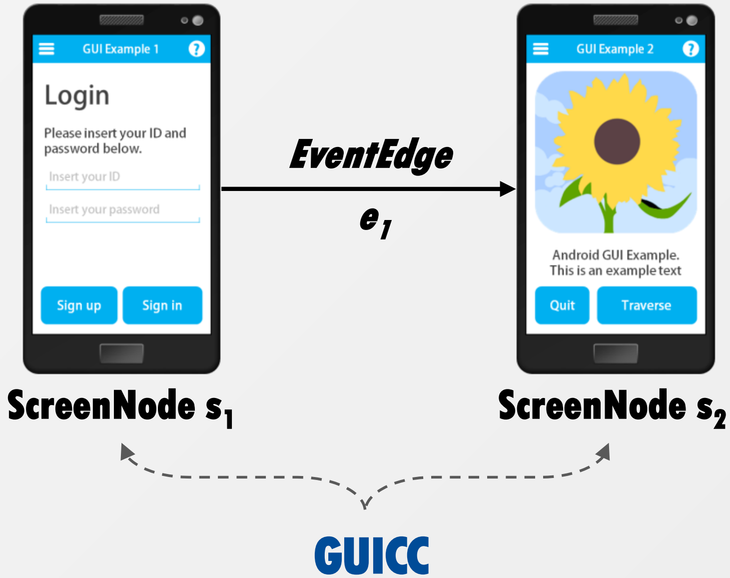
A GUI on Android screen



Hierarchical widget structure

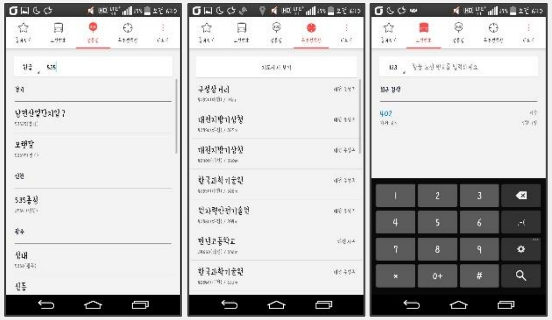
Android GUI – Formal definition of GUI graph

- **A GUI graph G is defined as $G = (S, E)$**
  - S is a set of **ScreenNodes** ($S = \{s_1, s_2, \ldots, s_n\}$), n = # of nodes.
  - E is a set of **EventEdges** ($E = \{e_1, e_2, \ldots, e_m\}$), m = # of edges.
  - A **GUI Comparison Criterion (GUICC)** represents a specific type of GUI information to distinguish GUI states.

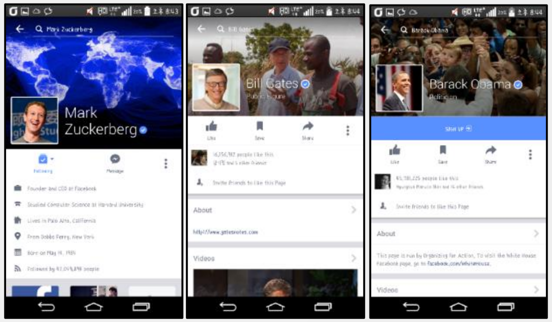


Examples of Dynamic/Volatile Android GUIs

- **Multiple dynamic pages of a single screen (Seoulbus view pages)**
  - ViewPager widget is used to provide multiple view pages in a single activity.



- **Non-deterministic (context-sensitive) GUIs (Facebook personal pages)**
  - Personal pages of SNSs are dependent on users' preference or edited/configured profile.



- **Endless GUIs (Facebook timeline views)**
  - Newsfeed of SNSs provides an endless scroll view to provide friends' or linked people's news.

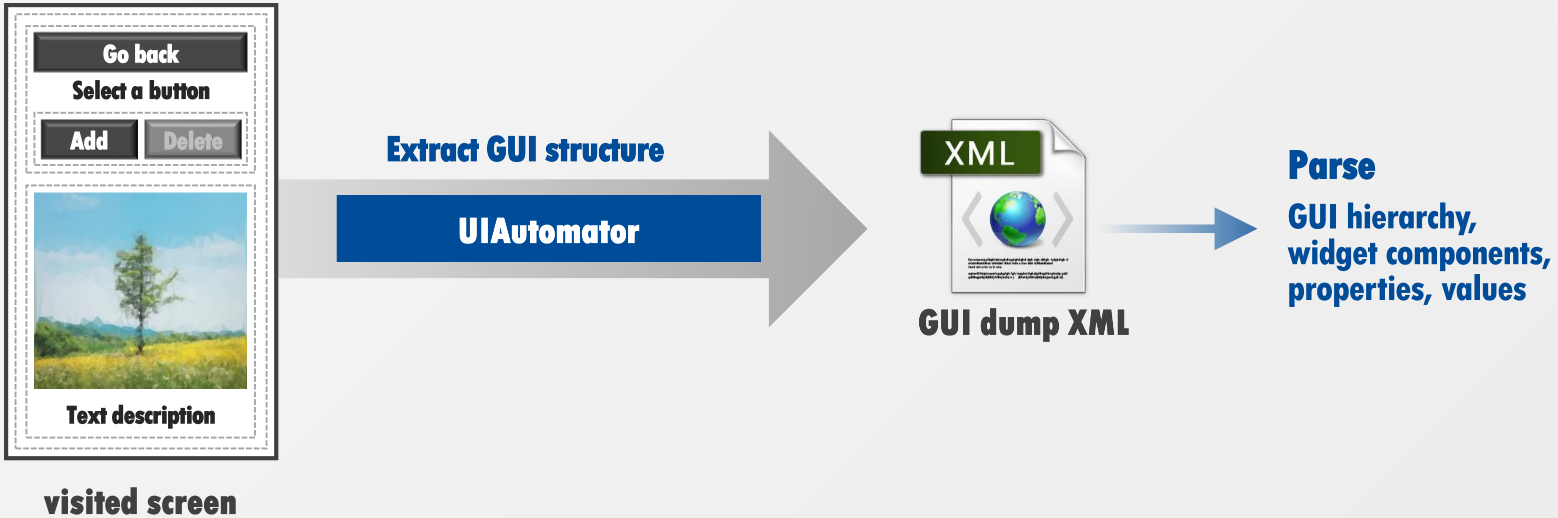
Android Apps Investigated for Building GUICC Model

- 93 real-world commercial Android apps registered in Google Playstore

No	Android application (package name)	Category
1	com.scribd.app.reader0	Book
2	com.spreadsong.freebooks	Book
3	com.tecarta.kjv2	Book
4	com.google.android.apps.books	Book
5	com.merriamwebster	Book
6	com.taptapstudio.dailyprayerlite	Book
7	com.audible.application	Book
8	wp.wattpad	Book
9	com.dictionary	Book
10	com.amazon.kindle	Book
11	org.wikipedia	Book
12	com.ebooks.ebookreader	Book
13	an.SpanishTranslate	Book
14	com.autodesk.autocadws	Business
15	com.futuresimple.base	Business
16	mm.android	Business
17	com.yammer.v1	Business
18	com.invoice2go.invoice2goplus	Business
19	com.docusign.ink	Business
20	com.alarex.gred	Business
21	com.fedex.ida.android	Business
22	com.google.android.calendar	Business
23	com.metago.astro	Business
24	com.squareup	Business
25	com.dynamixsoftware.printershare	Business
26	kik.android	Communication
27	com.tumblr	Communication
28	com.twitter.android	Communication
29	com.oovoo	Communication
30	com.facebook.orca	Communication
31	com.yahoo.mobile.client.android.mail	Communication
32	com.skout.android	Communication
33	com.mrnumber.blocker	Communication
34	com.taggedapp	Communication
35	com.timehop	Communication
36	com.carezone.caredroid.careapp.medications	Medical
37	com.hp.pregnancy.lite	Medical
38	com.medscape.android	Medical
39	com.szyk.myheart	Medical
40	com.smsrobot.period	Medical
41	com.hssn.anatomyfree	Medical
42	au.com.penguinapps.android.babyfeeding.client.android	Medical
43	com.cube.arc.blood	Medical
44	com.doctorondemand.android.patient	Medical
45	com.bandsintown	Music
46	com.djit.equalizerplusforandroidfree	Music
47	com.madebyappolis.spinrilla	Music
48	com.magix.android.mmjam	Music
49	com.shazam.android	Music
50	com.songkick	Music
51	com.famousbluemedia.yokee	Music
52	com.musixmatch.android.lyrify	Music
53	tunein.player	Music
54	com.google.android.music	Music
55	com.ebay.mobile	Shopping
56	com.grandst	Shopping
57	com.biggu.shopsavvy	Shopping
58	com.ebay.redlaser	Shopping
59	com.alibaba.aliexpresshd	Shopping
60	com.newegg.app	Shopping
61	com.islickapp.pro	Shopping
62	com.ubermind.rei	Shopping
63	com.inditex.zara	Shopping
64	com.linkedin.android	Social
65	com.foursquare.robin	Social
66	com.match.android.matchmobile	Social
67	com.whatsapp	Social
68	flipboard.app	Social
69	com.facebook.katana	Social
70	com.instagram.android	Social
71	net.mypapit.mobile.speedmeter	Transportation
72	com.lelic.speedcam	Transportation
73	com.sygic.speedcamapp	Transportation
74	com.funforfones.android.dcmetro	Transportation
75	br.com.easytaxi	Transportation
76	com.nyctrans.it	Transportation
77	com.ninetyeightideas.nycapp	Transportation
78	com.nomadrobot.mycarlocatorfree	Transportation
79	com.ubercab	Transportation
80	org.mrchops.android.digihud	Transportation
81	com.drivemode.android	Transportation
82	com.greyhound.mobile.consumer	Transportation
83	com.citymapper.app.release	Transportation
84	com.alokmandavgane.sunrisesunset	Weather
85	com.pelmorex.WeatherEyeAndroid	Weather
86	com.cube.arc.hfa	Weather
87	com.cube.arc.tfa	Weather
88	com.handmark.expressweather	Weather
89	com.accuweather.android	Weather
90	mobi.infolife.ezweather	Weather
91	com.levelup.brightweather	Weather
92	com.weather.Weather	Weather
93	com.yahoo.mobile.client.android.weather	Weather

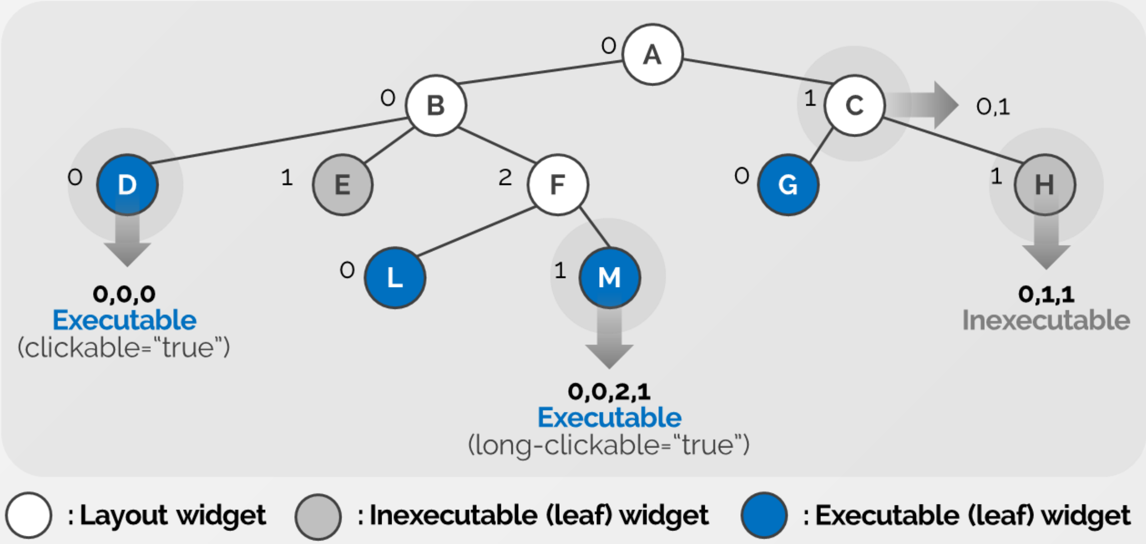
Comparison of Widget Compositions using UIAutomator (1/3)

- The Activity-based comparison could not distinguish detailed GUI states in real-world apps.
  - Many commercial apps utilize the `ViewPager` widget, which contains multiple pages to show.
  - However, Activity-based GUI models cannot distinguish multiple different views.
- A widget hierarchy, which is extracted by UIAutomator, is used to compare two GUIs based on the composition of widgets.



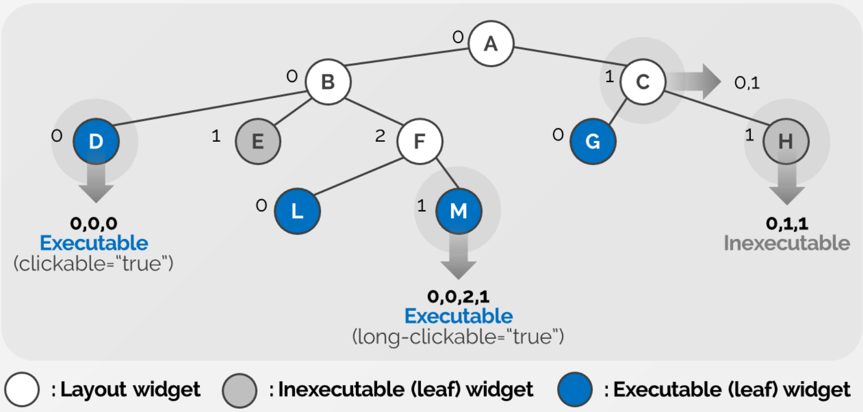
Comparison of Widget Compositions using UIAutomator (2/3)

- Every widget node has parents-children or sibling relationships.
- The relationships are encoded in an <index> property, which represents the order of child widget nodes.
  - If the value of an index of a certain widget w_i is 0, w_i is the first child of its parent widget node.
- By using index values, each widget (node X) can be specified as an index sequence that accumulates the indices from the root node to the target node.



Comparison of Widget Compositions using UIAutomator (3/3)

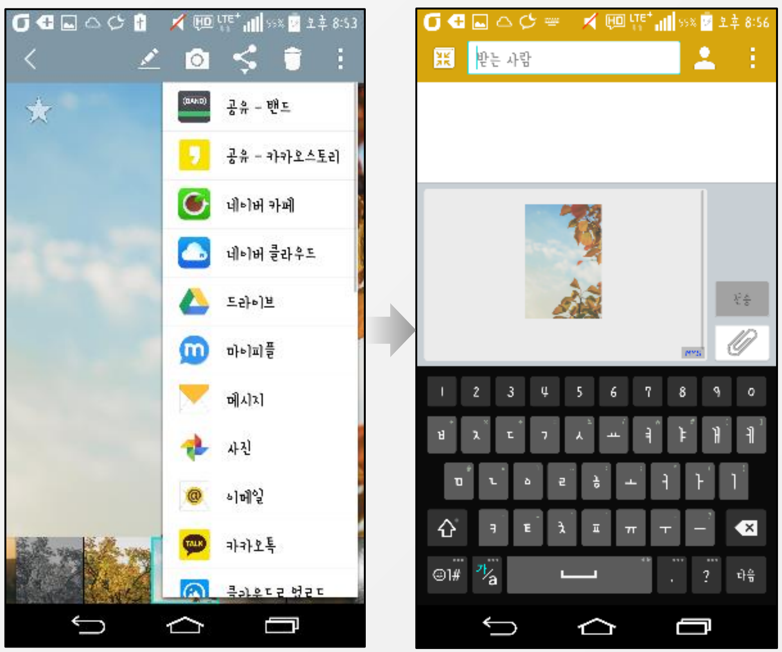
- **Our comparison model obtains the composition of specific types of widgets using these index sequences.**
  - Refer to non-leaf widgets as **layout widgets**
  - Refer to leaf widget nodes, whose event properties (e.g., clickable) have at least one "true" value as **executable widgets.**
    - If a non-leaf widget node has an executable property, its child leaf nodes are considered as executable widgets.
- **In order to utilize the extracted event sequences as the widget information, we store cumulative index sequences (CIS).**



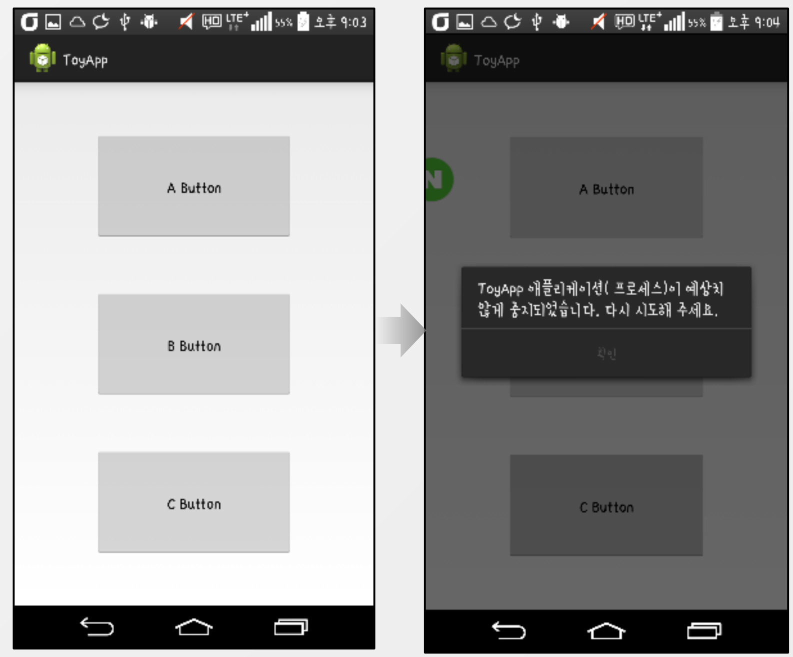
Type	Nodes	CIS
Layout	A, B, C, F	[0]-[0,0]-[0,1]-[0,0,2]
Executable	D, G, L, M	[0,0,0]-[0,1,0]-[0,0,2,0]-[0,0,2,1]

Comparison Example: C-Lv1

- **Comparison level 1 (C-Lv1): Package name comparison**
  - **By comparing package names, the testing tool distinguishes the boundary of the behavior space of an AUT**



Out of AUT boundary
3rd party app

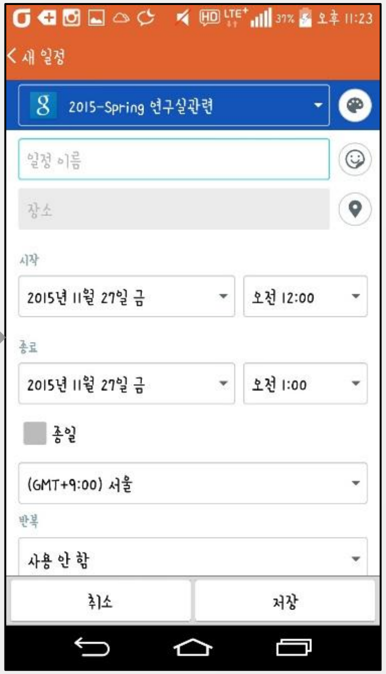


Out of AUT boundary
App termination by crash

Comparison Example: C-Lv2

- **Comparison level 2 (C-Lv2): Activity name comparison**
  - **By comparing activity names, the testing tool distinguishes the physically-independent GUIs (i.e., MainActivity and OtherActivity are implemented in different Java files).**

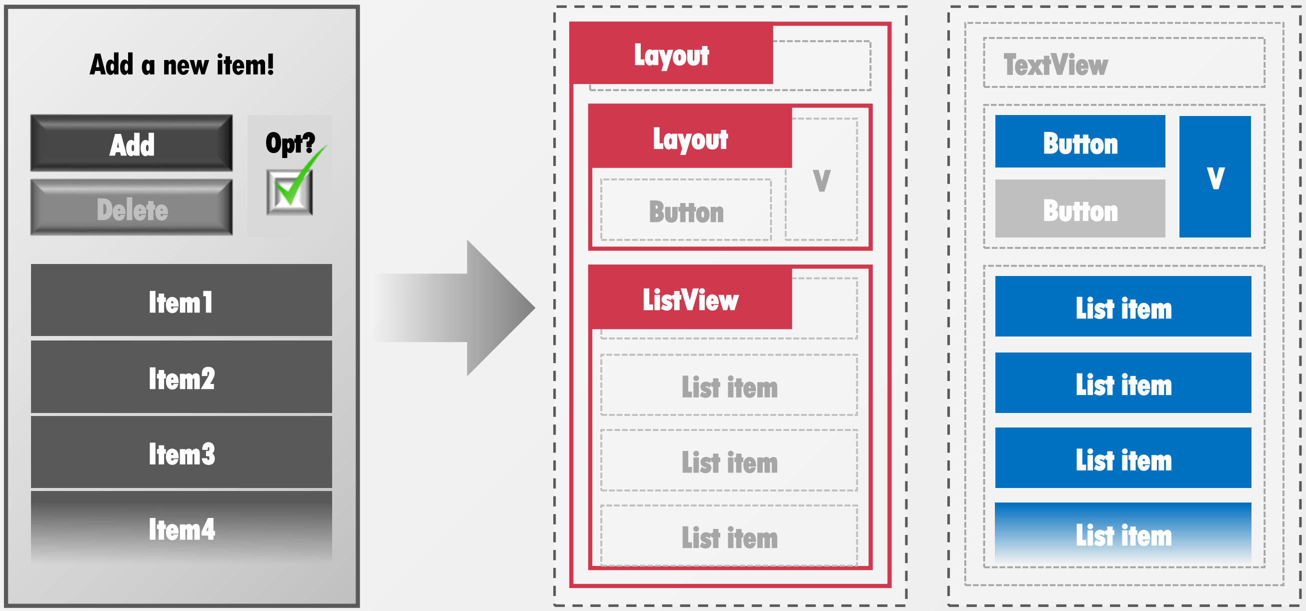
Activity 1
com.android.calendar.
AllInOneActivity



Activity2
com.android.calendar.
event.EditEventActivity

Comparison Example: C-Lv3/4

- **Comparison level 3, 4 (C-Lv4): Widget composition comparison**
  - Compare the widget composition of GUIs using UIAutomator hierarchy tree.
    - **Layout widgets**: composition of non-leaf widgets.
    - **Executable widgets**: composition of leaf widgets whose event properties have at least one "true" values



Widgets of an execution screen

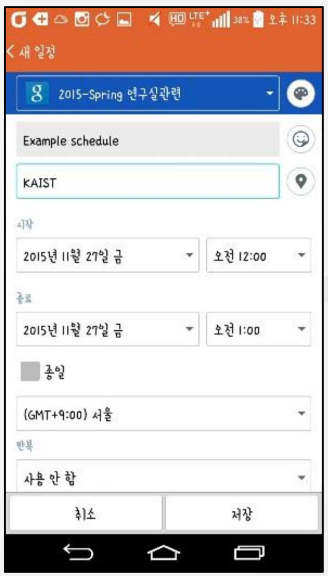
Layout widgets

Executable widgets

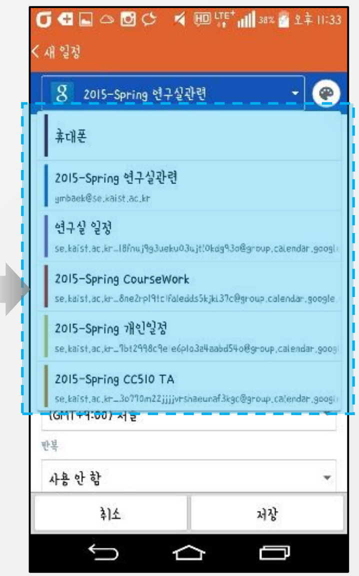
Comparison Example: C-Lv3/4

- **Comparison level 3, 4 (C-Lv4): Widget composition comparison**
  - Compare the widget composition of GUIs using UIAutomator hierarchy tree.
    - **Layout widgets**: composition of non-leaf widgets.
    - **Executable widgets**: composition of leaf widgets whose event properties have at least one “true” values

C_Lv3
Layout widget comparison



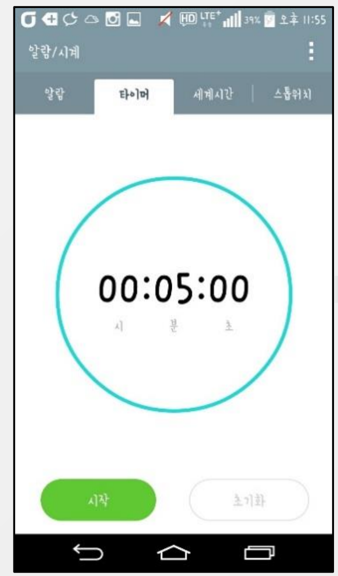
Layout widget composition 1



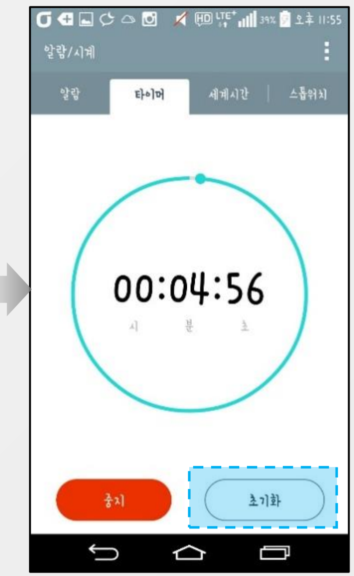
Layout widget composition 2

Context menu layout

C_Lv4
Executable widget composition



Executable widget composition 1



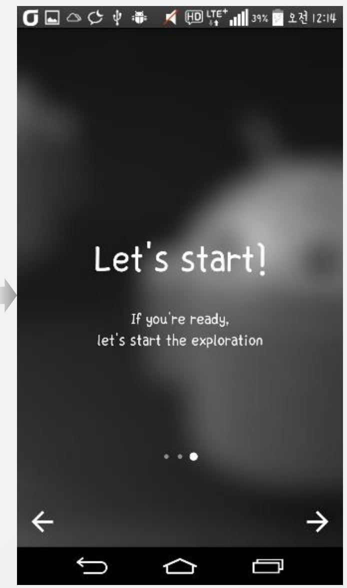
Executable widget composition 2

<Initialize> button

Comparison Example: C-Lv5

- **Comparison level 5 (C-Lv5): Text contents, item comparison**
  - Text information: compare the context of GUIs, which is represented as text
  - List item: distinguish the GUIs after scroll events

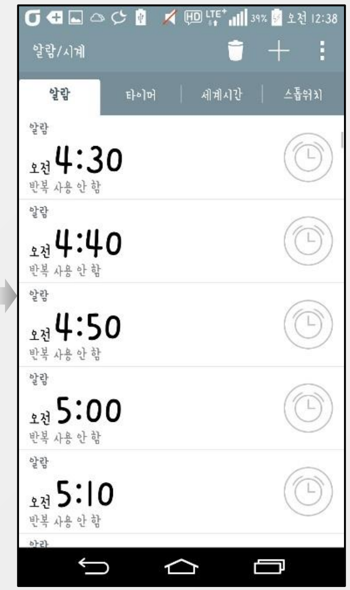
Text comparison



Text contents 1
GUI Testing?

Text contents 2
Let's start!

ListView item comparison



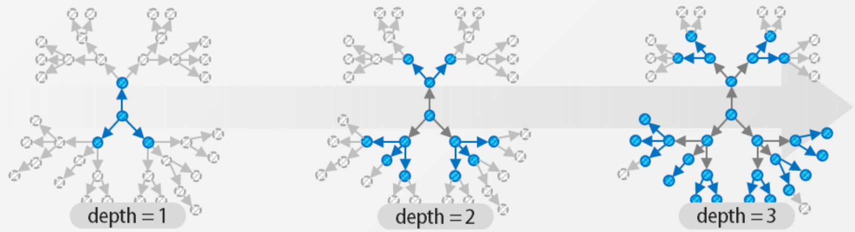
ListView before a scroll event
12:00 AM ~

ListView after scroll event execution
4:30 AM ~

Exploration Strategies in Our Testing Framework

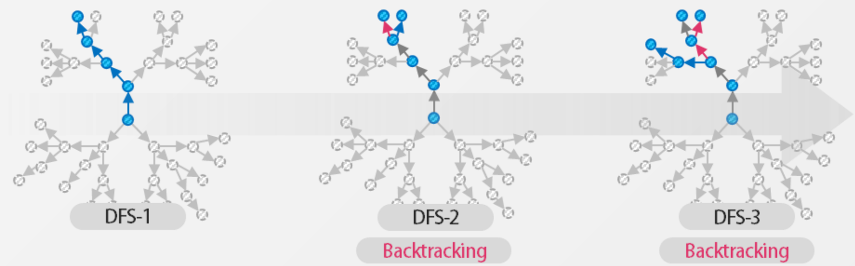
- **Breadth-First-Search (BFS): default strategy**

▶ Detailed algorithm

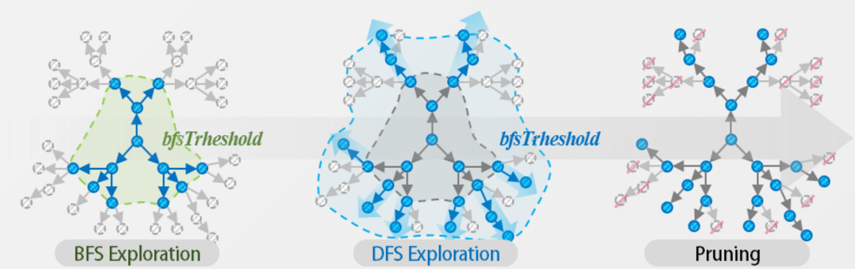


- **Depth-First-Search (DFS)**

▶ Detailed algorithm



- **Hybrid-Search (BFS + DFS)**



Exploration Strategy: Breadth-First-Search (BFS)

- BFS traverses the behavior space of an AUT in order of depth.
- BFS requires repetitive restart operation after test execution.
- BFS have a higher chance to reach more diverse range of states during the same amount of time.

Algorithm 1: BFS-CRAWLING-MODEL-GENERATION

```
Input: Target app A, BFS Threshold bThreshold, Comparison criteria CC
Output: GUI graph G = (S, E), S: set of ScreenNodes, E: set of EventEdges

Install an app using apk file of A
Run A on Android device (or emulator)
n ← 0
s_n ← ExtractGUI(deviceScreen)
s_n.E ← ExtractEvents(s_n)
Add s_n.E into EventQueue(EQ); EQ ← EQ ∪ s_n.E
Add s_n into G as a ScreenNode; S ← S ∪ {s_n}
s_c ← s_n
while EQ is not empty and bThreshold is not reached
    Restart the app A
    Poll the first event e from EQ
    Execute e
    e_c ← e
    s' ← ExtractGUI(deviceScreen)
    compareGUI(G, s', CC)
    if s' is an existing screen (i.e., s' = s_i, where s_i is one of ScreenNodes in S)
        then LinkWithEdge(s_c, s_i, e_c); E ← E ∪ {e_c}
        else n ← (n+1)
            s_n ← s'
            Add s_n into G as a ScreenNode; S ← S ∪ {s_n}
            LinkWithEdge(s_c, s_n, e_c); E ← E ∪ {e_c}
            s_n.E ← ExtractEvents(s_n)
            Add s_n.E into EventQueue(EQ); EQ ← EQ ∪ s_n.E
    end if
end while
return G
```

Exploration Strategy: Depth-First-Search (DFS)

- **Traverse as much behavior depth of an AUT along each branch before backtracking**
- **DFS have a higher chance to exercise behaviors caused by sequences of consecutive events.**

Algorithm 2: DFS-CRAWLING-MODEL-GENERATION

```
Input: Target app A, DFS Threshold dThreshold, Comparison criteria CC
Output: GUI graph G = (S, E), S: set of ScreenNodes, E: set of EventEdges
Initialized: <StateTrace> ← ∅, <EventTrace> ← ∅

Install an app using apk file of A
Run A on Android device (or emulator)
n ← 0
s_n ← ExtractGUI(deviceScreen)
s_n.E ← ExtractEvents(s_n)
Add s_n into G as a ScreenNode; S ← S ∪ {s_n}
s_c ← s_n
Add s_c into <StateTrace> in order
while dThreshold is not reached
    if s_c is an open-state (i.e., s_c.E is not empty)
        then Take out an event e from s_c.E; s_c.E ← s_c.E – {e}
            ExecuteEventsDFS(n, e, sc, EQ, G, <EventTrace>, <StateTrace>)
        else Find an open-state in <StateTrace>
            if there is a remaining open-state s_op in <StateTrace>
                then Restart the app A
                    Move to s_op using <EventTrace>
                else return G
            end if
    end if
end while
```

Algorithm 5: ExecuteEventDFS

```
Input: Node number n, Event e, Current screen sc, Event queue EQ, GUI graph G,
       Event trace <EventTrace>, State trace <StateTrace>
Output: Execution result R

ec ← e
Execute ec
if <EventTrace> is not null, then Add ec into <EventTrace> in order
s' ← ExtractGUI(deviceScreen)
compareGUI(G, s', CC)
if s' is an existing sreen (i.e., s' = si, where si is one of ScreenNodes in S)
    then LinkWithEdge(sc, si, ec)
        sc ← s'
        R ← EXISTING_SCREEN
    else n ← n + 1
        sn ← s'
        Add sn into G as a ScreenNode; S ← S ∪ {sn}
        LinkWithEdge(sc, sn, ec)
        sc ← sn
        sc.E ← ExtractEvents(sc)
        if <StateTrace> is not null, then Add sc into <StateTrace>
        R ← NEW_SCREEN
end if
return R
```

RQ2: Evaluation on Code Coverage by GUICC

- **Achieved code coverage by GUICC (Max C-Lv)**
  - C-Lv shows the minimum comparison level to achieve the maximum coverage M

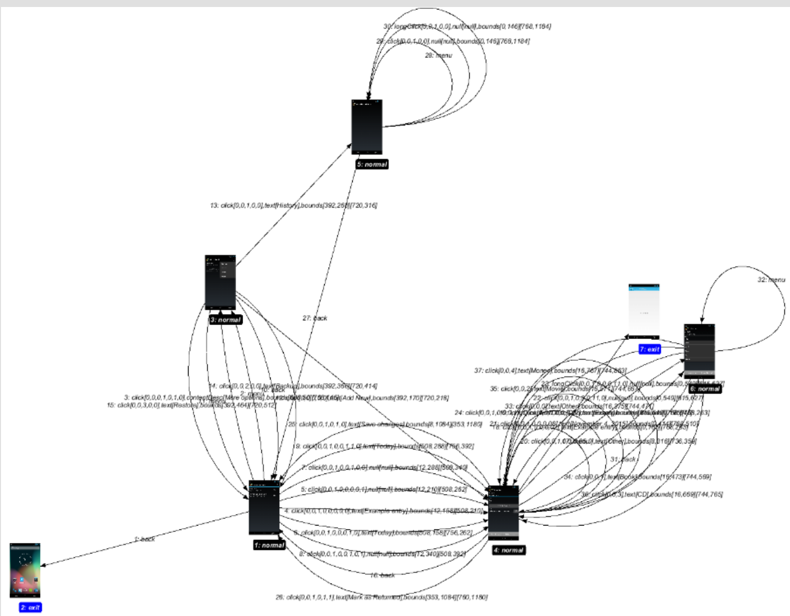
No	Package name	Class coverage			Method coverage			Block coverage			Statement coverage		
		A	C-Lv	M	A	C-Lv	M	A	C-Lv	M	A	C-Lv	M
1	*org.jtb.alogcat*	Several apps showed relatively low coverage					%	42%	5	60%	39%	5	56%
2	*com.example.anycut*						%	18%	5	56%	19%	4	55%
3	*com.evancharlton.mileage*	Sophisticated text inputs									18%	5	33%
4	*cri.sanity*		n/a			n/a			n/a			n/a	
5	*ori.jessies.dalvikexplorer*	71%	4	73%	65%	4	70%	60%	4	67%	57%	4	64%
6	*i4nc4mp.myLock*	Service-driven functionalities									10%	4	11%
7	*com.bwx.bequick*	43%	4	51%	24%	5	39%	22%	5	38%	21%	5	39%
8	*com.nloko.android.syncmypix*	Required external data (pictures)									6%	4	17%
9	*net.mandaria.tippytipper*	70%	5	93%	42%	5	65%	37%	5	64%	36%	5	61%
10	*de.freewarepoint.whohasmystuff*	74%	5	89%	39%	5	62%	35%	5	52%	35%	4	51%
	Average	45%		65%	31%		50%	28%		44%	27%		43%

**C-Lv: minimum comparison level that achieves the maximum coverage, A: Activity-based, M: maximum coverage (C-Lv3~5)*

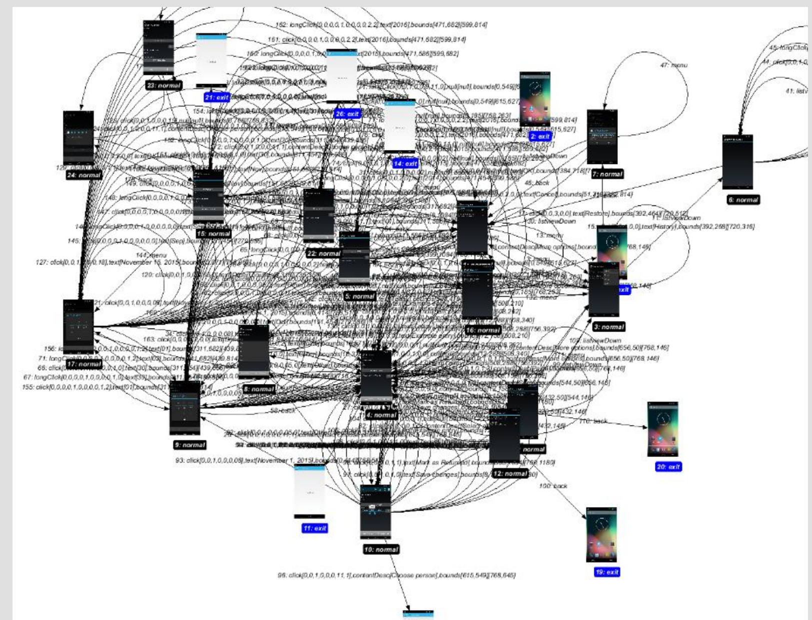
Example Modeling Results

- **Model generation of an app with two different GUICC***
  - <Who has my stuff?> App: de.freewarepoint.whohasmystuff

GUICC: Activity name
7 nodes, 37 edges



GUICC: widgets + text contents
26 nodes, 180 edges



* The generated GUI models are visualized by *GraphStream-Project*, http://graphstream-project.org/

At Which Level Conducting the GUI Testing?

Acceptance testing

- **Manual acceptance testing:** User (tester) exercises the system manually using the creativity, and evaluate the acceptance
- **Acceptance testing with GUI test drivers:** Tools help the developer do functional / acceptance testing
- **Table-based acceptance testing:** Starting from a user story (use case or textual requirement), the customer enters in a table the expectations of the program's behavior.

Regression testing

- Since GUIs are often realized by means of rapid prototyping or automatic framework, an efficient approach to generate and maintain GUI test suite is required.

This is the end of the file

Automated Software Engineering (ASE) 2016

Automated Model-Based Android GUI Testing using Multi-Level GUI Comparison Criteria

Young-Min Baek, Doo-Hwan Bae

{ymbaek, bae}@se.kaist.ac.kr
http://se.kaist.ac.kr