

# HANDS ON WITH CRASHSCOPE: AN AUTOMATED ANDROID TESTING TOOL

Kevin Moran,  
Assistant Professor  
George Mason University





**Kevin Moran**  
**Assistant Professor**  
**George Mason University**  
**Fairfax, VA, USA**  
**kpmoran@gmu.edu**  
**https://www.kpmoran.com**

---

# PART 0: BACKGROUND AND CORE CONCEPTS

---

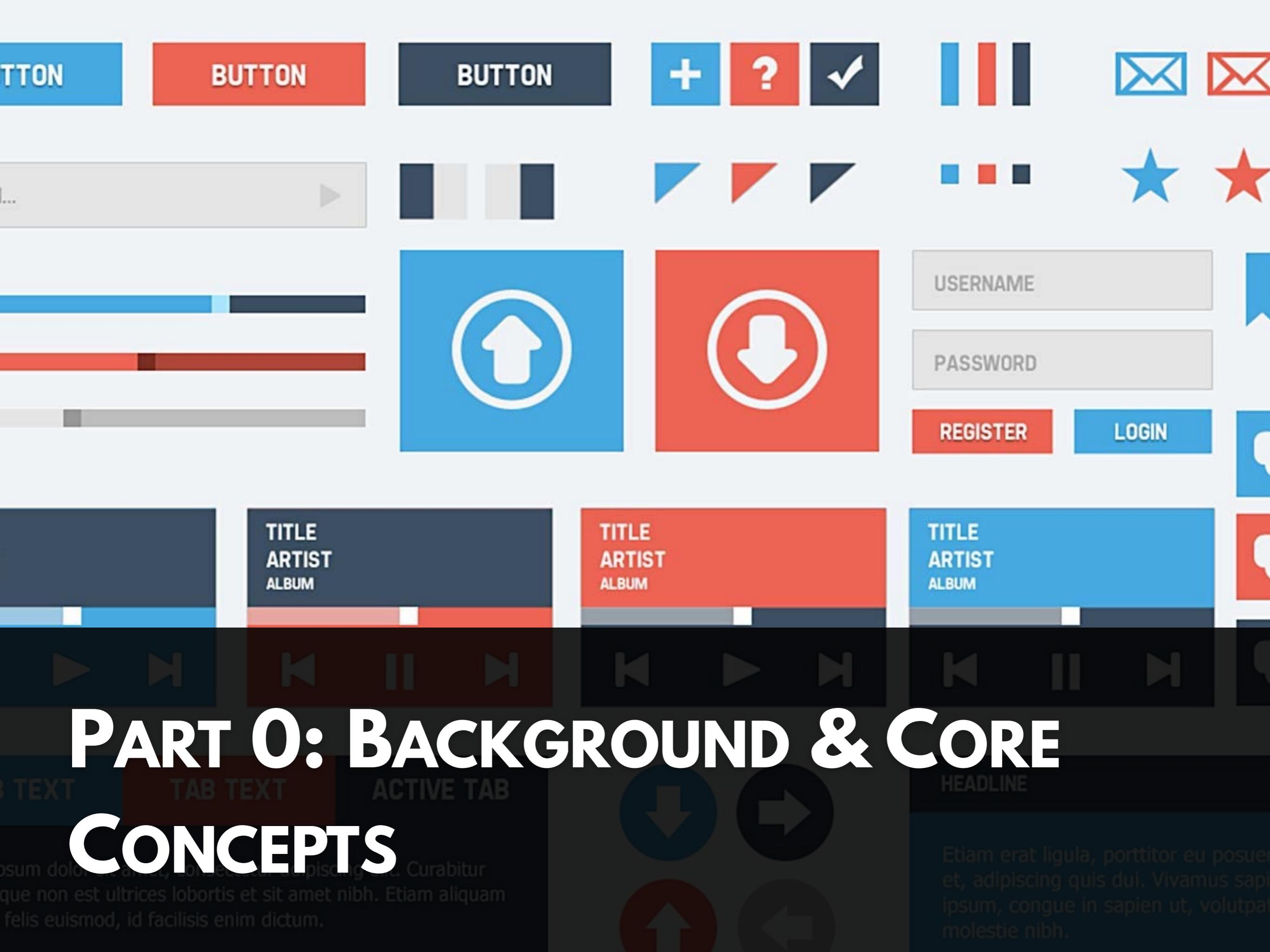
# PART 1: CURRENT RESEARCH & FUTURE WORK

---

# PART 2: AN OVERVIEW OF CRASHSCOPE

---

# PART 4: HANDS-ON SESSION WITH CRASHSCOPE



# PART 0: BACKGROUND & CORE CONCEPTS

Etiam erat ligula, porttitor eu posueret, adipiscing quis dui. Vivamus sapien ipsum, congue in sapien ut, volutpat molestie nibh. Curabitur que non est ultrices lobortis et sit amet nibh. Etiam aliquam felis euismod, id facilisis enim dictum.

Etiam erat ligula, porttitor eu posueret, adipiscing quis dui. Vivamus sapien ipsum, congue in sapien ut, volutpat molestie nibh.

# The Importance of GUI Testing

- Several different types of testing are important for ensuring software quality:

# The Importance of GUI Testing

- Several different types of testing are important for ensuring software quality:

Unit Testing

Performance Testing

Integration Testing

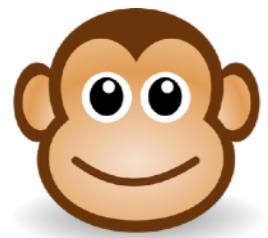
Compatibility Testing

Regression Testing

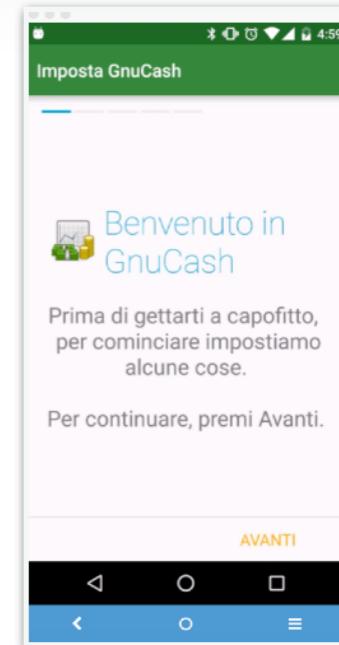
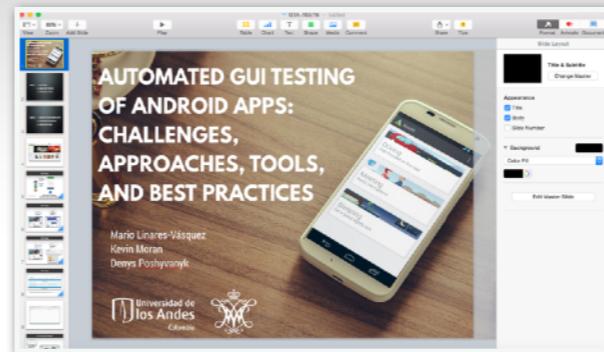
# The Importance of GUI Testing

- For Mobile, GUI-Based Testing subsumes many other types of testing
- GUI-Testing is typically expensive, and test scripts are difficult to maintain
- There is a clear opportunity for automation to Improve development workflows

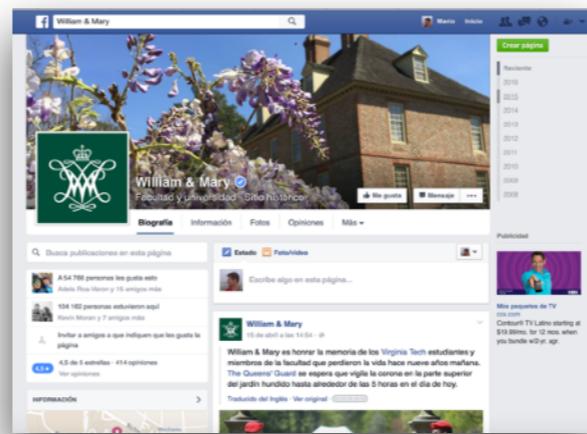
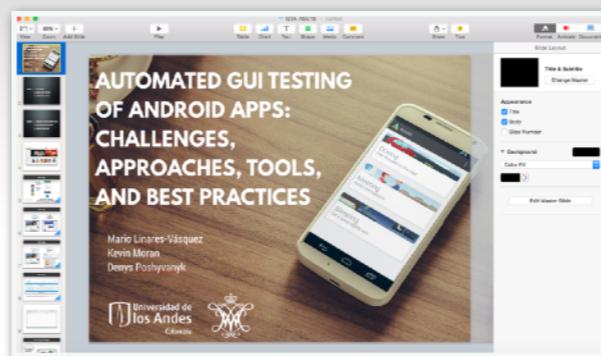
# GUI Testing: The Main Idea



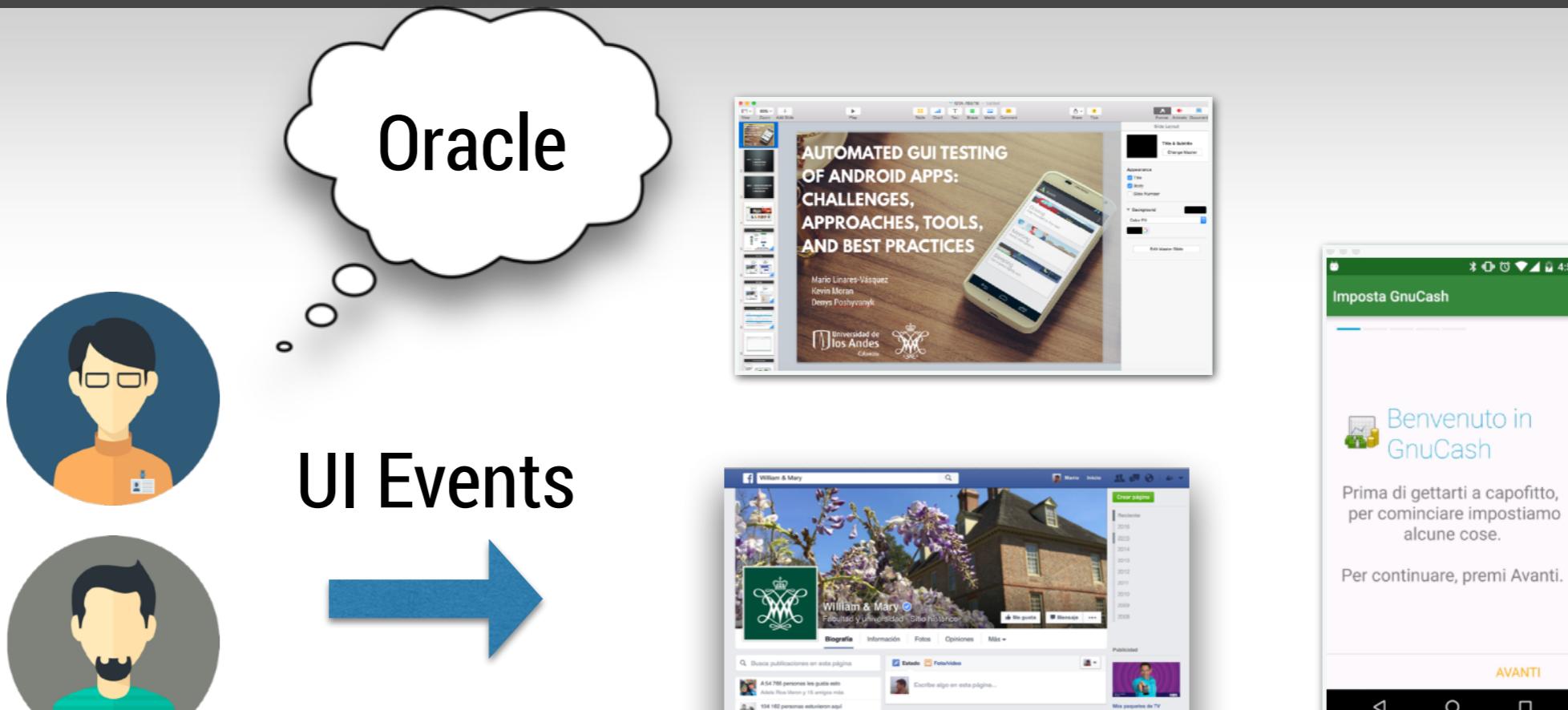
UI Events



# GUI Testing: The Main Idea

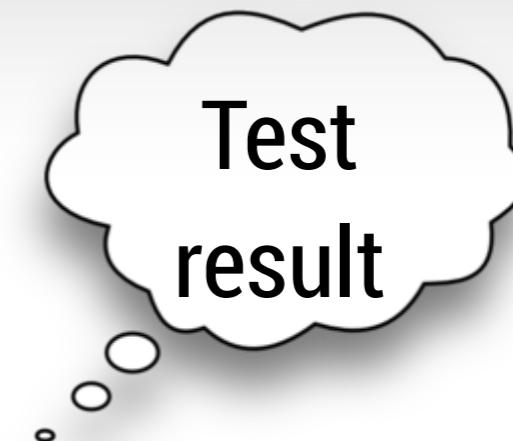


# GUI Testing: The Main Idea



Output, layout, exceptions,  
presentation logic, quality attributes, ...

# GUI Testing: Core Concepts



A screenshot of the official Facebook page for William &amp; Mary. The page features a large header image of a building with purple flowers in the foreground. Below the header, there's a green sidebar with the university's crest logo. The main content area shows various posts, including one from Adela Rios-Varon and another from William &amp; Mary about TV packages. The page has over 54,000 likes and 104,000 visitors.

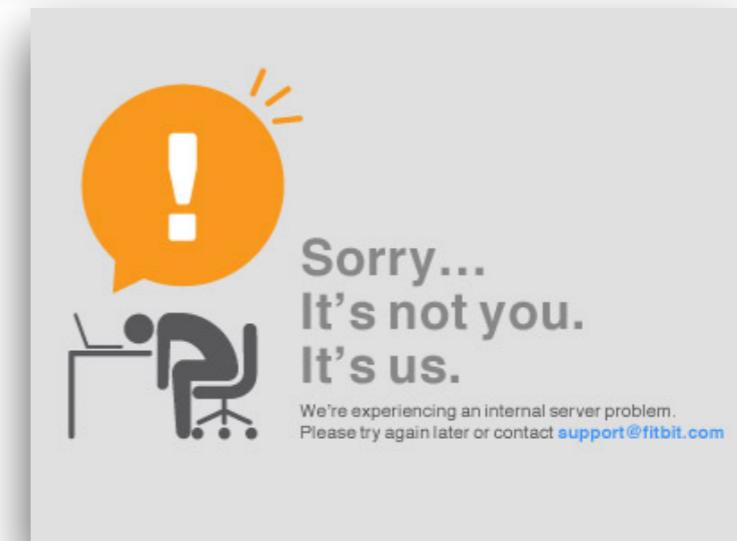
A screenshot of a generic Facebook page titled "Your page name". The page has a blue header and a blue cover photo. It includes sections for "About Your page name", "Timeline", and "About". There are also tabs for "Welcome", "FWB", and "More". The sidebar on the right shows notifications and messages, typical of a user's profile view.

# GUI Testing: Core Concepts



A screenshot of a Facebook page for "William &amp; Mary". The page features a large purple wisteria flower banner at the top. Below it, there's a profile picture of a building, a blue "William &amp; Mary" logo with a checkmark, and the text "Facultad y universidad · Sitio histórico". There are tabs for "Biografía", "Información", "Fotos", "Opiniones", and "Más". On the left, there's a sidebar with stats like "A 54 766 personas les gusta esto" and "104 162 personas estuvieron aquí". The main content area shows a post from "William &amp; Mary" about honoring Virginia Tech students.

A screenshot of a 500 Internal Server Error page. It features a large blue "500" and the text "Unexpected Error :(" below it. A smaller message at the bottom says "An error occurred and your request couldn't be completed. Please try again.".



# GUI Testing: Example

The image shows two screenshots of the Twitter Help Center interface, labeled (a) English version and (b) Russian version.

**(a) English version:** The top screenshot shows the English version of the Twitter Help Center. It features a white header with a blue Twitter logo and the text "Help Center". On the right side of the header are a search bar, language selection ("English"), and account links ("Got an account?", "Sign in"). Below the header is a navigation bar with links: "Welcome to Twitter", "Me", "Notifications", "Discover", "Mobile & Apps", and "Troubleshooting". A large blue horizontal bar runs across the middle of the page.

**(b) Russian version:** The bottom screenshot shows the Russian version of the Twitter Help Center. It has a similar layout but with俄语text. The header includes a blue Twitter logo and the text "Справочный центр". The language selection shows "Русский" and account links in Russian. The navigation bar includes links: "Добро пожаловать в Твиттер", "Я", "Уведомления", "В курсе", and "Мобильные устройства и приложения". A link "Поиск и устранение неисправностей" is visible below the navigation bar. A large blue horizontal bar is at the bottom of the page.

**Detecting and Localizing Internationalization Presentation Failures in Web Applications.** Abdulmajeed Alameer, Sonal Mahajan, William G.J. Halfond. In Proceeding of the 9th IEEE International Conference on Software Testing, Verification, and Validation (ICST). April 2016.

# GUI Testing: Example

The image shows two screenshots of the Twitter Help Center interface, labeled (a) English version and (b) Russian version.

**(a) English version:** The top screenshot shows the English version of the Twitter Help Center. It features a white header with a blue Twitter logo and the text "Help Center". On the right side of the header are a search bar, language selection ("English"), and account links ("Got an account?", "Sign in"). Below the header is a navigation bar with links: "Welcome to Twitter", "Me", "Notifications", "Discover", "Mobile & Apps", and "Troubleshooting". A large blue horizontal bar runs across the middle of the page.

**(b) Russian version:** The bottom screenshot shows the Russian version of the Twitter Help Center. It has a similar layout but with Russian text. The header includes a blue Twitter logo and the text "Справочный центр". The navigation bar links are: "Добро пожаловать в Твиттер", "Я", "Уведомления", "В курсе", and "Мобильные устройства и приложения". A yellow dashed rectangular box highlights the "Поиск и устранение неисправностей" link under the "Я" section. A large blue horizontal bar runs across the middle of the page.

**Detecting and Localizing Internationalization Presentation Failures in Web Applications.** Abdulmajeed Alameer, Sonal Mahajan, William G.J. Halfond. In Proceeding of the 9th IEEE International Conference on Software Testing, Verification, and Validation (ICST). April 2016.

Excel File Edit View Insert Format Tools Data Window Help

93% Sun 3:34 PM

Workbook1

Home Insert Page Layout Formulas Data Review View

Cut Calibri (Body) 12 A A Wrap Text General

Paste Copy B I U Merge & Center \$ % .00 Conditional Insert Delete Format

Format as Table Cell Styles AutoSum Fill Sort & Filter Clear

A1 fx

1 A B C D E F G H I J K L M N O P Q R S T U V

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

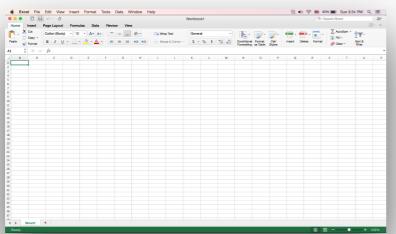
38

Sheet1 +

Ready 100%

The screenshot shows a clean, empty Excel spreadsheet. The title bar indicates it's 'Workbook1' and the status bar shows the date and time as 'Sun 3:34 PM'. The ribbon at the top has tabs for Home, Insert, Page Layout, Formulas, Data, Review, and View. The Home tab is active, displaying the ribbon bar with various tools for text, tables, and formulas. The main workspace is a grid of 16 columns (A-V) and 38 rows (1-38). Cell A1 is highlighted with a green border, indicating it is the active cell. The formula bar above the grid shows 'fx' and the cell reference 'A1'. The bottom navigation bar includes tabs for 'Sheet1' and '+', along with zoom controls and a 'Ready' status message.

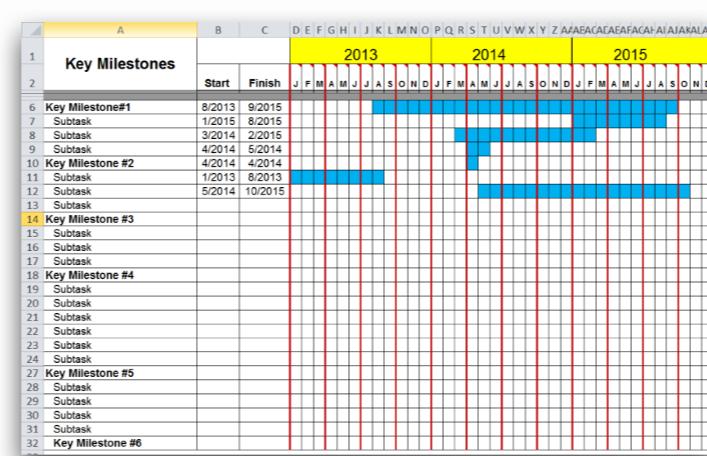
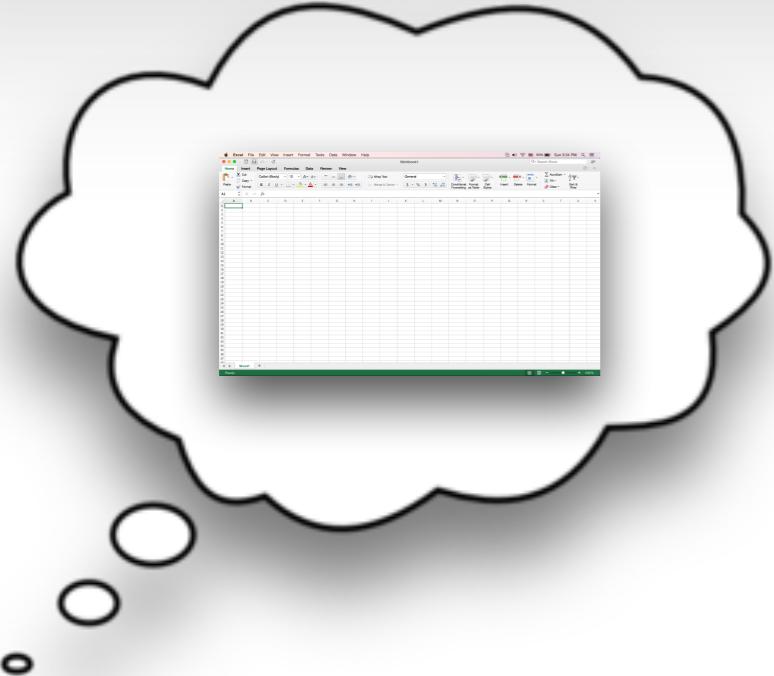
# GUI Testing (Challenges)



1	Key Milestones	A	B	C	D E F G H I J K L M N O P Q R S T U V W X Y Z	AA AB AC AD AE AF AC AH AI AJ AK AL AN		
		Start	Finish	J F M A M J J A S O N D J F M A M J J A S O N D J F M A M J J A S O N D	2013	2014	2015	
6	Key Milestone#1	8/2013	9/2015					
7	Subtask	1/2015	8/2015					
8	Subtask	3/2014	2/2015					
9	Subtask	4/2014	5/2014					
10	Key Milestone #2	4/2014	4/2014					
11	Subtask	1/2013	8/2013					
12	Subtask	5/2014	10/2015					
13	Subtask							
14	Key Milestone #3							
15	Subtask							
16	Subtask							
17	Subtask							
18	Key Milestone #4							
19	Subtask							
20	Subtask							
21	Subtask							
22	Subtask							
23	Subtask							
24	Subtask							
27	Key Milestone #5							
28	Subtask							
29	Subtask							
30	Subtask							
31	Subtask							
32	Key Milestone #6							

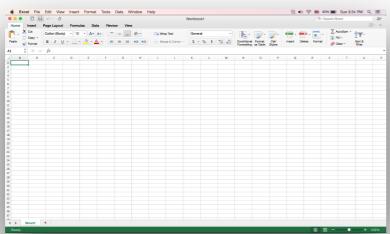
Inputs: combinatorial explosion

# GUI Testing (Challenges)



Inputs: combinatorial explosion

# GUI Testing (Challenges)

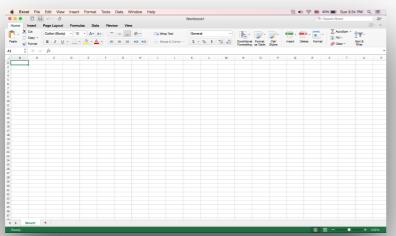


The screenshot shows a Microsoft Excel spreadsheet titled "Product Cost - 营业成本". The spreadsheet tracks daily food expenses from January 1st to January 25th. The columns represent various food items: 菜水黑 (Cai Shui He), 肉类 (Meat), 面米饭 (Nan Fan Mi), 鸡蛋 (Ji Dan), 香油 (Xiang You), 调料 (Diao Liao), 饮料 (Yin Liao), 薪罐食品 (Xin Kan Pin Sheng), 面包 (Mian Bao), 奶脂/黄油 (Nai Zi / Huang You), and 茶类 (Cha Liao). The "Total" column shows the sum of all items for each day, and the "工资" (Wage) column shows the total wage for each day. The data is as follows:

日期	Product Cost - 营业成本										工资
	Food - 食物贮备	Total									
1日	29.5	85	36		60	43.9	74.4			33	1105
2日	8.2	33					18				
3日	11.7	33									
4日	19	64.5						45			
5日	10	21.6				3					
6日	5	37		17		55		133.5	30		
7日	21.6	33	56				10				
8日	10	24.8					10				
9日	18			12							
10日		22									
11日	16.6	20					15				
12日		24									
13日	21.8	23	35				10				
14日	1						15				
15日	9	22		12							
16日											
17日	14.8	20									
18日	10	23			25						
19日					25						
20日	2	25	54	12.8		220.5	114.3	65		6	
21日											

Internationalization

# GUI Testing (Challenges)



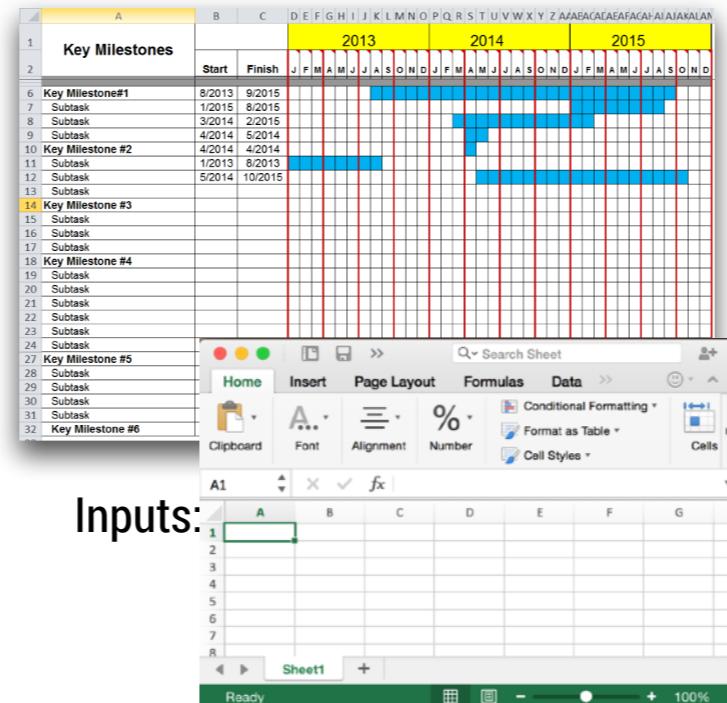
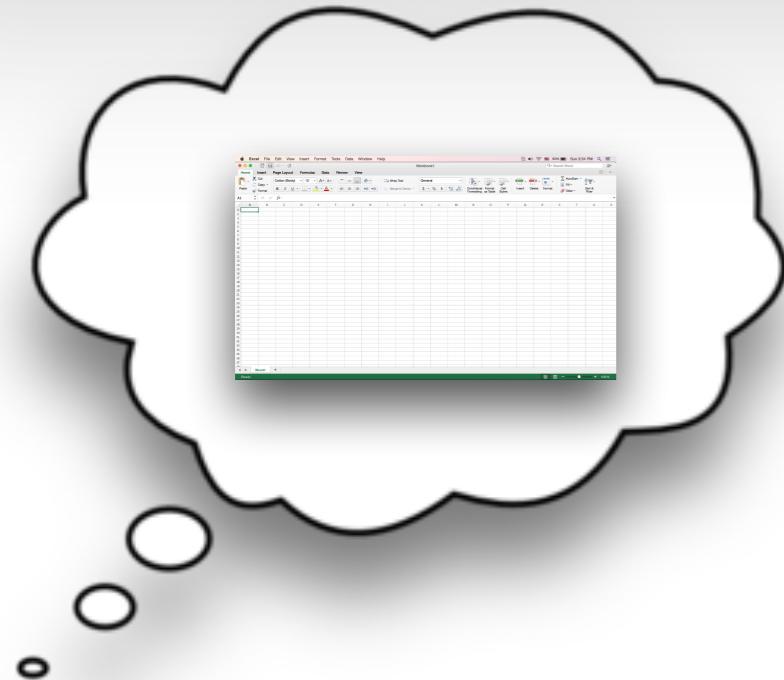
Key Milestones	Start	Finish	2013			2014			2015		
			J	F	M	A	M	J	A	S	O
6 Key Milestone #1	8/2013	9/2015									
7 Subtask	1/2015	8/2015									
8 Subtask	3/2014	2/2015									
9 Subtask	4/2014	5/2014									
10 Key Milestone #2	4/2014	4/2014									
11 Subtask	1/2013	8/2013									
12 Subtask	5/2014	10/2015									
13 Subtask											
14 Key Milestone #3											
15 Subtask											
16 Subtask											
17 Subtask											
18 Key Milestone #4											
19 Subtask											
20 Subtask											
21 Subtask											
22 Subtask											
23 Subtask											
24 Subtask											
27 Key Milestone #5											
28 Subtask											
29 Subtask											
30 Subtask											
31 Subtask											
32 Key Milestone #6											

Inputs: combinatorial explosion

Product Cost - 营业成本											
Food - 食物类											
	G	H	I	J	K	L	M	N	O	P	R
3 日期											
4											
5 1日	29.5	85	36								
6 2日	8.2	33									
7 3日	11.7	33									
8 4日	19	64.5									
9 5日	10	21.6									
10 6日	5	37	17								
11 7日	21.6	33	56								
12 8日	10	24.8									
13 9日	18		12								
14 10日											
15 11日	16.6	20									
16 12日		24									
17 13日	21.8	23	35								
18 14日	1										
19 15日	9	22	12								
20 16日											
21 17日	14.8	20									
22 18日	10	23									
23 19日											
24 20日	2	25	54 12.8	220.5	114.3	65					
25											
Total											
工时											
工时											
1105											
33											
6											

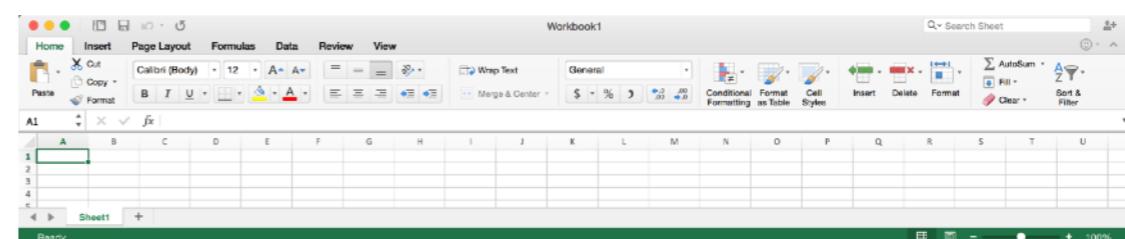
Internationalization

# GUI Testing (Challenges)



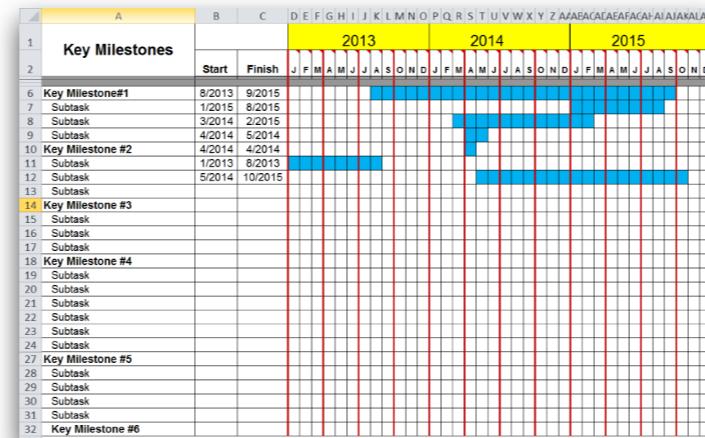
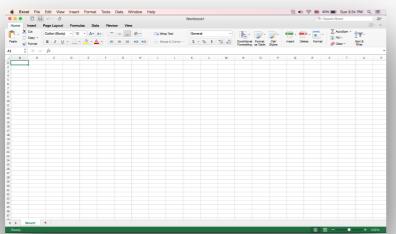
	G	H	I	J	K	L	M	N	O	P	Q	R
	日期											
1	深水果	动态	固定版	鸡蛋	酱油	调料	饮料	麦片食品	面包	酸奶/豆油	盐	工时
2	1日	29.5	85	36	60	43.9	74.4					1105
3	2日	8.2	33	18								
4	3日	11.7	33									
5	4日	19	64.5									
6												45
7												10
8												10
9												15
10												15
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												
23												
24												
25												
26												
27												
28												
29												
30												
31												
32												

ationalization



Responsive design

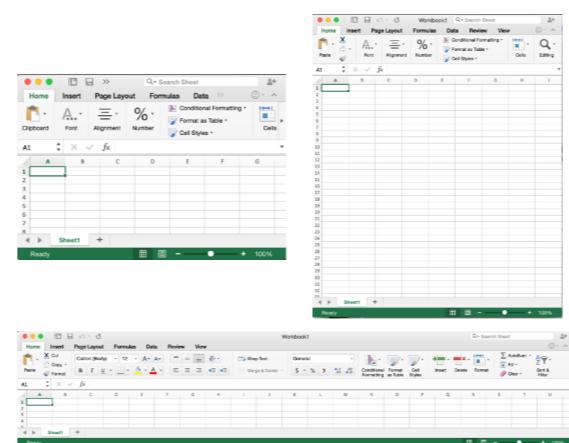
# GUI Testing (Challenges)



Inputs: combinatorial explosion

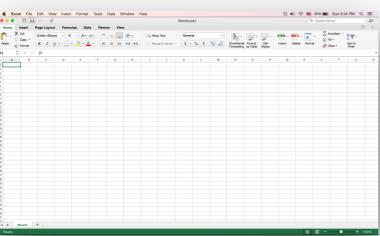
A screenshot of Microsoft Excel showing a complex expense sheet titled 'Product Cost - 营业成本'. The table has many columns for different categories like 'Food - 食物' and 'Total 工资'. It contains numerous rows of data with numerical values, demonstrating the complexity of the input data.

Internationalization

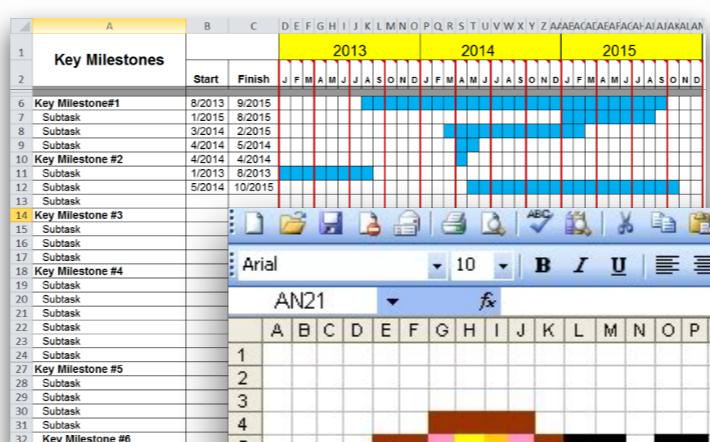


Responsive design

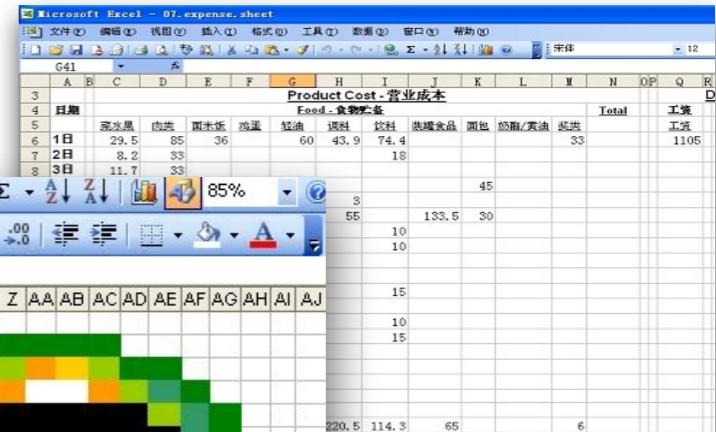
# GUI Testing (Challenges)



## Inputs:



## Nationalization

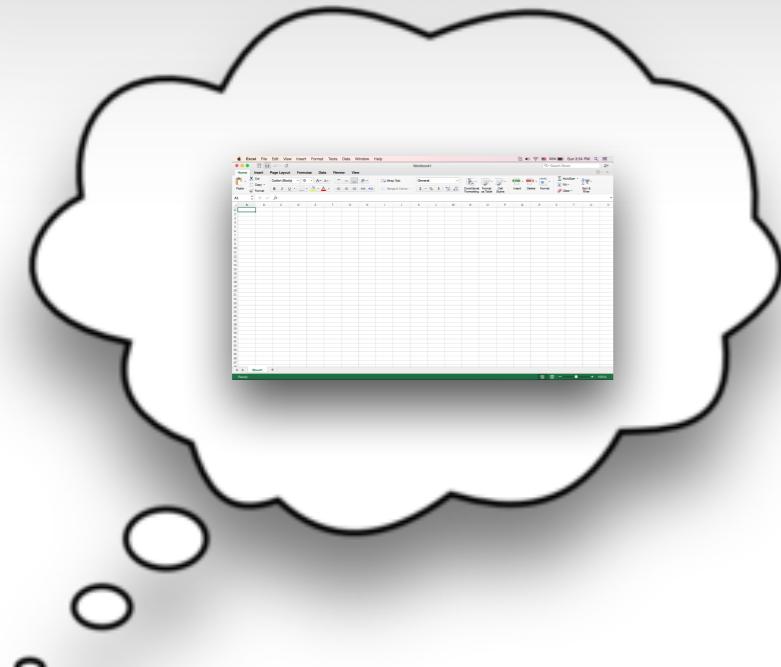


A pixel art version of the Mario Kart 8 character Bowser, rendered in Microsoft Excel. The character is a large, tan, multi-headed dragon with black stripes, red mohawks, and a blue shell on its back. He is standing on a black base. The image is set against a background of a white grid with numerical labels from 1 to 34 on the left and top axes.

# Unexpected usage scenarios

Responsive design

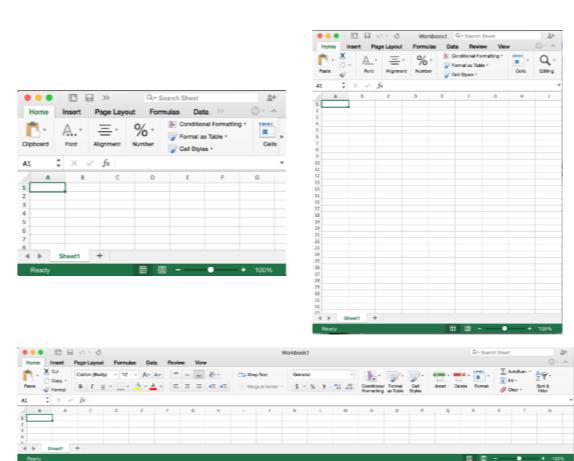
# GUI Testing (Challenges)



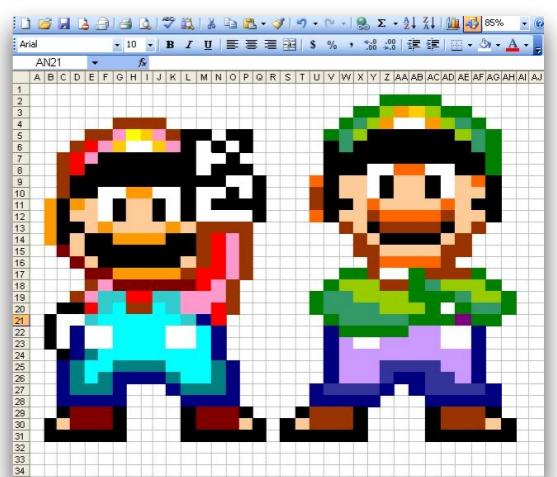
## Inputs: combinatorial explosion

Product Cost - 营业成本													
日期	Food - 食物成本											Total	工资 工时
	薪水	肉桂	重味盐	鸡蛋	酱油	速食	饮料	烘焙食品	面包	防腐/黄油	蔬菜		
6月 1日	29.5	85	36		60	43.9	74.4					33	1105
2日	8.2		33					18					
3日	11.7	33											
4日	19	64.5							45				
5日	10	21.6			3								
6日	5	37		17		55			133.5	30			
7日	21.6	33	56					10					
8日	10	24.8						10					
9日	18			12									
10日		22											
11日	16.6	20						15					
12日		24											
13日	21.8	23	35					10					
14日	1							15					
15日	9	22		12									
16日													
17日	14.8	20											
18日	10	23			25								
19日					25								
20日	2	25	54	12.8		220.5	114.3	65			6		

# Internationalization



# Responsive design



## Unexpected usage scenarios

# MONKEY TESTING !!



~~MONKEY TESTING !!~~

**AUTOMATED TESTING !!**



# Automated GUI Testing



UI Events →

Monkey



Output, layout, exceptions, presentation  
logic, quality attributes, ...



# ANDROID GUI TESTING

# Unique Challenges in Mobile Development



# Thousands of apps are released and updated every day on the online store



# 2.8M apps - Google Play

# 65B downloads - Google Play

# 25 releases (Android) since 2008

# Large volume of crowdsourced requirements and ratings

**Benjamin Sanchez** September 14, 2015  
★★★★★  
**Keeps skipping my music** listening to a song, and it just itself. Is there something I can do on the settings for that to stop?

**April Senchuk** September 13, 2015  
★★★★★  
**What happened to Spotify?** Once upon a time when I wanted to hear a specific song, I could do that. Now it starts these playlists that start with stuff I like, but then play random stuff I'd never have chosen. Why can't I just listen to the song I have stuck in my head?

**Ian Sammut** September 14, 2015  
★★★★★  
**Some help maybe?** I really like Spotify but recently i cant change song from lock screen or change songs from my Bluetooth car stereo because of this. I tried my phone to try and nothing happened

**Bosco Wong** September 14, 2015  
★★★★★  
**I can't pay** The app is great but comes to payment, you only have option to pay in credit card by visa or mastercard, but most banks in hong Kong does not support either one, and uses Union Pay and EPS, I don't have those cards.

**eva yadmeiri olivares martinez** September 12, 2015  
★★★★★  
**Wouldve given 5 but its taking too much internal memory** I will give it a 5 if they would allow for app to be saved in background

**Saumitra Dixit** September 12, 2015  
★★★★★  
**Not logging in. Says offline.** First it started with saying no network connection and said "offline". Reinstalled after trying out the settings. Now it won't even log in.

**Albert Rinck** September 11, 2015  
★★★★★  
**Doesnt have some music** It doesn't have any taylor swift , & not the original " God Help The Outcasts "

# Fragmentation at device and OS level



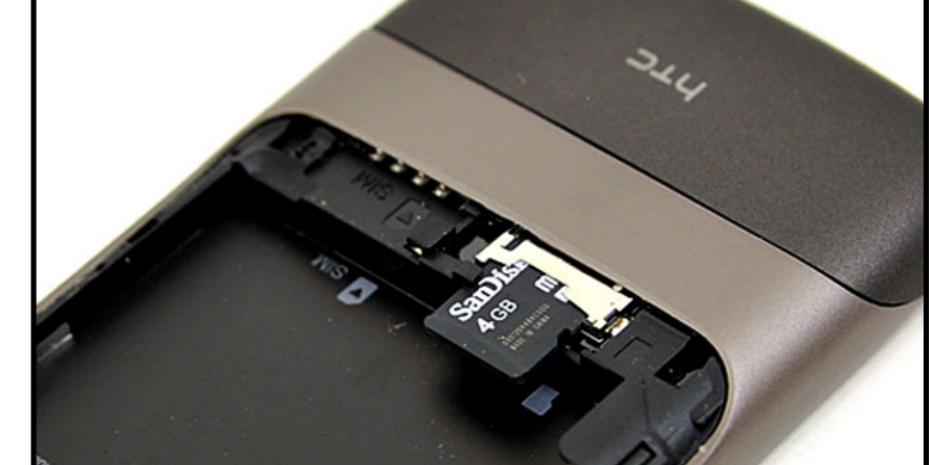
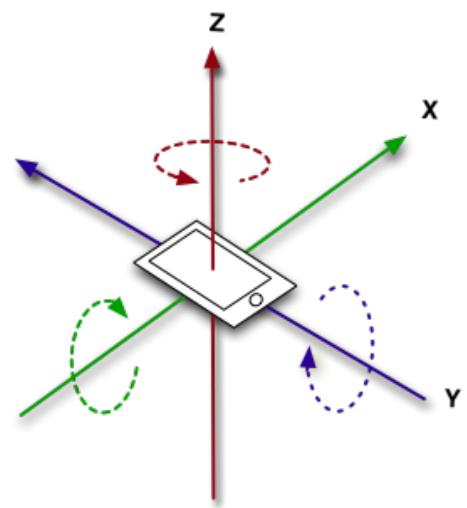


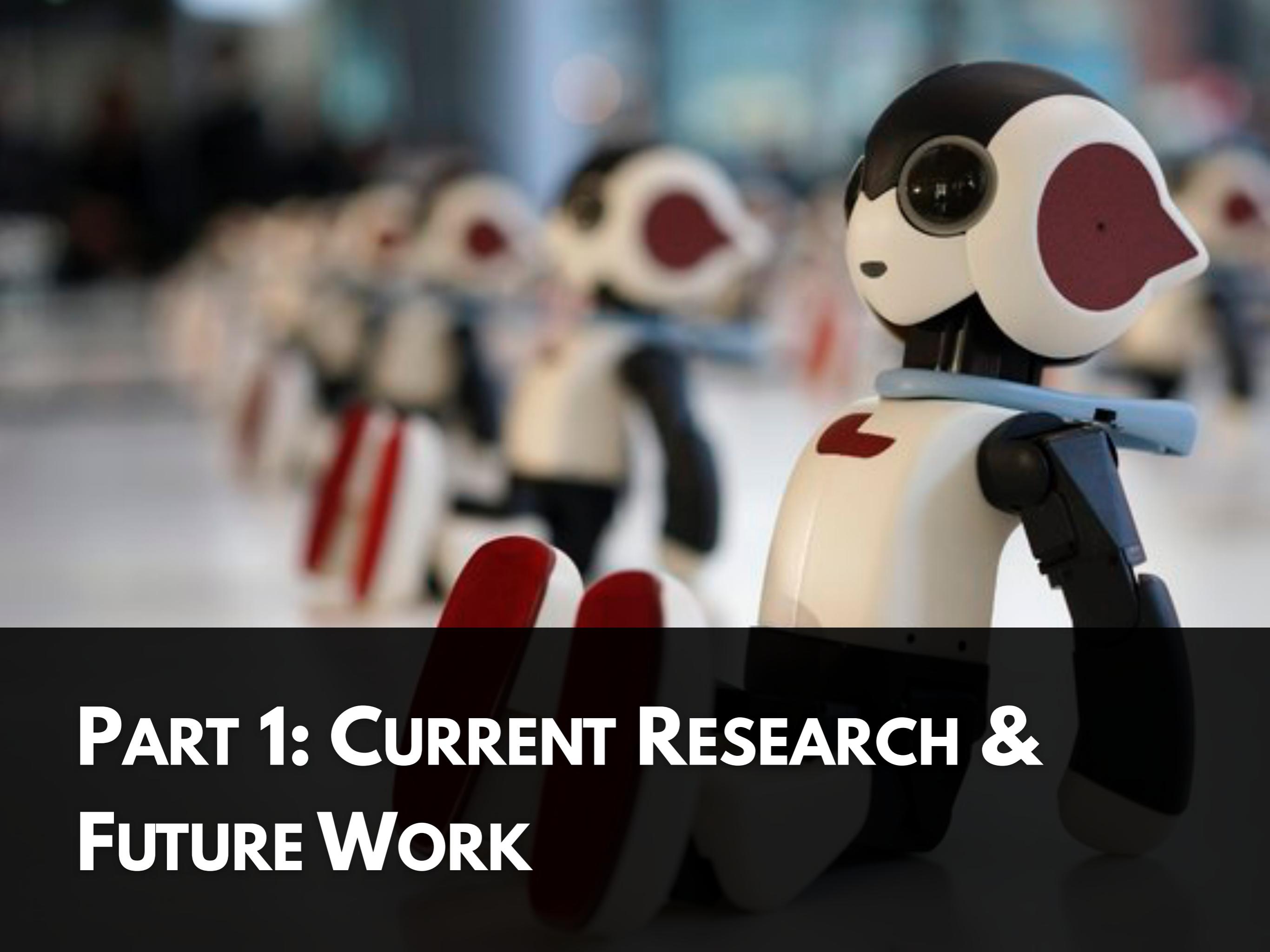
# Pressure for continuous delivery

# Manual testing is still preferred



# Mobile-specific quality attributes, inputs, and scenarios





# PART 1: CURRENT RESEARCH & FUTURE WORK

# Overview of Tools & Services

# Overview of Tools & Services

- Automation Frameworks & APIs

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting
- Crowdsourced Testing

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting
- Crowdsourced Testing
- Cloud Testing Services

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting
- Crowdsourced Testing
- Cloud Testing Services
- Device Streaming Tools

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting
- Crowdsourced Testing
- Cloud Testing Services
- Device Streaming Tools



Traditional Android Testing  
Tools and Approaches

# Overview of Tools & Services

- Automation Frameworks & APIs
- Record & Replay Tools
- Automated Input Generation Tools
- Bug & Error Reporting
- Crowdsourced Testing
- Cloud Testing Services
- Device Streaming Tools



Traditional Android Testing  
Tools and Approaches



Bug Reporting,  
Crowdsourcing and Services

# ANDROID TESTING TOOLS & APPROACHES

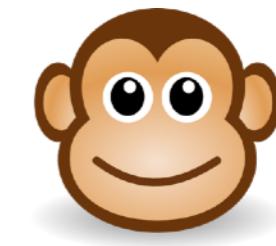


# Automation Frameworks/APIs (AF/A)

## TESTS



```
@Test  
public void autoCompleteTextView_oneSuggestion() {  
    // Type "South" to trigger one suggestion.  
    onView(withId(R.id.auto_complete_text_view))  
        .perform(typeTextIntoFocusedView("South "), closeSoftKeyboard());  
  
    // Should be displayed  
    onView(withText("South China Sea"))  
        .inRoot(withDecorView(not(is(mActivity.getWindow().getDecorView()))))  
        .check(matches(isDisplayed()));  
  
    // Should not be displayed.  
    onView(withText("Southern Ocean"))  
        .inRoot(withDecorView(not(is(mActivity.getWindow().getDecorView()))))  
        .check(doesNotExist());  
}
```

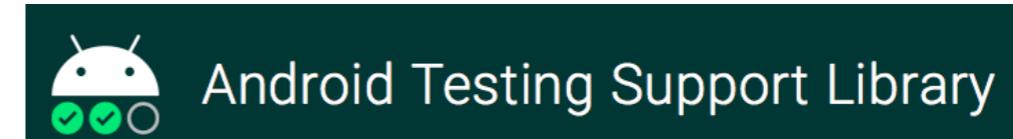
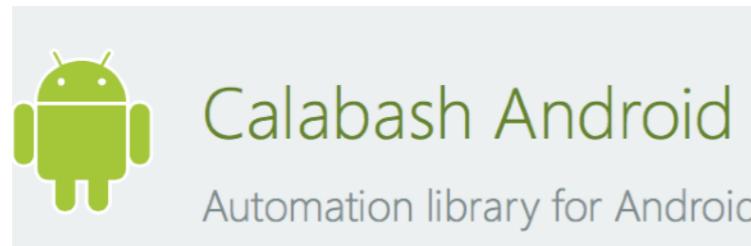


Monkey

```
@Test  
public void checkPreconditions() {  
    assertThat(mDevice, notNullValue());  
}  
  
@Test  
public void testChangeText_sameActivity() {  
    // Type text and then press the button.  
    mDevice.findObject(By.res(BASIC_SAMPLE_PACKAGE, "editTextUserInput"))  
        .setText(STRING_TO_BE_TYPED);  
    mDevice.findObject(By.res(BASIC_SAMPLE_PACKAGE, "changeTextBt"))  
        .click();  
  
    // Verify the test is displayed in the UI  
    UiObject2 changedText = mDevice  
        .wait(Until.findObject(By.res(BASIC_SAMPLE_PACKAGE, "textToBeChanged")),  
              500 /* wait 500ms */);  
    assertThat(changedText.getText(), is(equalTo(STRING_TO_BE_TYPED)));  
}
```

JUnit, Espresso, UI Automator, Robotium

# Testing Automation Frameworks/APIs



UI Automator

# Testing Automation Frameworks/APIs

The screenshot shows the GitHub repository page for 'googlesamples / android-testing'. The repository has 79 commits, 3 branches, 0 releases, and 17 contributors. The latest commit was made 29 days ago by JoseAlcerreca. The repository contains samples for different testing frameworks and techniques.

File	Description	Time Ago
integration/ServiceTestRuleSample	Updates Gradle plugin to 2.1.0-alpha3	29 days ago
runner/AndroidJUnitRunnerSample	Updates Gradle plugin to 2.1.0-alpha3	29 days ago
ui	Updates Gradle plugin to 2.1.0-alpha3	29 days ago
unit	Updates Gradle plugin to 2.1.0-alpha3	29 days ago
.gitignore	Adds Eclipse configuration files to gitignore	11 months ago
CONTRIBUTING.md	Update CONTRIBUTING.md	4 months ago
LICENSE	Adds top-level README and license-related files and fixes a link in t...	a year ago
README.md	Fix some weird typos in README	4 months ago
projects.conf	Added new script to run all tests, removed top-level build.gradle bec...	7 months ago
test_all.sh	Added new script to run all tests, removed top-level build.gradle bec...	7 months ago

<https://github.com/googlesamples/android-testing>

# Testing Automation Frameworks/APIs

```
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)));
}

@Test
public void changeText_newActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
        closeSoftKeyboard());
    onView(withId(R.id.activityChangeTextBtn)).perform(click());

    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)));
}
```

# Testing Automation Frameworks/APIs

```
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)));
}

@Test
public void changeText_newActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
        closeSoftKeyboard());
    onView(withId(R.id.activityChangeTextBtn)).perform(click());

    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)));
}
```

# Testing Automation Frameworks/APIs

```
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)));
}

@Test
public void changeText_newActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
        closeSoftKeyboard());
    onView(withId(R.id.activityChangeTextBtn)).perform(click());

    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)));
}
```

# Testing Automation Frameworks/APIs

```
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)));
}

@Test
public void changeText_newActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
        closeSoftKeyboard());
    onView(withId(R.id.activityChangeTextBtn)).perform(click());

    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)));
}
```

# Testing Automation Frameworks/APIs

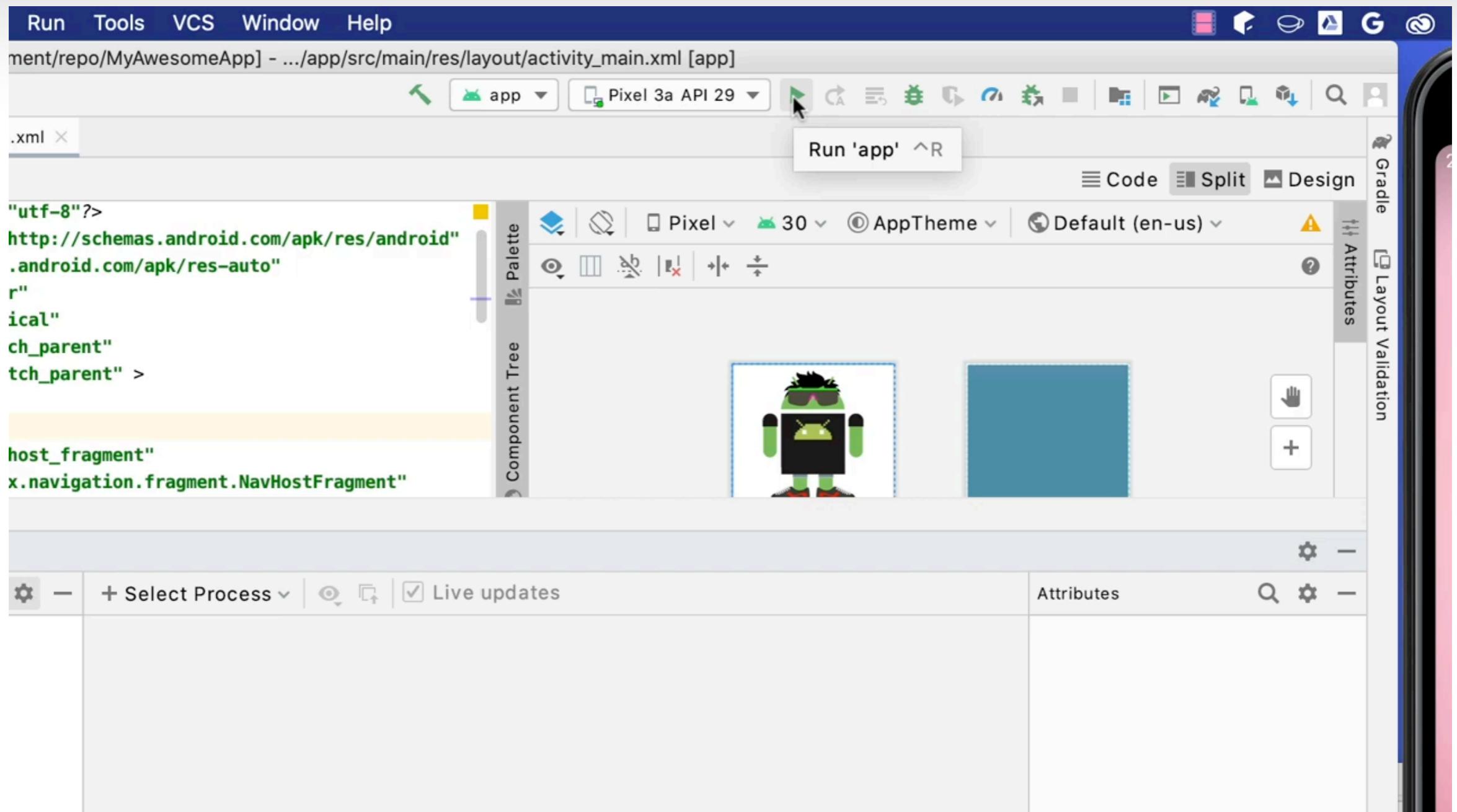
```
@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(STRING_TO_BE_TYPED), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged)).check(matches(withText(STRING_TO_BE_TYPED)));
}

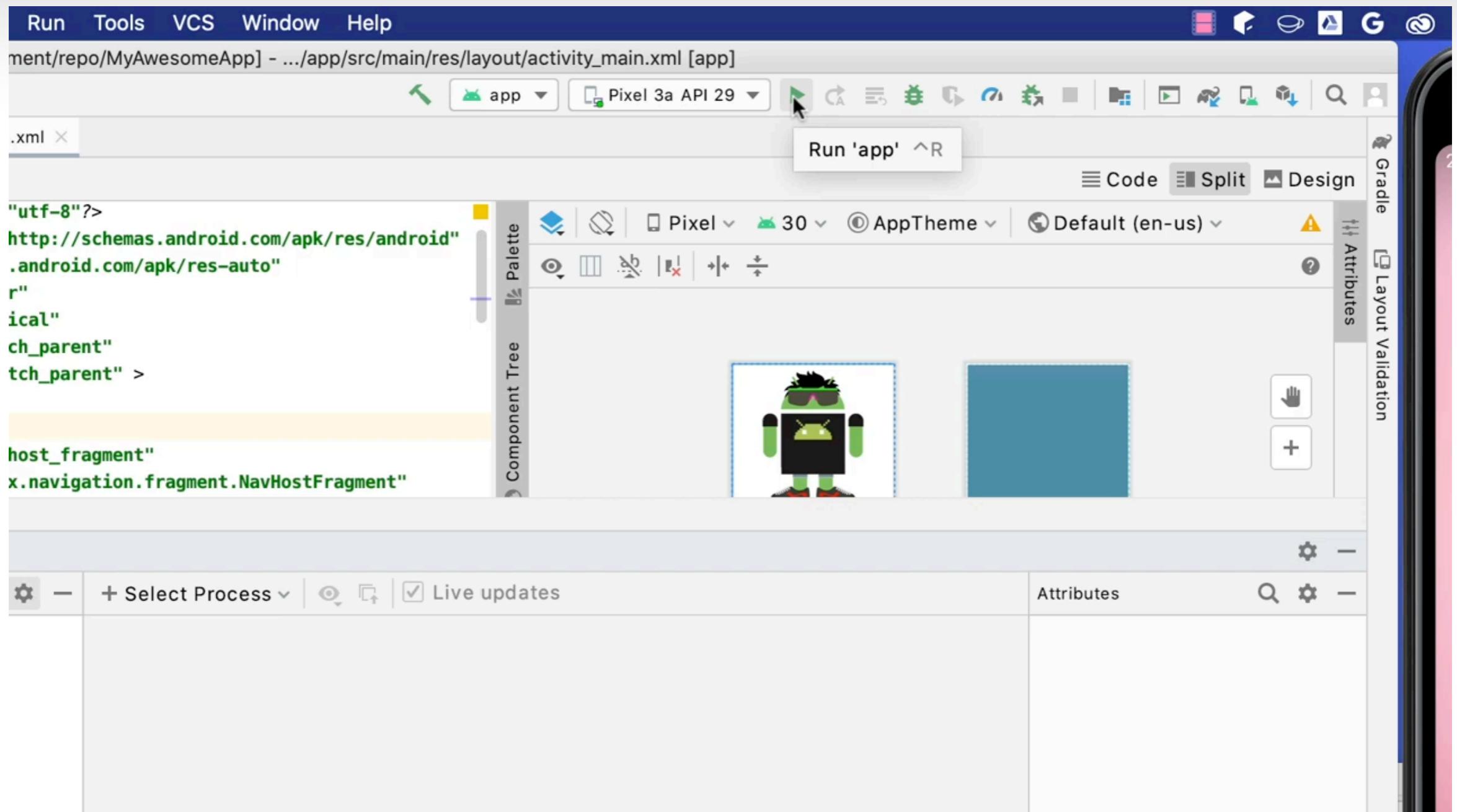
@Test
public void changeText_newActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput)).perform(typeText(STRING_TO_BE_TYPED),
        closeSoftKeyboard());
    onView(withId(R.id.activityChangeTextBtn)).perform(click());

    // This view is in a different Activity, no need to tell Espresso.
    onView(withId(R.id.show_text_view)).check(matches(withText(STRING_TO_BE_TYPED)));
}
```

# Tools: Layout Inspector



# Tools: Layout Inspector



# Automation Frameworks

## Pros and Cons



- ✓ Easy reproduction
- ✓ High level syntax
- ✓ Black box testing

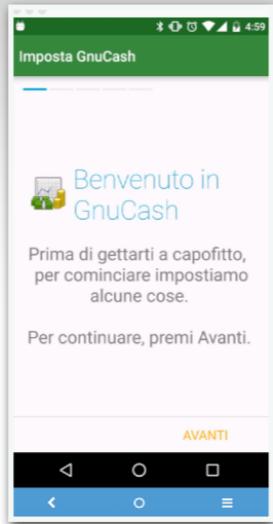
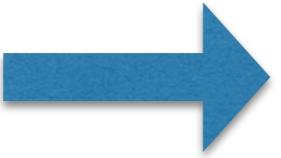


- Learning curve
- User-defined oracles
- Expensive maintenance

# Record and Replay (R&R)



UI Events



AUT/SUT

# Record and Replay (R&R)



UI Events



AUT/SUT



Recorder



Script

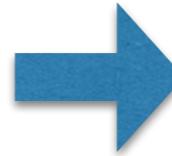
# Record and Replay (R&R)



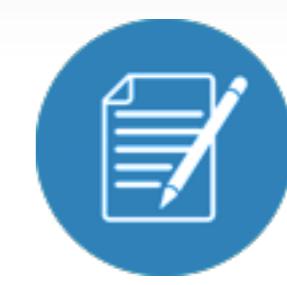
UI Events



AUT/SUT



Recorder



Script



Scripts

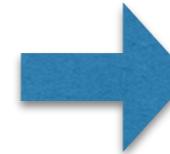
# Record and Replay (R&R)



UI Events



AUT/SUT



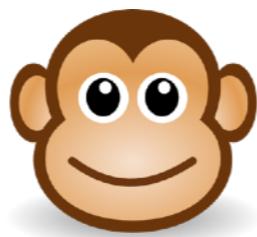
Recorder



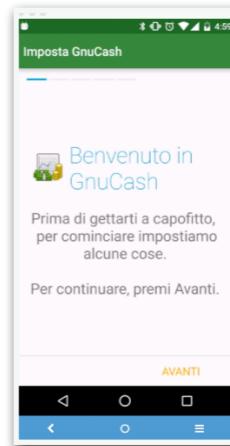
Script



UI Events



UI Events

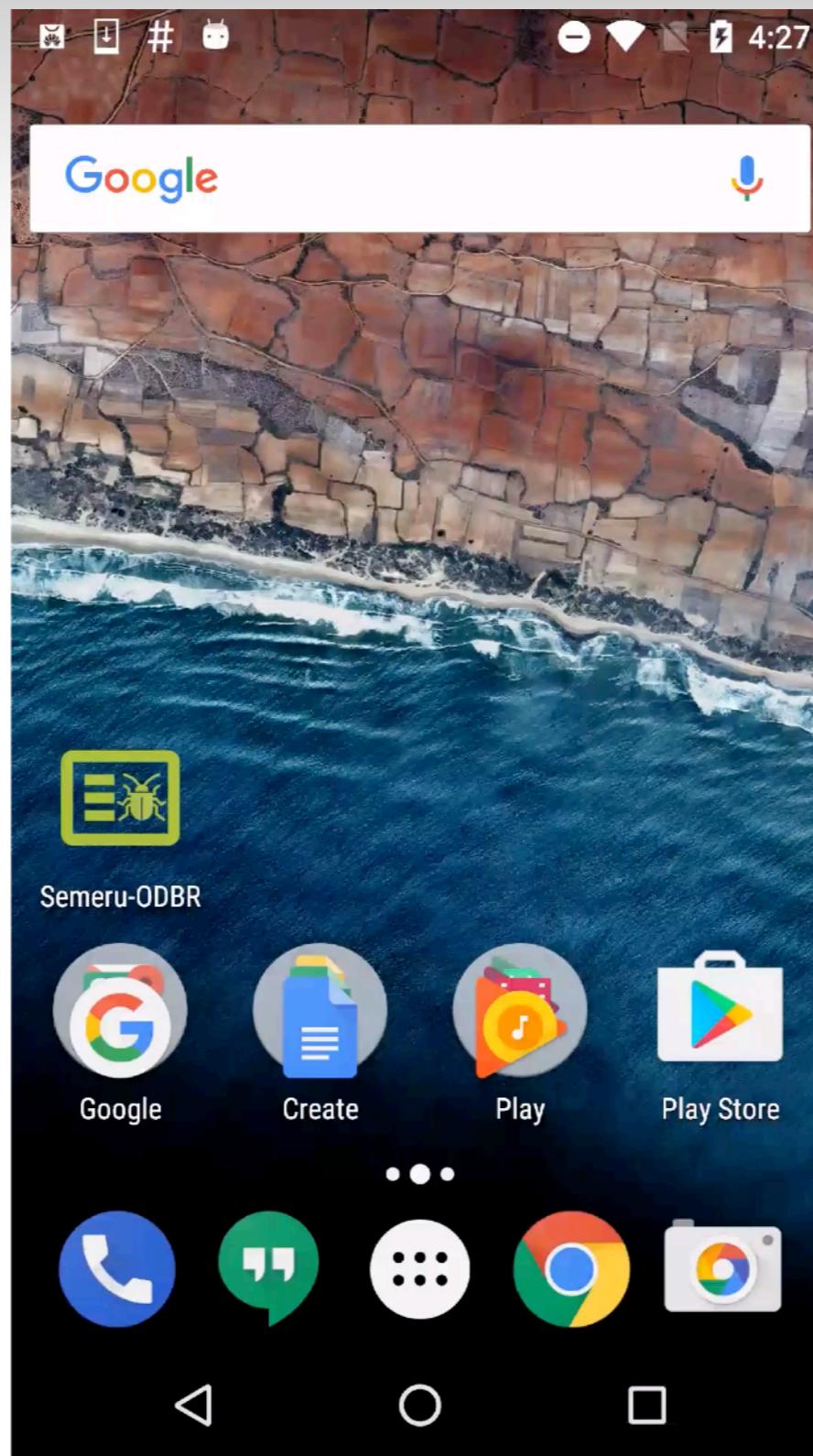


Scripts

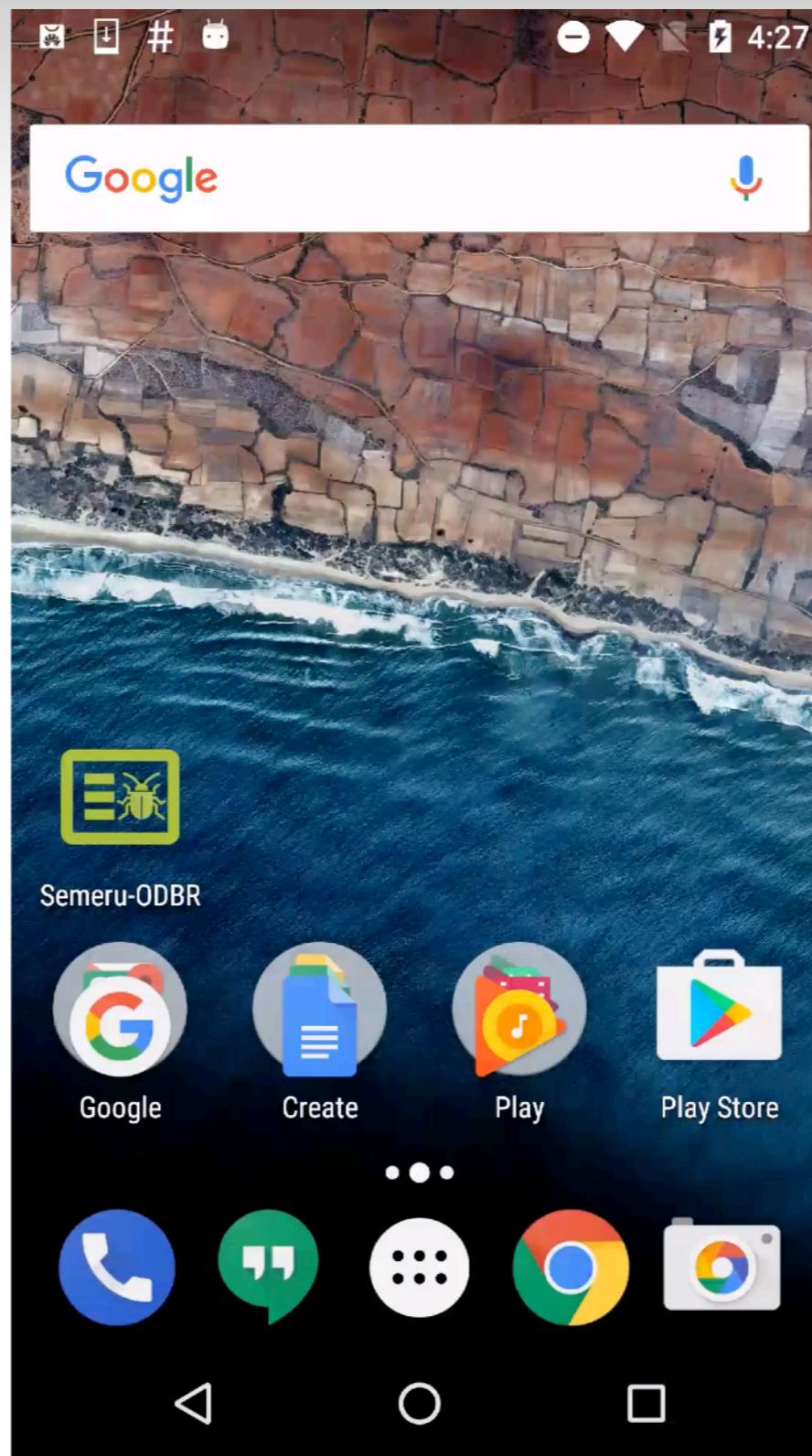
Monkey

AUT/SUT

# Tools: ODBR



# Tools: ODBR



## Automation Frameworks



- ✓ Easy reproduction
- ✓ High level syntax
- ✓ Black box testing



- Learning curve
- User-defined oracles
- Expensive maintenance

## Record & Replay

- ✓ Easy reproduction

- Expensive collection and maintenance
- Coupled to locations

# Automated Input Generation (AIG) Techniques



# Automated Input Generation (AIG) Techniques

- **Differing Goals:**

# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*

# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*
  - *Crashes*

# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*
  - *Crashes*
- **Three Main Types:**

# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*
  - *Crashes*
- **Three Main Types:**
  - *Random-Based*

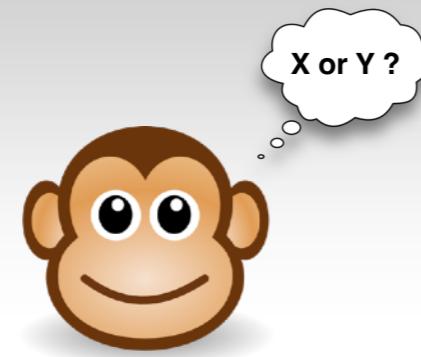
# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*
  - *Crashes*
- **Three Main Types:**
  - *Random-Based*
  - *Systematic*

# Automated Input Generation (AIG) Techniques

- **Differing Goals:**
  - *Code Coverage*
  - *Crashes*
- **Three Main Types:**
  - *Random-Based*
  - *Systematic*
  - *Model-Based*

# Random/Fuzz Testing (R/FT)

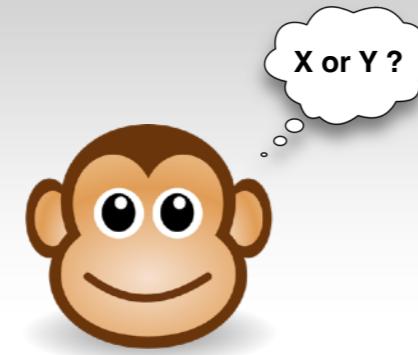


Monkey



AUT/SUT

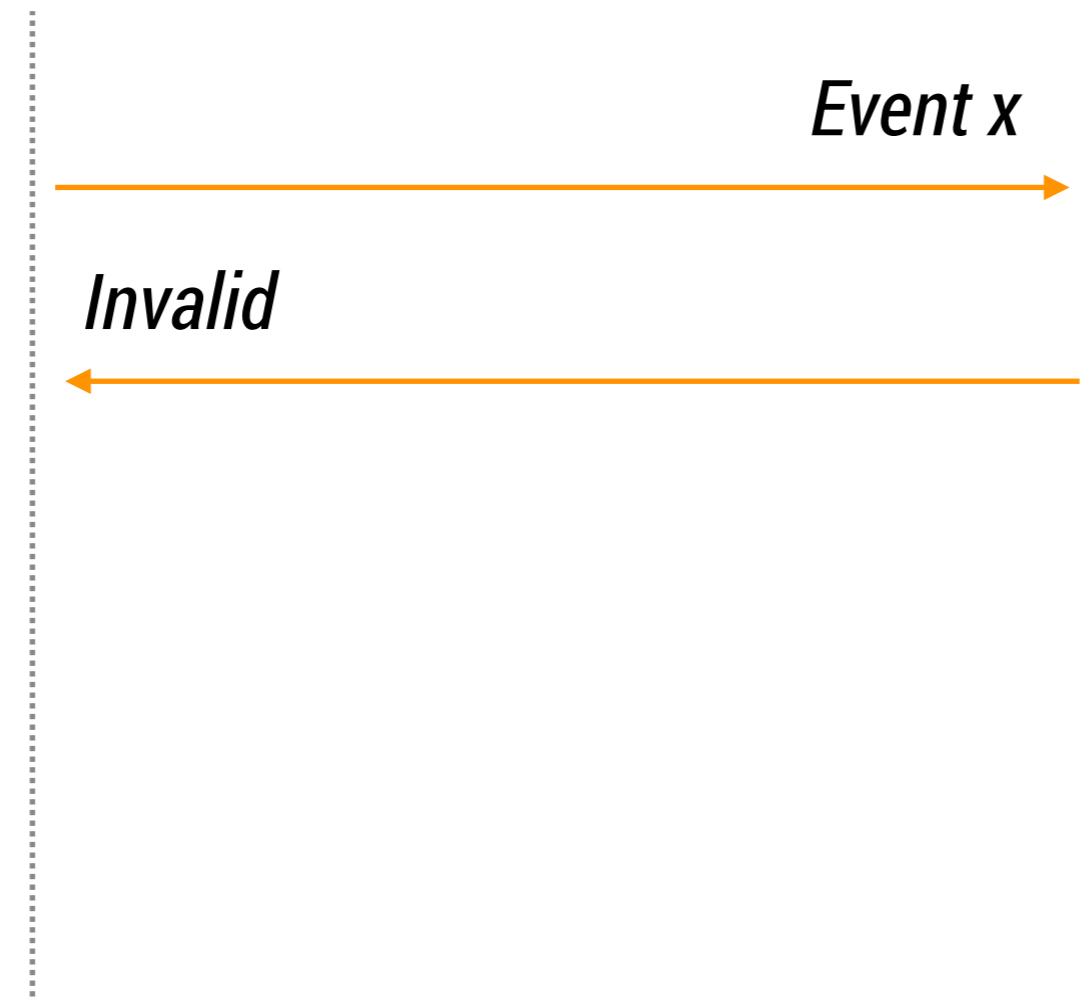
# Random/Fuzz Testing (R/FT)



Monkey



AUT/SUT



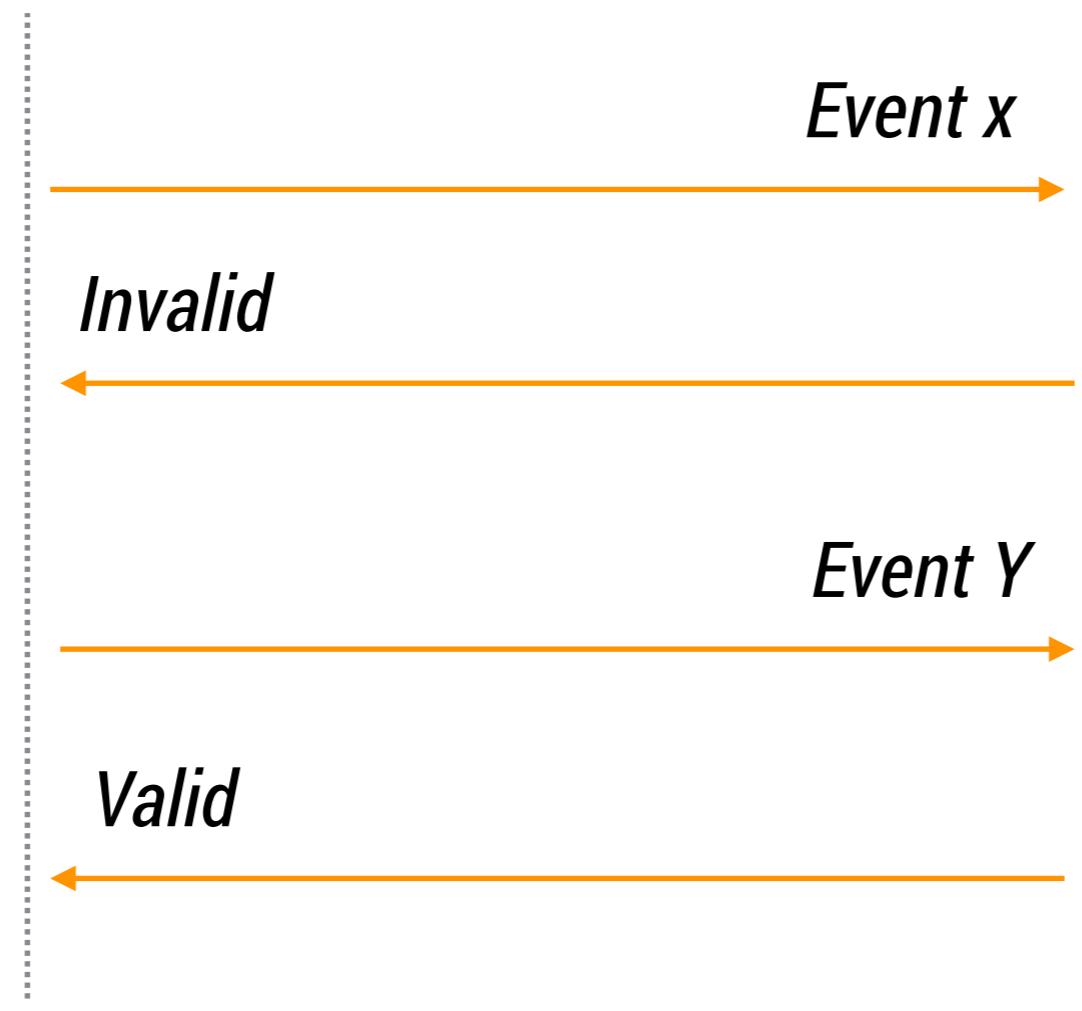
# Random/Fuzz Testing (R/FT)



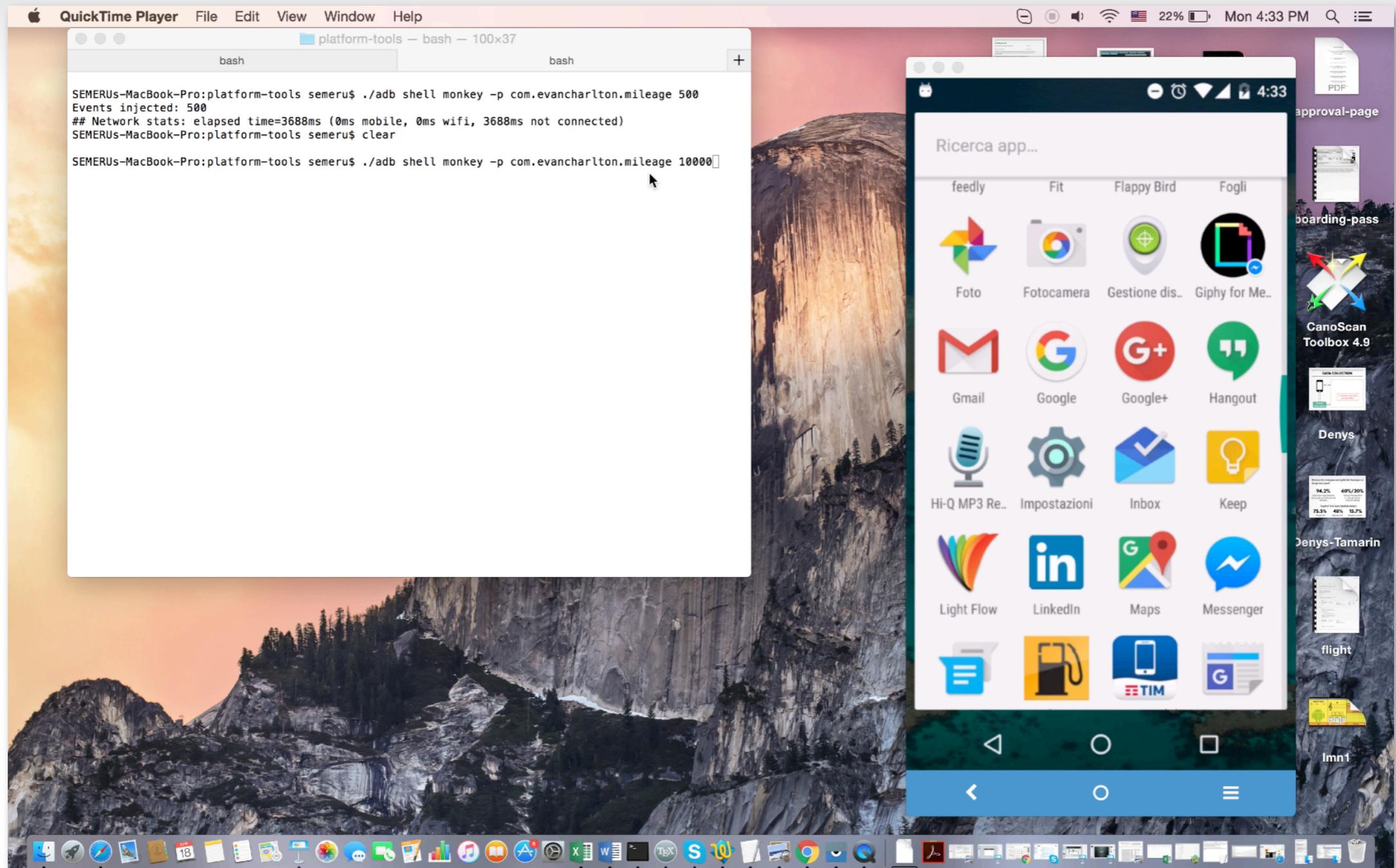
Monkey



AUT/SUT

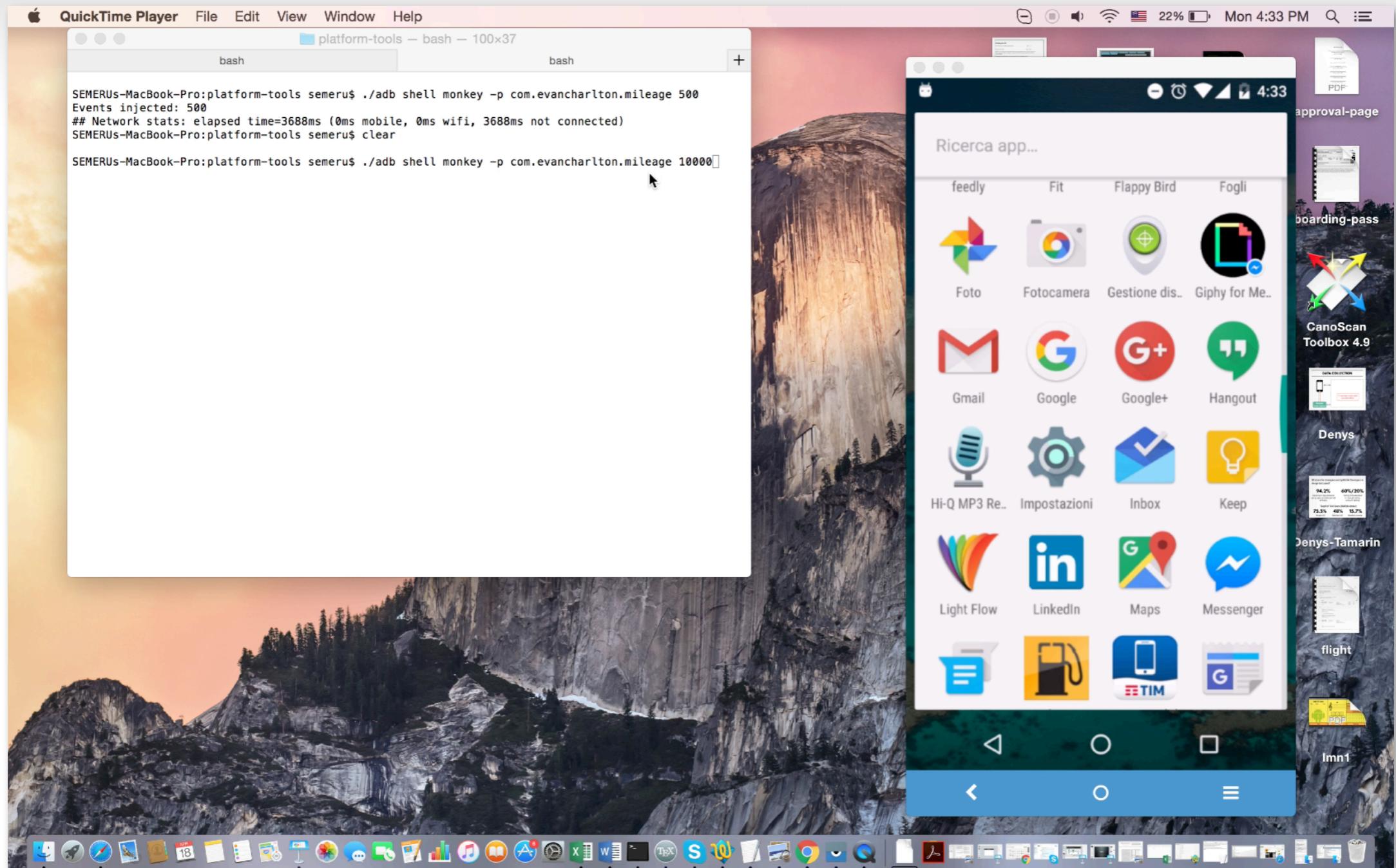


# Random/Fuzz Testing (R/FT)



./adb shell monkey -p com.evancharlton.mileage 10000

# Random/Fuzz Testing (R/FT)



./adb shell monkey -p com.evancharlton.mileage 10000

# Pros and Cons

## Automation Frameworks



- ✓ Easy reproduction
- ✓ High level syntax
- ✓ Black box testing



- Learning curve
- User-defined oracles
- Expensive maintenance

## Record & Replay

- ✓ Easy reproduction

- Expensive collection and maintenance
- Coupled to locations

## AIG: Random Based

- ✓ Fast execution
- ✓ Good at finding crashes

- Invalid events
- Lack of expressiveness

# Aside: GUI Ripping



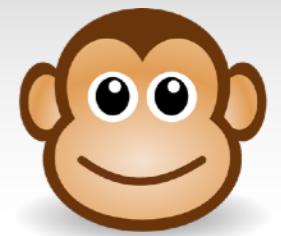
AUT/SUT



Ripper/Extractor

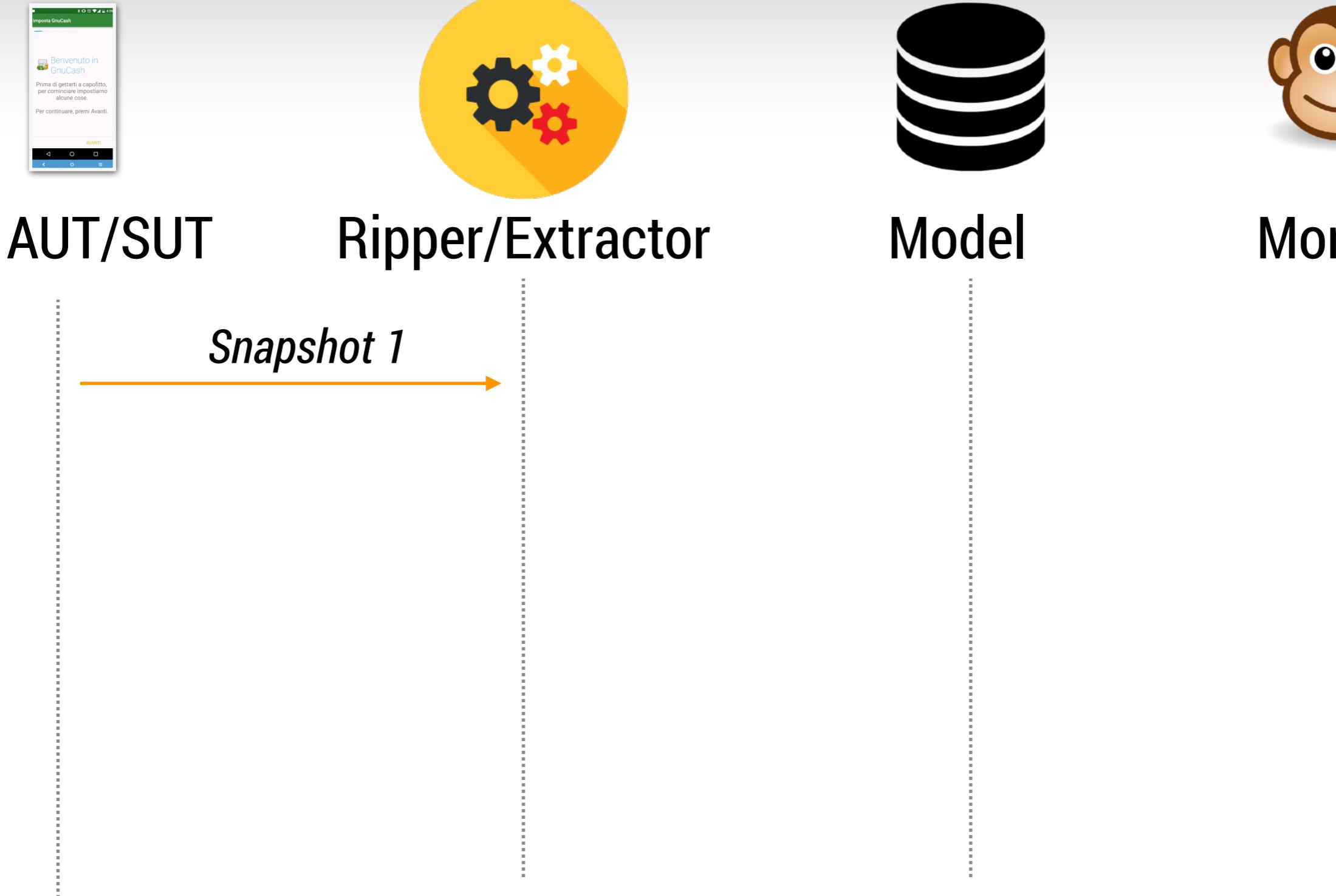


Model

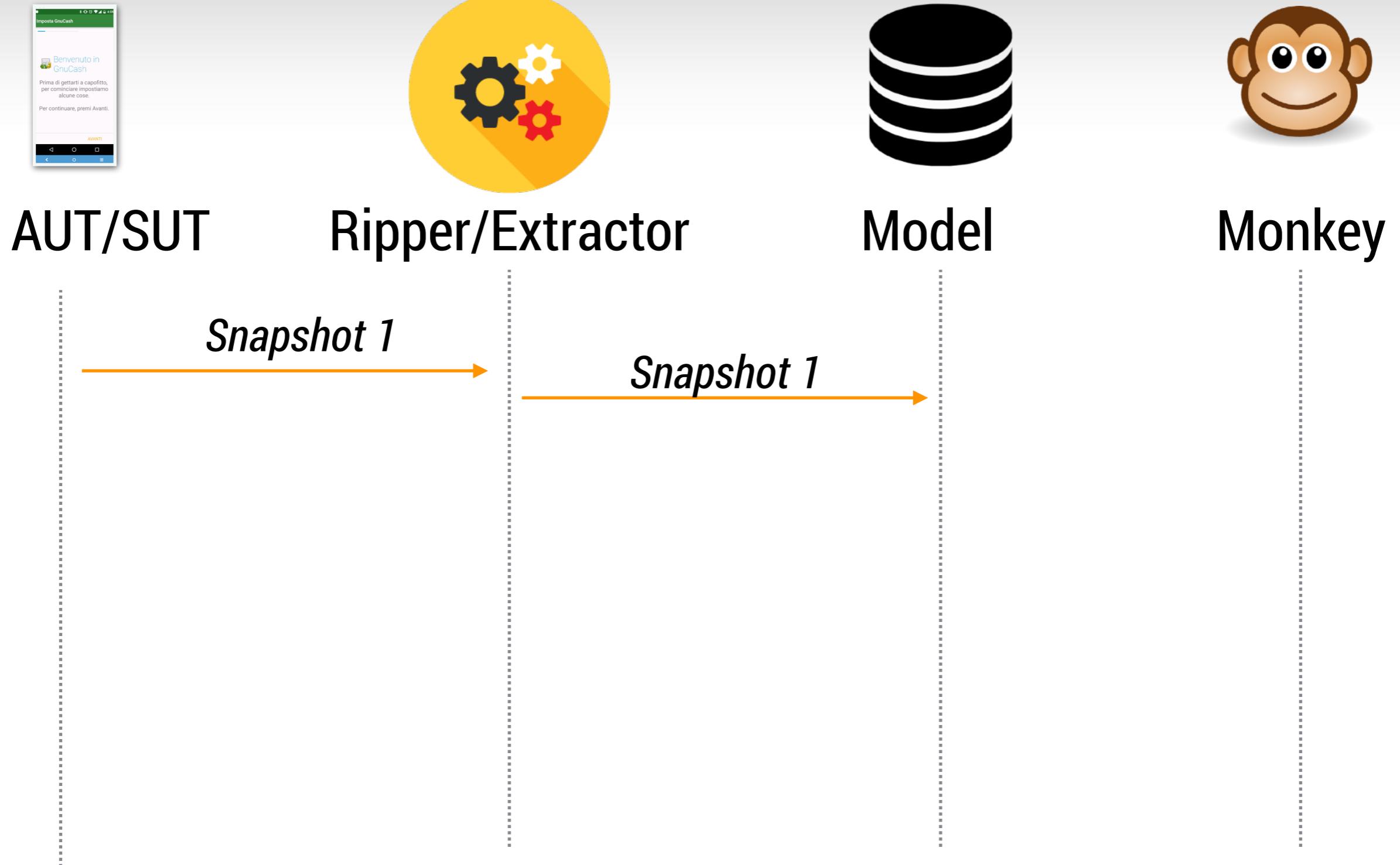


Monkey

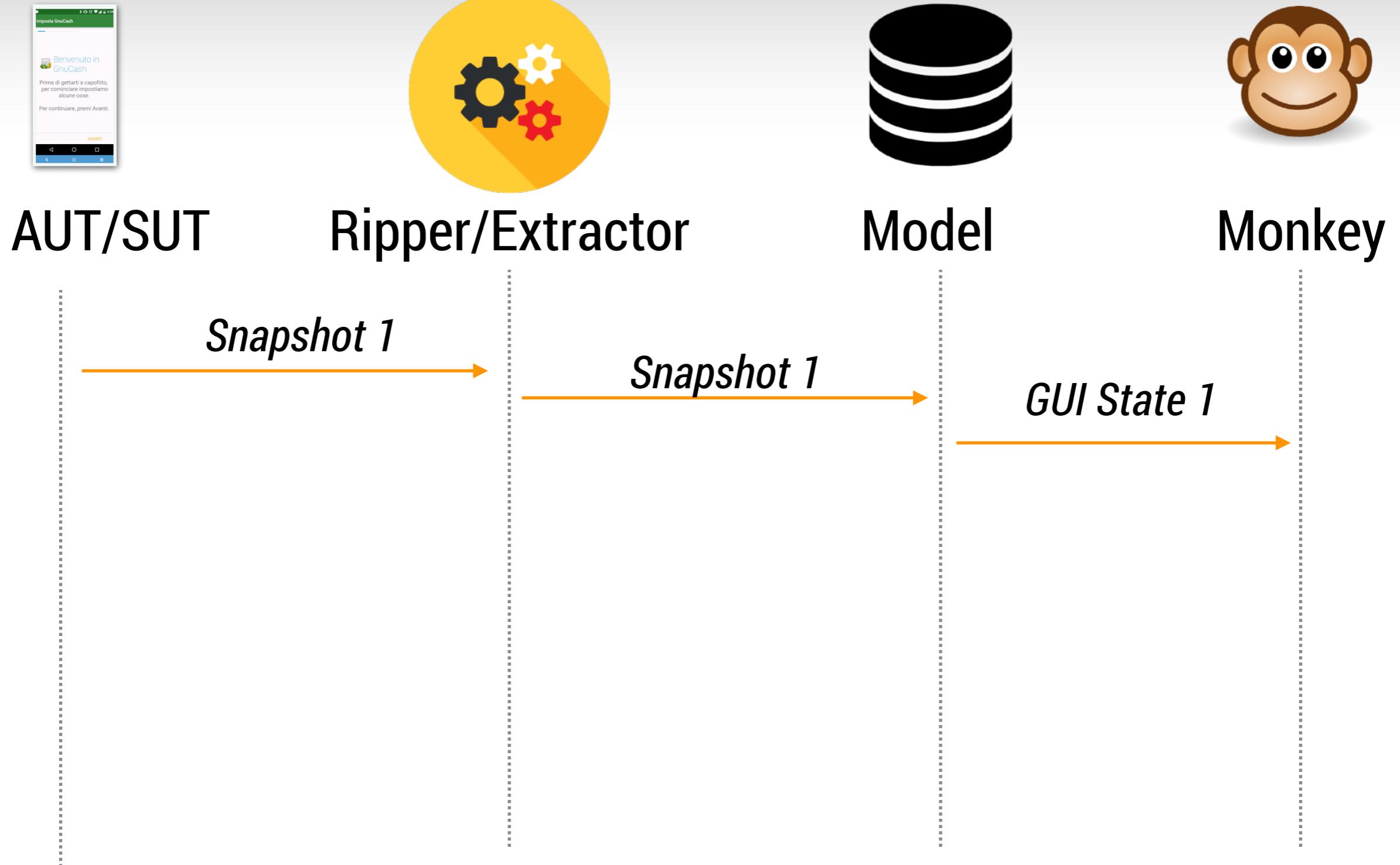
# Aside: GUI Ripping



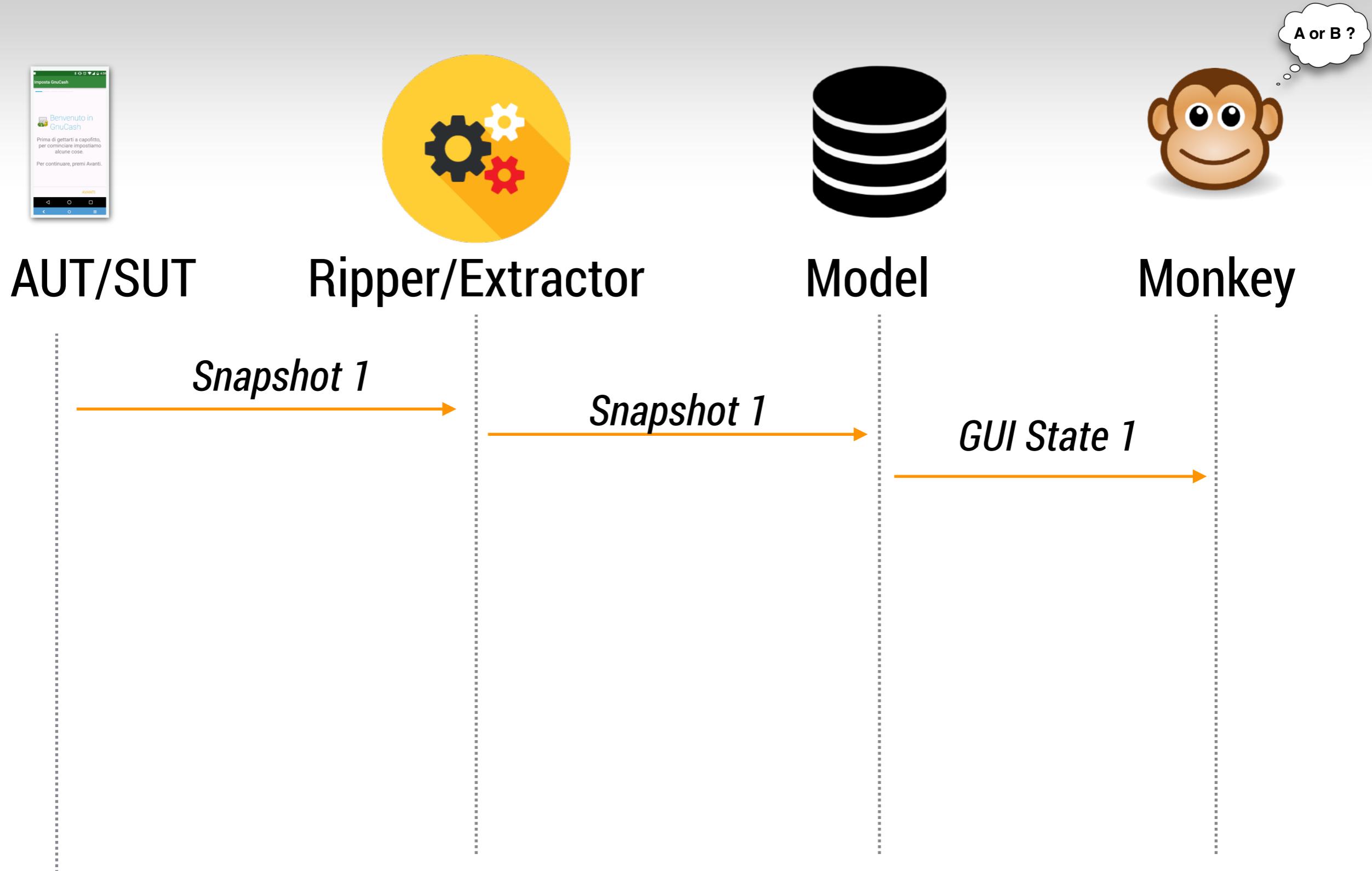
# Aside: GUI Ripping



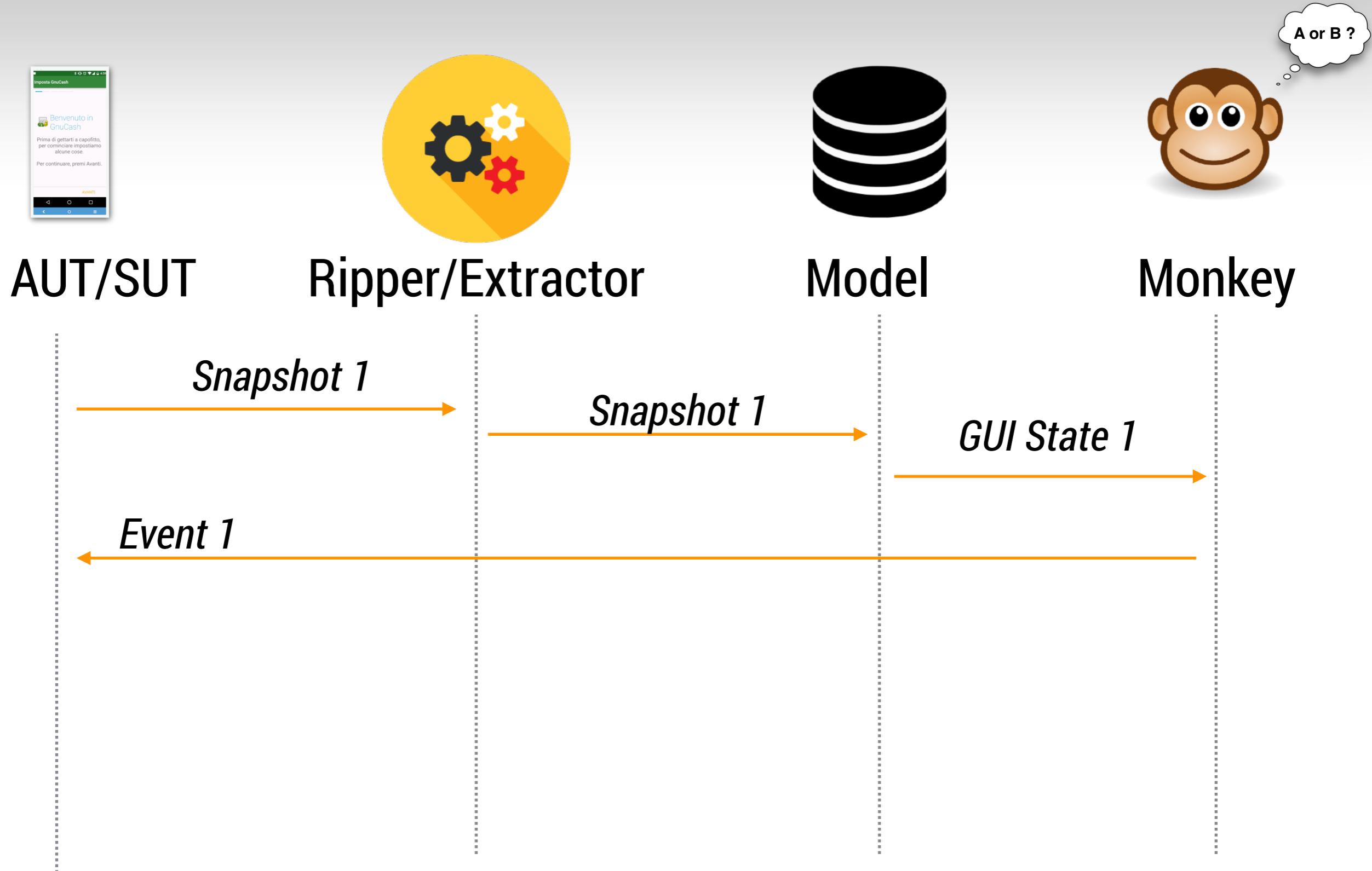
# Aside: GUI Ripping



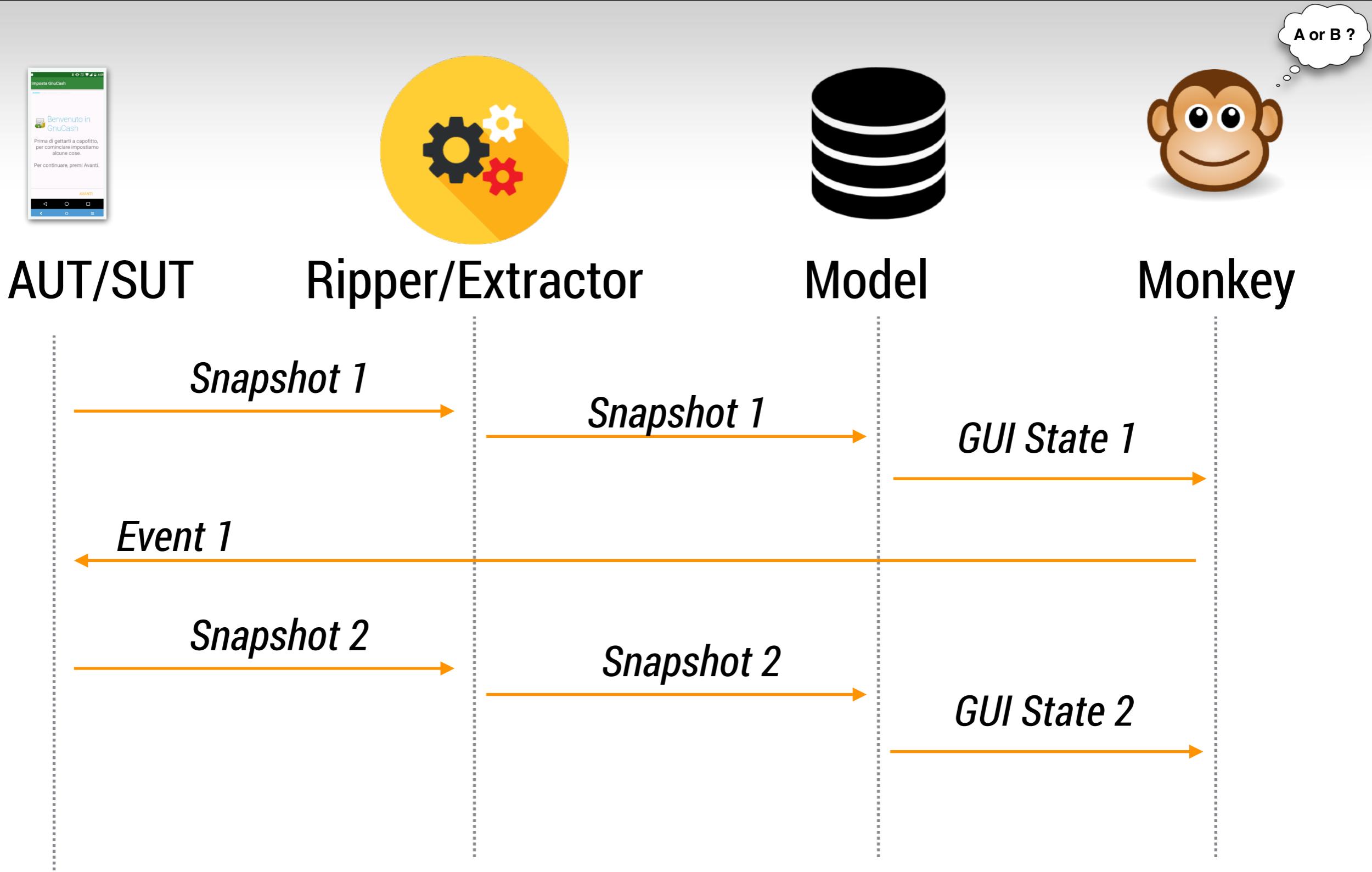
# Aside: GUI Ripping



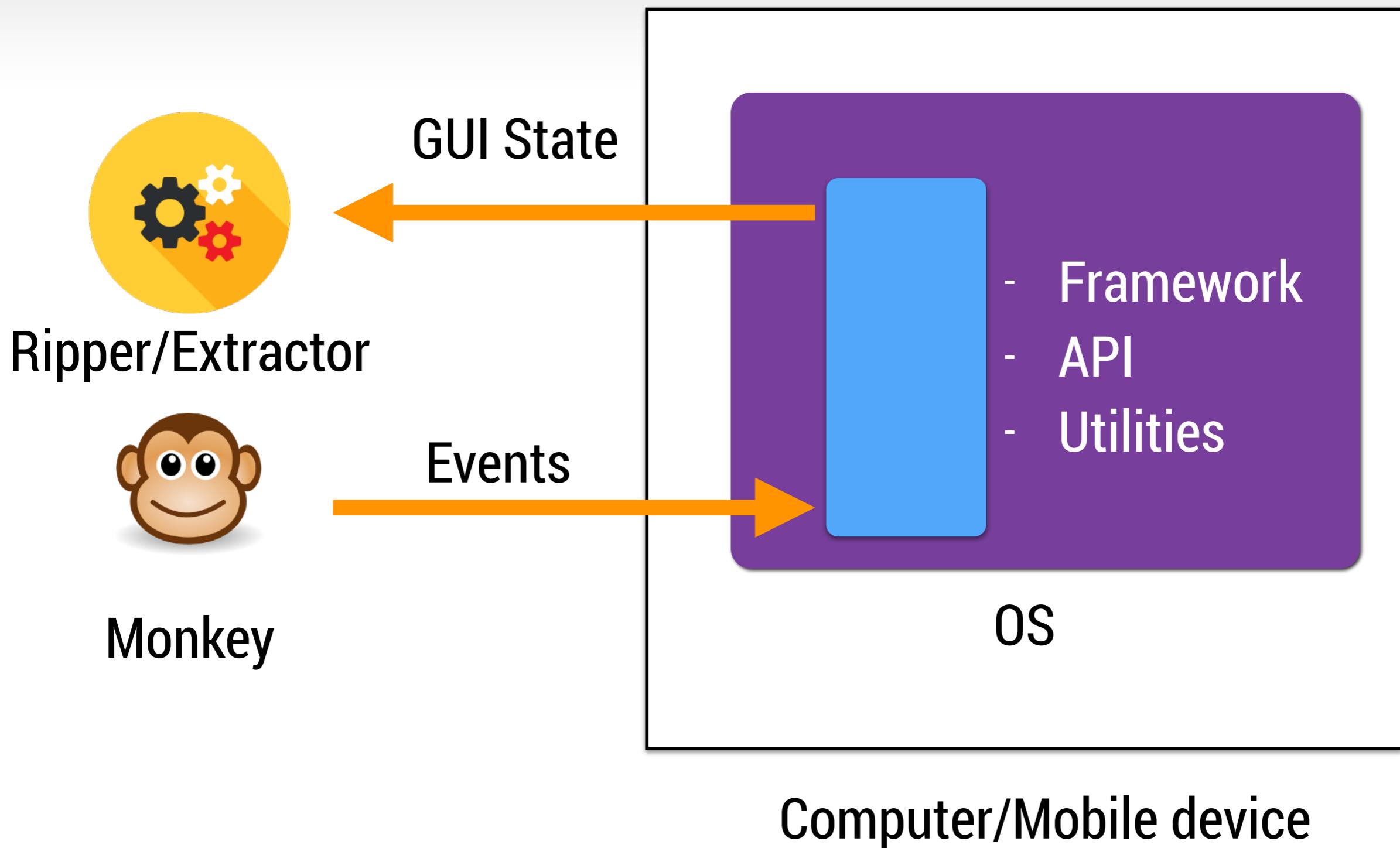
# Aside: GUI Ripping



# Aside: GUI Ripping



# GUI State extraction



# GUI State extraction

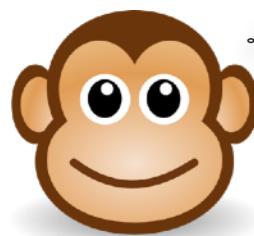
```
1  <?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
2  <hierarchy rotation="0">
3      <node index="0" text="" resource-id="" class="android.widget.FrameLayout" package="org.secuso.privacyfriendlydicer">
4          <node index="0" text="" resource-id="" class="android.widget.LinearLayout" package="org.secuso.privacyfriendlydicer">
5              <node index="0" text="" resource-id="" class="android.widget.FrameLayout" package="org.secuso.privacyfriendlydicer">
6                  <node index="0" text="" resource-id="org.secuso.privacyfriendlydicer:id/decor_content">
7                      <node index="0" text="" resource-id="org.secuso.privacyfriendlydicer:id/action_bar">
8                          <node index="0" text="" resource-id="org.secuso.privacyfriendlydicer:id/action_bar">
9                              <node index="0" text="" resource-id="org.secuso.privacyfriendlydicer:id/icon">
10                             <node index="1" text="Settings" resource-id="" class="android.widget.TextView" style="font-size: 14pt; font-weight: bold; color: #0000ff; background-color: #e0e0e0; border: 1px solid black; border-radius: 5px; padding: 5px; margin: 5px; text-align: center; width: fit-content; height: fit-content;">
11                         </node>
12                     </node>
13                 <node index="1" text="" resource-id="android:id/content" class="android.widget.FrameLayout" package="org.secuso.privacyfriendlydicer">
14                     <node index="0" text="" resource-id="" class="android.widget.LinearLayout" package="org.secuso.privacyfriendlydicer">
15                         <node index="0" text="" resource-id="android:id/list" class="android.widget.ListView" style="background-color: #f0f0f0; border: 1px solid black; border-radius: 5px; padding: 5px; margin: 5px; width: fit-content; height: fit-content;">
16                             <node index="0" text="Dicing Settings" resource-id="android:id/title" style="font-size: 12pt; font-weight: bold; color: #0000ff; margin-bottom: 5px; margin-left: 10px; margin-right: 10px;">
17                             <node index="1" text="" resource-id="" class="android.widget.LinearLayout" style="background-color: #f0f0f0; border: 1px solid black; border-radius: 5px; padding: 5px; margin: 5px; width: fit-content; height: fit-content;">
18                                 <node index="1" text="" resource-id="" class="android.widget.RelativeLayout" style="background-color: #f0f0f0; border: 1px solid black; border-radius: 5px; padding: 5px; margin: 5px; width: fit-content; height: fit-content;">
19                                     <node index="0" text="Roll Dice by Shaking" resource-id="android:id/text1" style="font-size: 10pt; color: #0000ff; margin-left: 10px; margin-right: 10px;">
20                                 </node>
21                                 <node index="2" text="" resource-id="android:id/widget_frame" class="android.widget.CheckBox" style="margin-left: 10px; margin-right: 10px;">
22                                     <node index="0" text="" resource-id="android:id/checkbox" class="android.widget.CompoundButton" style="margin-left: 10px; margin-right: 10px;">
23                                     </node>
24                                 </node>
25                             </node>
26                         </node>
27                     </node>
28                 </node>
29             </node>
30         </node>
31     </node>
32 </hierarchy>
```

# Systematic Exploration

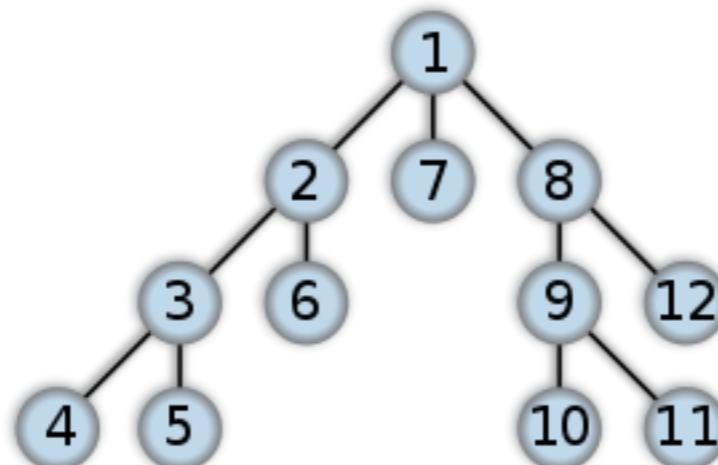


Monkey

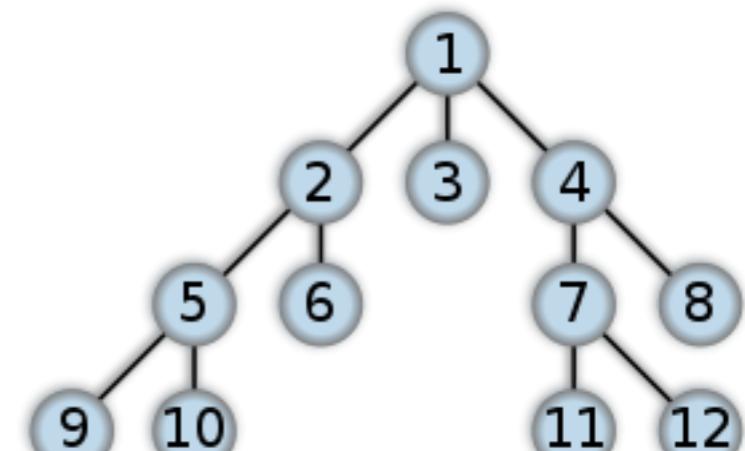
# Systematic Exploration



A or B ?



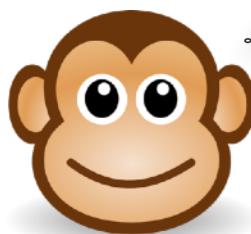
Depth-First (DF)



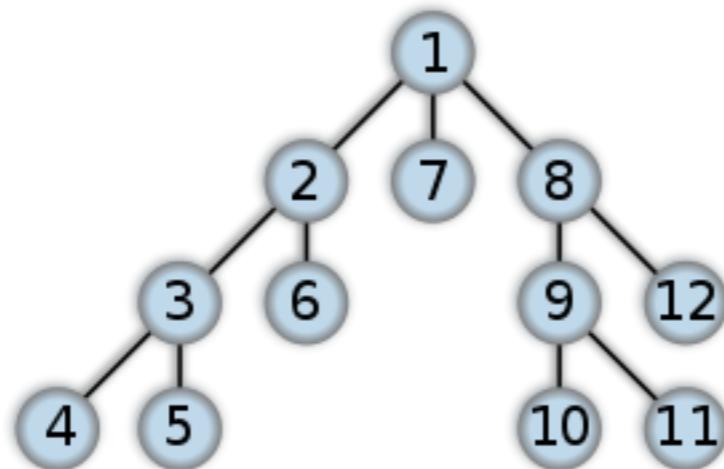
Breadth-First (BF)

Monkey

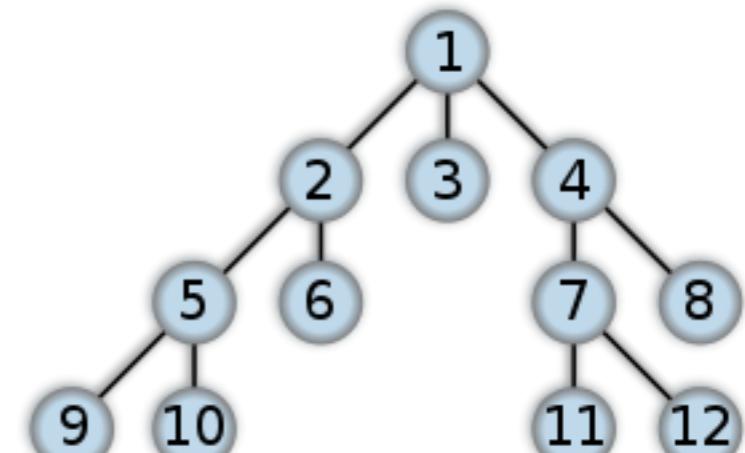
# Systematic Exploration



A or B ?



Depth-First (DF)



Breadth-First (BF)

Monkey

**Other options (online decision)**

Random (Uniform)

Random (A-priori distr.)

# Tools: Google Robo Test

The screenshot shows the Firebase Documentation website. At the top, there's a navigation bar with links for Products, Use Cases, Pricing, Docs (which is underlined), and Support. To the right of the navigation is a search bar with a magnifying glass icon and the word "Search". Further right are links for "GO TO CONSOLE", three vertical dots, and a user profile picture. Below the navigation is a blue header bar with tabs for OVERVIEW, GUIDES (which is underlined), REFERENCE, SAMPLES, and LIBRARIES. On the far right of the blue bar are links for "SEND FEEDBACK" and a gear icon.

**Documentation**

**OVERVIEW**   **GUIDES**   **REFERENCE**   **SAMPLES**   **LIBRARIES**   **SEND FEEDBACK**

**Firebase Test Lab for Android Robo Test** ★★★★★

Robo test is a test tool that is integrated with Firebase Test Lab for Android. Robo test analyzes the structure of your app's UI and then explores it methodically, automatically simulating user activities. Unlike the [UI/Application Exerciser Monkey](#) test, Robo test always simulates the same user activities in the same order when you use it to test an app on a specific device configuration with the same maximum depth and timeout settings. This lets you use Robo test to validate bug fixes and test for regressions in a way that isn't possible when testing with the [UI/Application Exerciser Monkey](#) test.

Robo test captures log files, saves a series of annotated screenshots, and then creates a video from those screenshots to show you the simulated user operations that it performed. These logs, screenshots, and videos can help you to determine the root cause if your app crashes, and can also help you to find issues with your app's UI.

**Important:** Robo test is not the same as (or based on) the Robotium or Robolectric test frameworks.

**Contents**

- Configuring Robo test
- Integration with Google Play
- Test account sign-in and predefined text input
  - Sign-in
  - Predefined text input
- Known issues

**Android**

- Overview
- Test with the Firebase Console
- Test with the gcloud CLI
- Test with CI Systems
- Test Lab and Android Studio
- Analyze Test Results
- Robo Test**
- Test Screenshots
- Test with Virtual Devices
- Game Test Loop
- Beyond Pre-Launch Reports

**Performance Monitoring**

**Crash Reporting**

<https://firebase.google.com/docs/test-lab/robo-ux-test>

# Pros and Cons

## Automation Frameworks



- ✓ Easy reproduction
- ✓ High level syntax
- ✓ Black box testing



- Learning curve
- User-defined oracles
- Expensive maintenance

## Record & Replay

- ✓ Easy reproduction

- Expensive collection and maintenance
- Coupled to locations

## AIG: Random Based

- ✓ Fast execution
- ✓ Good at finding crashes

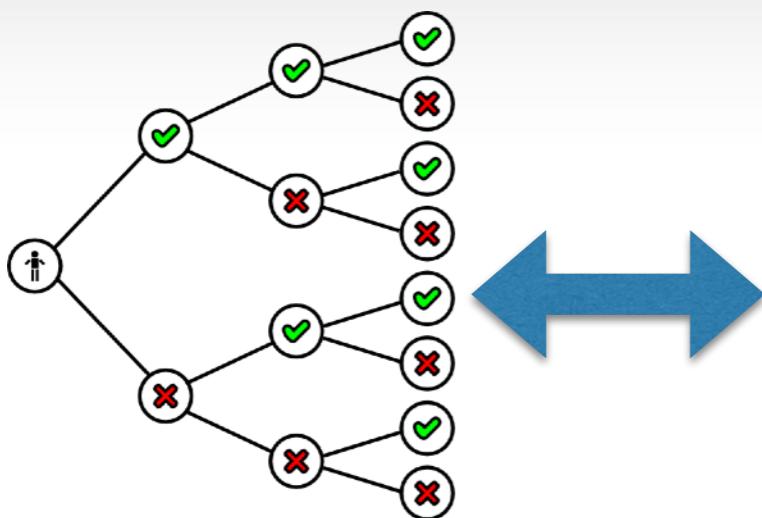
- Invalid events
- Lack of expressiveness

## AIG: Systematic

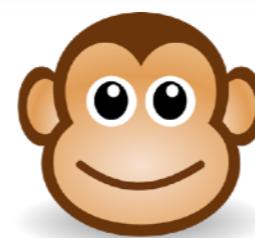
- ✓ Achieves Reasonable Coverage
- ✓ May miss crashes

- Can be time consuming
- Typically cannot exercise complex features

# Model-Based Testing (MBT)



Model

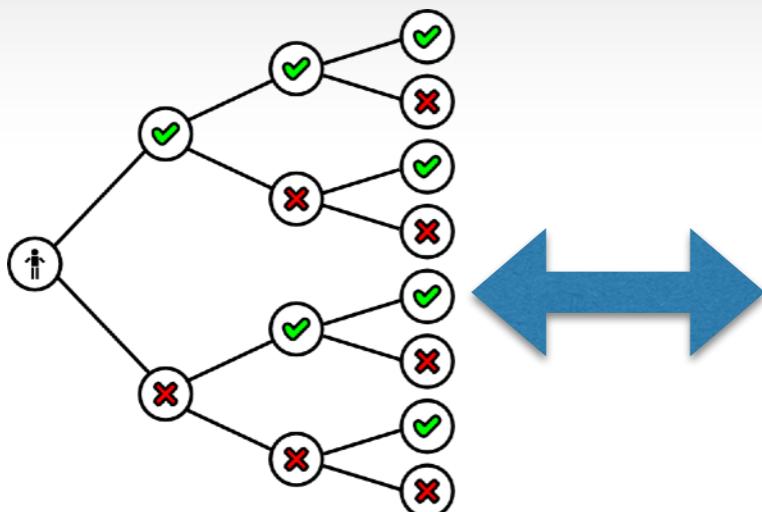


UI Events

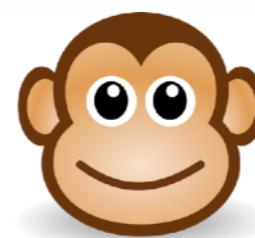


AUT/SUT

# Model-Based Testing (MBT)



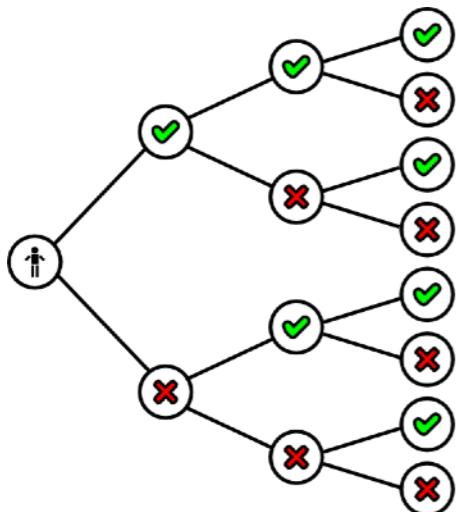
Model



UI Events



AUT/SUT



- Manually generated
- Automatically generated (source code)
- Ripped at runtime (upfront)
- Ripped at runtime (interactive)

# Pros and Cons

## Automation Frameworks



- ✓ Easy reproduction
- ✓ High level syntax
- ✓ Black box testing



- Learning curve
- User-defined oracles
- Expensive maintenance

## Record & Replay

- ✓ Easy reproduction

- Expensive collection and maintenance
- Coupled to locations

## AIG: Random Based

- ✓ Fast execution
- ✓ Good at finding crashes

- Invalid events
- Lack of expressiveness

## AIG: Systematic

- ✓ Achieves Reasonable Coverage
- ✓ May miss crashes

- Can be time consuming
- Typically cannot exercise complex features

## AIG: Model Based

- ✓ Event sequences
- ✓ Automatic exploration

- Some Invalid sequences
- State Explosion
- Incomplete models

# Other Types of AIG Approaches

- **Recently New Approaches have been introduced for AIG:**
  - Search-Based Approaches<sup>1</sup>
  - Symbolic/Concolic Execution<sup>2</sup>

<sup>1</sup>Ke Mao, Mark Harman, and Yue Jia. 2016. *Sapienz: multi-objective automated testing for Android applications*. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*

<sup>2</sup>Nariman Mirzaei, Joshua Garcia, Hamid Bagheri, Alireza Sadeghi, and Sam Malek. 2016. *Reducing combinatorics in GUI testing of android applications*. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*

# Extending Research to XR/AR/VR apps



# Extending Research to XR/AR/VR apps

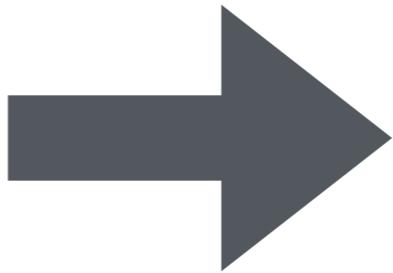
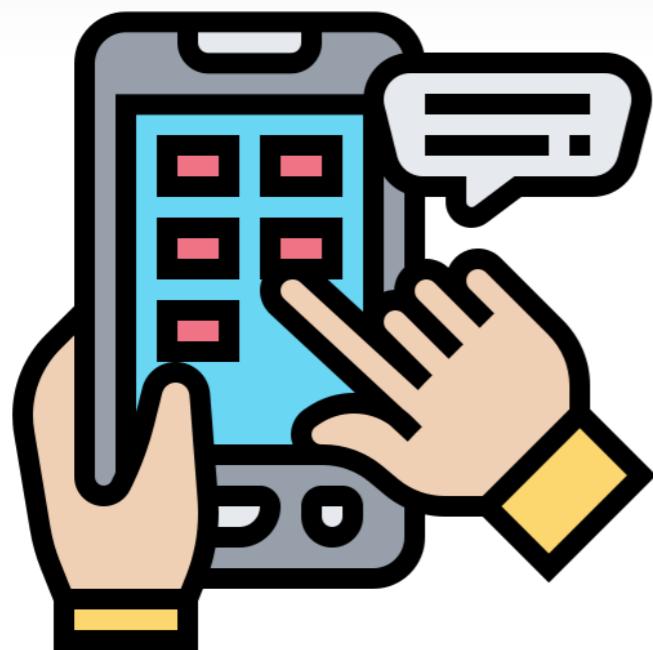


# Extending Research to XR/AR/VR apps

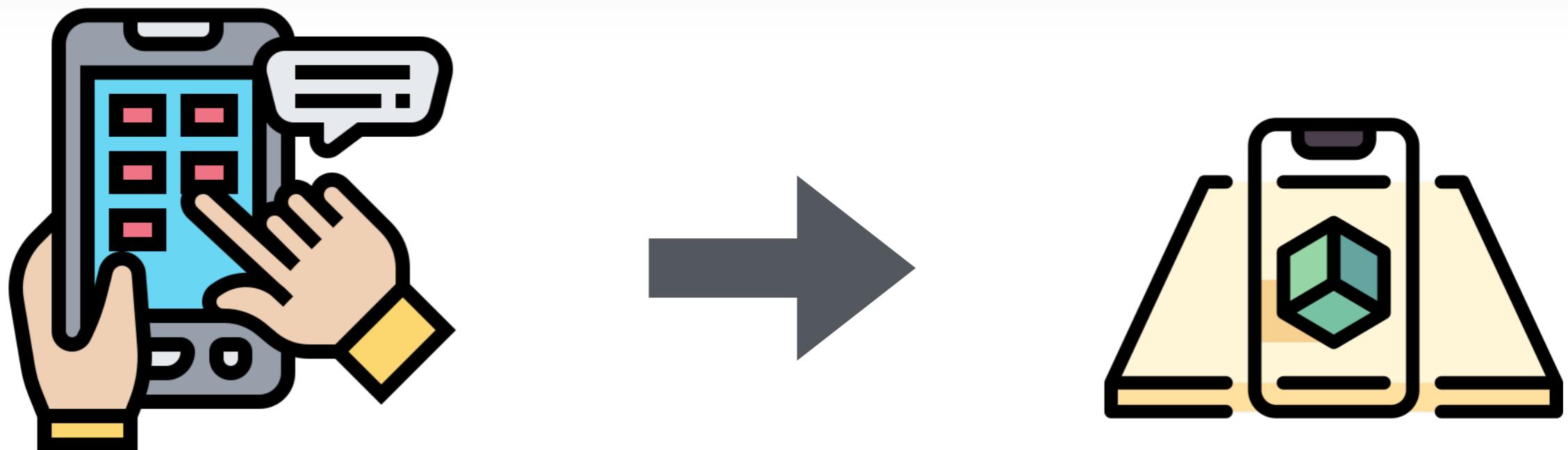


**Main Challenge 1:** Interfacing with and fetching GUI information

# Extending Research to XR/AR/VR apps

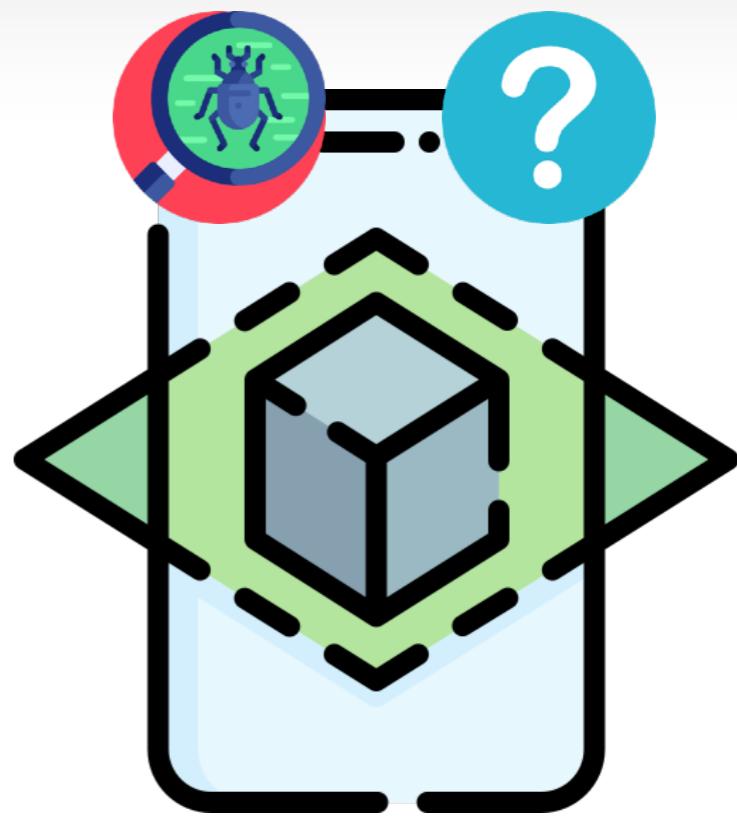
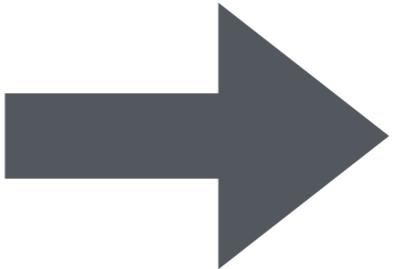
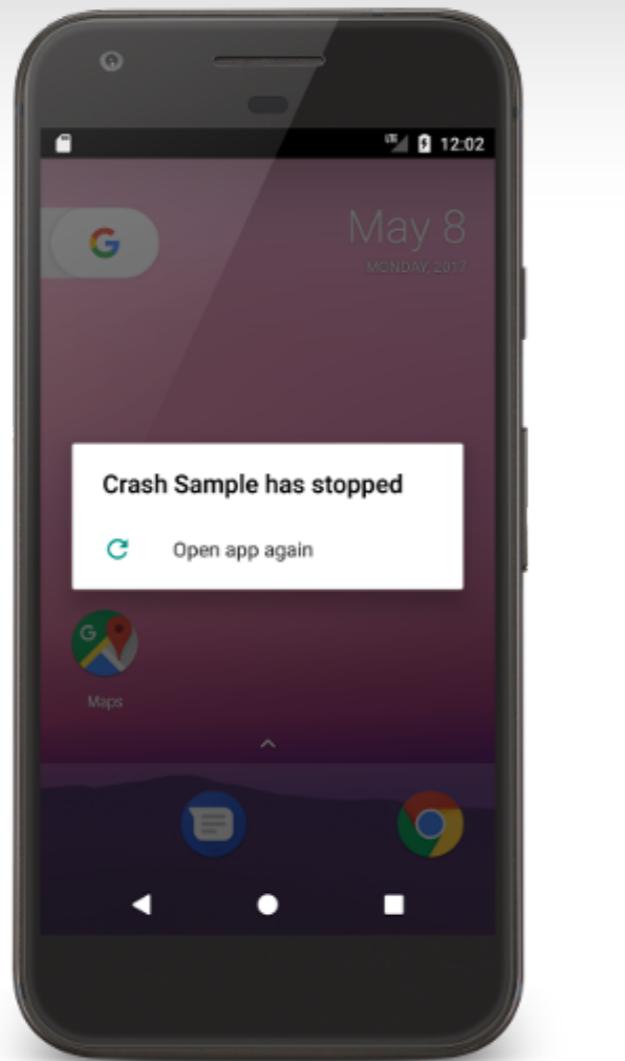


# Extending Research to XR/AR/VR apps

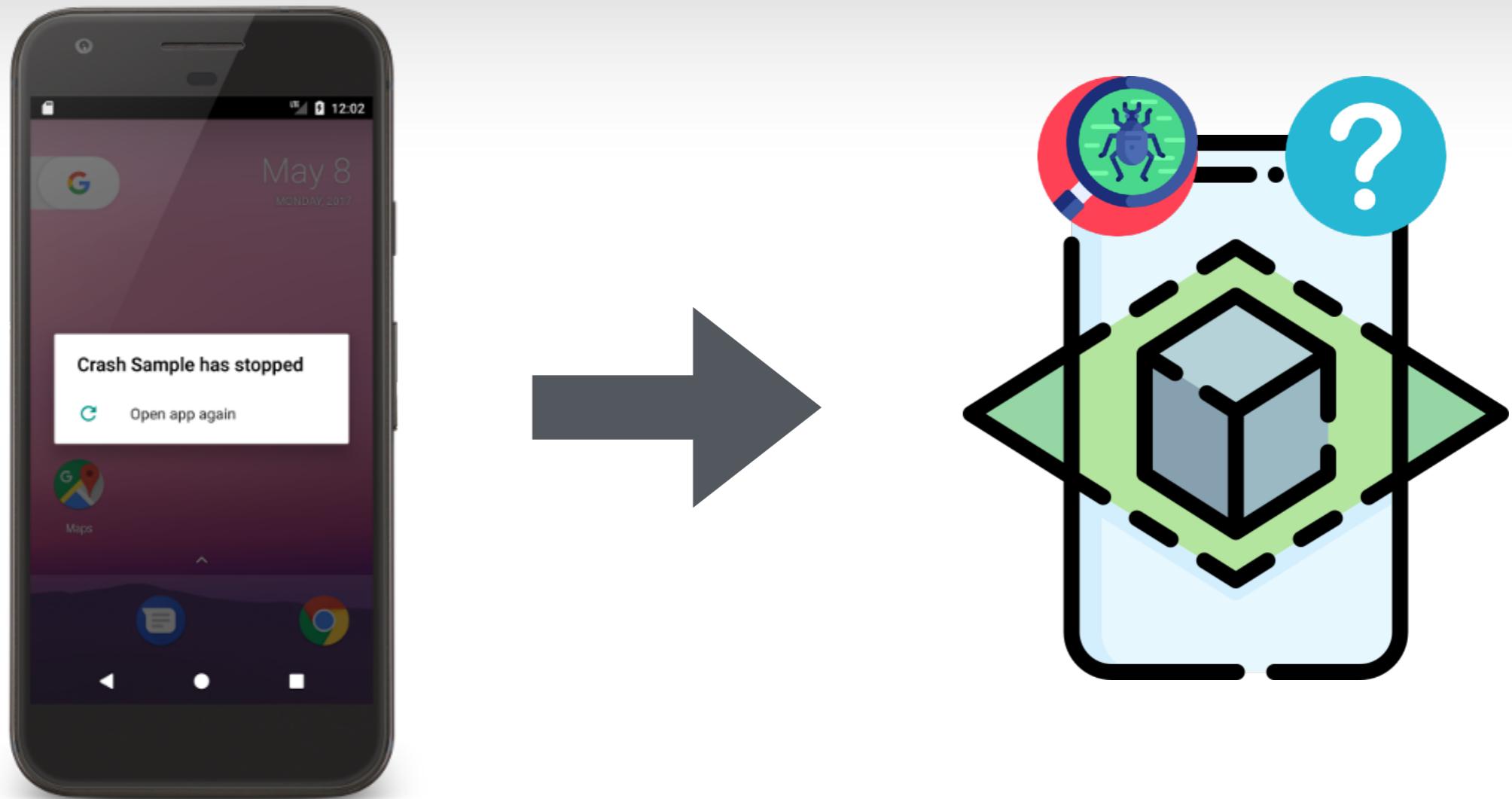


**Main Challenge 2: Generating meaningful inputs**

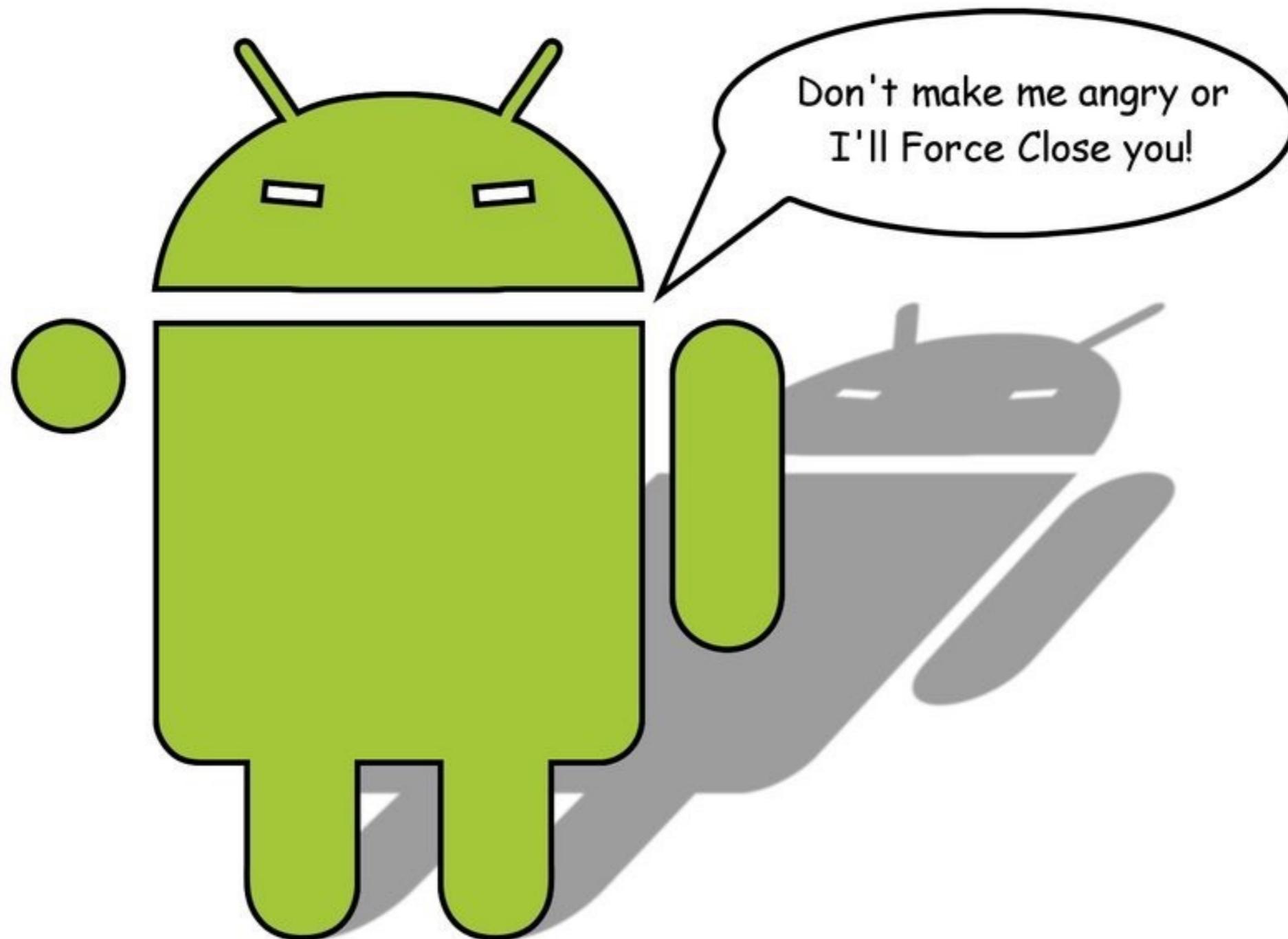
# Extending Research to XR/AR/VR apps



# Extending Research to XR/AR/VR apps



**Main Challenge 3: Understanding & Detecting Failures**



# Automatically Discovering, Reporting and Reproducing Android Application Crashes with CrashScope

# CrashScope Publication

## Automatically Discovering, Reporting and Reproducing Android Application Crashes

Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, Christopher Vendome, and Denys Poshyvanyk  
College of William & Mary  
`{kpmoran, mlinarev, cebernal, cvendome, denys}@cs.wm.edu`

**Abstract**—Mobile developers face unique challenges when detecting and reporting crashes in apps due to their prevailing GUI event-driven nature and additional sources of inputs (e.g., sensor readings). To support developers in these tasks, we introduce a novel, automated approach called CRASHSCOPE. This tool explores a given Android app using systematic input generation, according to several strategies informed by static and dynamic analyses, with the intrinsic goal of triggering crashes. When a crash is detected, CRASHSCOPE generates an augmented crash report containing screenshots, detailed crash reproduction steps, the captured exception stack trace, and a fully replayable script that automatically reproduces the crash on a target device(s).

We evaluated CRASHSCOPE’s effectiveness in discovering crashes as compared to five state-of-the-art Android input generation tools on 61 applications. The results demonstrate that CRASHSCOPE performs about as well as current tools for detecting crashes and provides more detailed fault information. Additionally, in a study analyzing eight real-world Android app crashes, we found that CRASHSCOPE’s reports are easily readable and allow for reliable reproduction of crashes by presenting more explicit information than human written reports.

### I. INTRODUCTION

Continued growth in the mobile hardware and application marketplace is being driven by a landscape where users tend to prefer mobile smart devices and apps for tasks over their desktop counterparts. The gesture-driven nature of mobile apps has given rise to new challenges encountered by programmers during development and maintenance, specifically with regard to testing and debugging [41]. One of the most difficult [22], [24] and important maintenance tasks is the creation and resolution of bug reports [35]. Reports concerning application crashes are of particular importance to developers, because crashes represent a jarring software fault that is directly user facing and immediately impacts an app’s utility and success. If an app is not behaving as expected due to crashes, missing features, or other bugs, nearly half of users are likely to abandon the app for a competitor [12] in a marketplace such as Google Play [10].

Mobile developers heavily rely on user reviews [42], [49], [65], crash reports from the field in the form of stack traces, or reports in open source issue tracking systems to detect bugs in their apps. In each of these cases, the bug/crash reports are typically lacking in information [27], [41], containing only a stack trace, overly detailed logs or loosely structured natural language (NL) information regarding the crash [23]. This is not surprising as previous studies showed that information, which is most useful for a developer resolving a bug report (e.g., reproduction steps, stack traces and test cases), is often

the most difficult information for reporters to provide [33]. Furthermore, the absence of this information is a major cause of developers failing to reproduce bug/crash reports [22]. In addition to the quality of the reports, some other factors specific to Android apps such as hardware and software fragmentation [3], API instability and fault-proneness [21], [48], the event-driven nature of Android apps, gesture-based interaction, sensor interfaces, and the possibility of multiple contextual states (e.g., wifi/GPS on/off) make the process of detecting, reporting, and reproducing crashes challenging.

Motivated by these current issues developers face regarding mobile application crashes, we designed and implemented CRASHSCOPE, a practical system that automatically discovers, reports, and reproduces crashes for Android applications. CRASHSCOPE explores a given app using a systematic input generation algorithm and produces expressive crash reports with explicit steps for reproduction in an easily readable natural language format. This approach requires only an .apk file and an Android emulator or device to operate and requires no instrumentation of the subject apps or the Android OS. The entirety of the CRASHSCOPE workflow is completely automated, requiring no developer intervention, other than reading produced reports. Our systematic execution includes different exploration strategies, aimed at eliciting crashes from Android apps, which include automatic text generation capabilities based on the context of allowable characters for text entry fields, and targeted testing of contextual features, such as the orientation of the device, wireless interfaces, and sensors. We specifically tailored these features to test the common causes of app crashes as identified by previous studies [26], [45], [79]. During execution, CRASHSCOPE captures detailed information about the subject app, such as the inputs sent to the device, screenshots and GUI information, exceptions, and crash information. This information is then translated into detailed crash reports and replayable scripts, for any encountered crash.

This paper makes the following noteworthy contributions:

- 1) We design and implement a practical and automatic approach for discovering, reporting, and reproducing Android application crashes, called CRASHSCOPE. To the best of the author’s knowledge, this is the first approach that is able to generate expressive, detailed crash reports for mobile apps, including screenshots and augmented NL reproduction steps, in a completely automatic fashion. CRASHSCOPE is also one of the only available fully-automated Android testing approaches







# Categories of automated testing approaches for Mobile apps

- Model-based input generation
- Random-based input generation
- Record and replay
- Others (Manual Testing Frameworks)

# The Current State of Automated Mobile Testing

Tool Name	Instr.	GUI Exploration	Types of Events	Crash Resilient	Replayable Test Cases	NL Crash Reports	Emulators, Devices
Dynodroid	Yes	Guided/Random	System, GUI, Text	Yes	No	No	No
EvoDroid	No	System/Evo	GUI	No	No	No	N/A
AndroidRipper	Yes	Systematic	GUI, Text	No	No	No	N/A
MobiGUITar	Yes	Model-Based	GUI, Text	No	Yes	No	N/A
A3E DFS	Yes	Systematic	GUI	No	No	No	Yes
A3E Targeted [20]	Yes	Model-Based	GUI	No	No	No	Yes
Swifthand	Yes	Model-Based	GUI, Text	N/A	No	No	Yes
PUMA	Yes	Programmable	System, GUI, Text	N/A	No	No	Yes
ACTEve	Yes	Systematic	GUI	N/A	No	No	Yes
VANARSena	Yes	Random	System, GUI, Text	Yes	Yes	No	N/A
Thor	Yes	Test Cases	Test Case Events	N/A	N/A	No	No
QUANTUM	Yes	Model-Based	System, GUI	N/A	Yes	No	N/A
AppDoctor	Yes	Multiple	System, GUI, Text	Yes	Yes	No	N/A
ORBIT	No	Model-Based	GUI	N/A	No	No	N/A
SPAG-C	No	Record/Replay	GUI	N/A	N/A	No	No
JPF-Android	No	Scripting	GUI	N/A	Yes	No	N/A
MonkeyLab	No	Model-based	GUI, Text	No	Yes	No	Yes
CrashDroid	No	Manual Rec/Replay	GUI, Text	Manual	Yes	Yes	Yes
SIG-Droid	No	Symbolic	GUI, Text	N/A	Yes	No	N/A
CrashScope	No	Systematic	GUI, Text, System	Yes	Yes	Yes	Yes

# The Current State of Automated Mobile Testing

Tool Name	Instr.	GUI Exploration	Types of Events	Crash Resilient	Replayable Test	NL Crash Reports	Emulators, Devices
Droid	Yes	No	GUI, Text	Yes	Yes	No	N/A
AFT	Yes	No	GUI, Text	Yes	Yes	No	N/A
A3	Yes	No	GUI, Text	Yes	Yes	No	N/A
CrashScope	No	Systematic	GUI, Text, System	Yes	Yes	Yes	Yes

What are the limitations of current automated approaches?

# Limitations of Automated Mobile Testing and Debugging

- Lack of detailed, easy to understand testing results for faults/crashes<sup>1</sup>
- No easy way to reproduce test scenarios<sup>1</sup>
- Not practical from a developers viewpoint
- Few approaches enable different strategies capable of generating text and testing contextual features

---

<sup>1</sup>S. R. Choudhary, A. Gorla, and A. Orso. Automated Test Input Generation for Android: Are we there yet? In 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2015

# Past Studies of Mobile Bugs and Crashes

- Many crashes can be mapped to *well-defined, externally inducible faults*<sup>1</sup>
- *Contextual features*, such as network connectivity and screen rotation, account for many of these externally inducible faults<sup>12</sup>
- These dominant root causes can affect *many different* user execution paths<sup>1</sup>

---

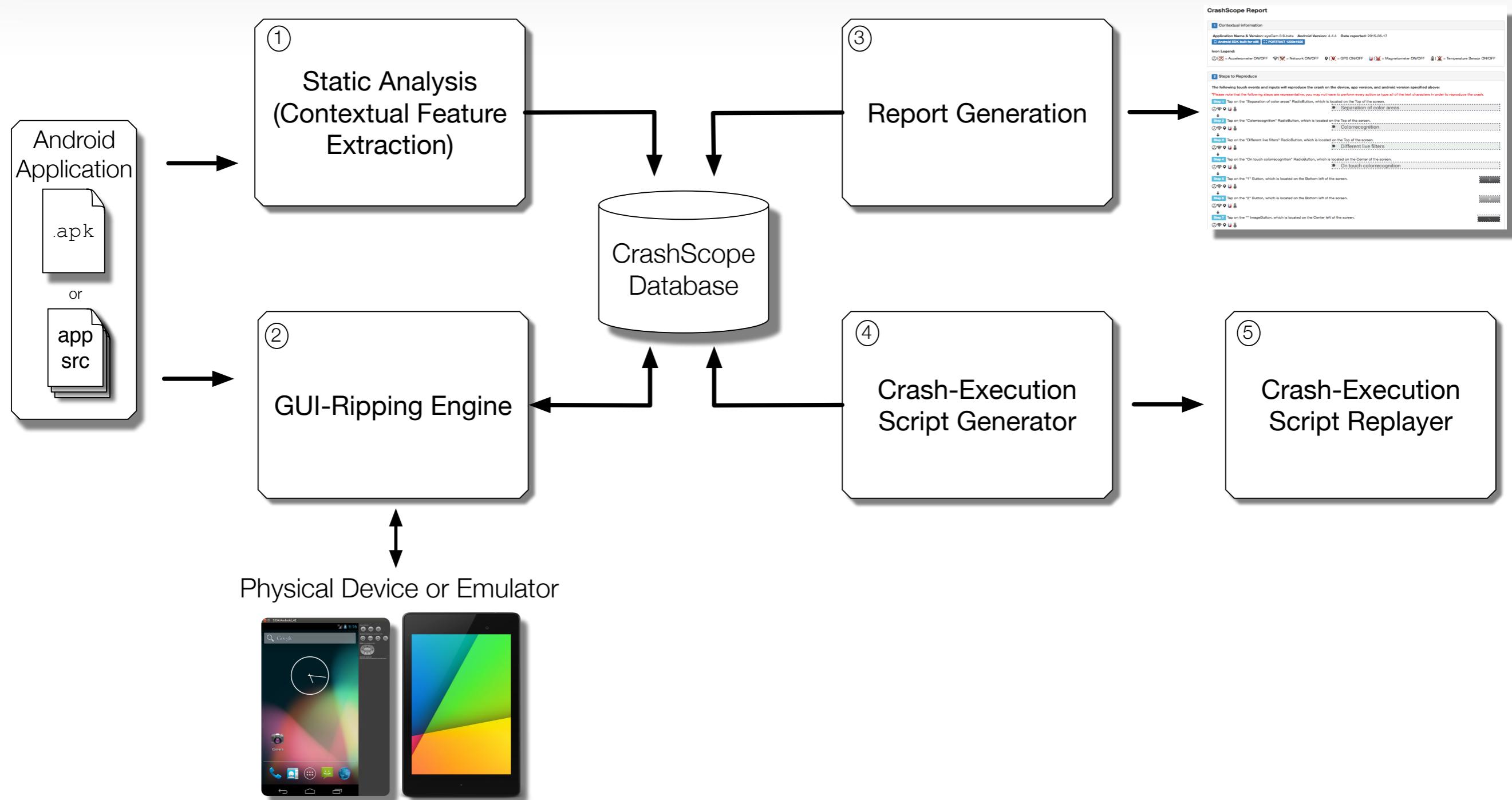
<sup>1</sup>L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan. Automatic and scalable fault detection for mobile applications. MobiSys '14

<sup>2</sup>R. N. Zaeem, M. R. Prasad, and S. Khurshid. Automated generation of oracles for testing user-interaction features of mobile apps, ICST '14

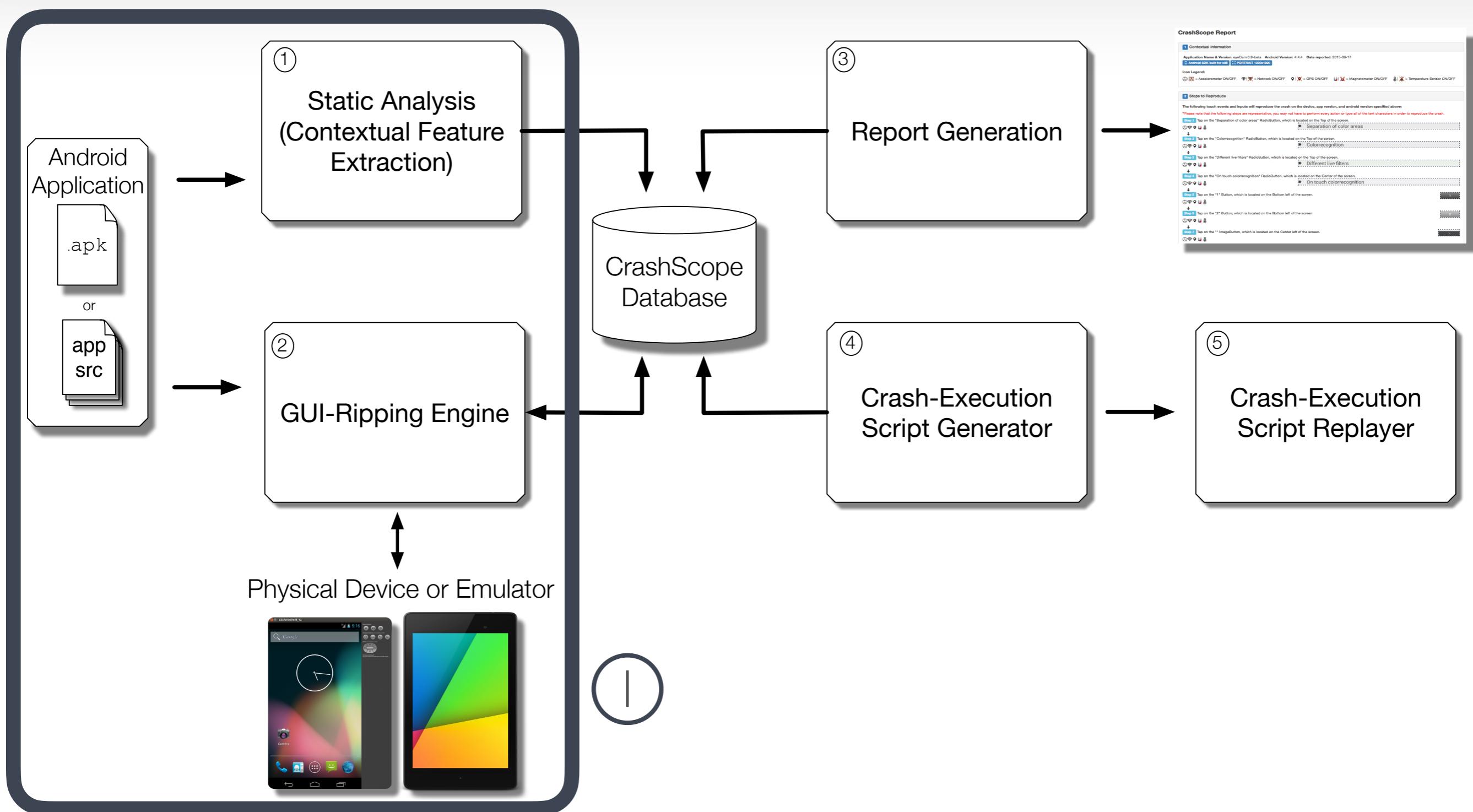
# Our Solution: CRASHSCOPE

- Completely automated approach
- Generates detailed, expressive bug reports and replayable scripts
- A practical tool, requiring no instrumentation framework, or modification to the OS or applications
- Capable of running on both physical devices and emulators
- Differing execution strategies able to test contextual features

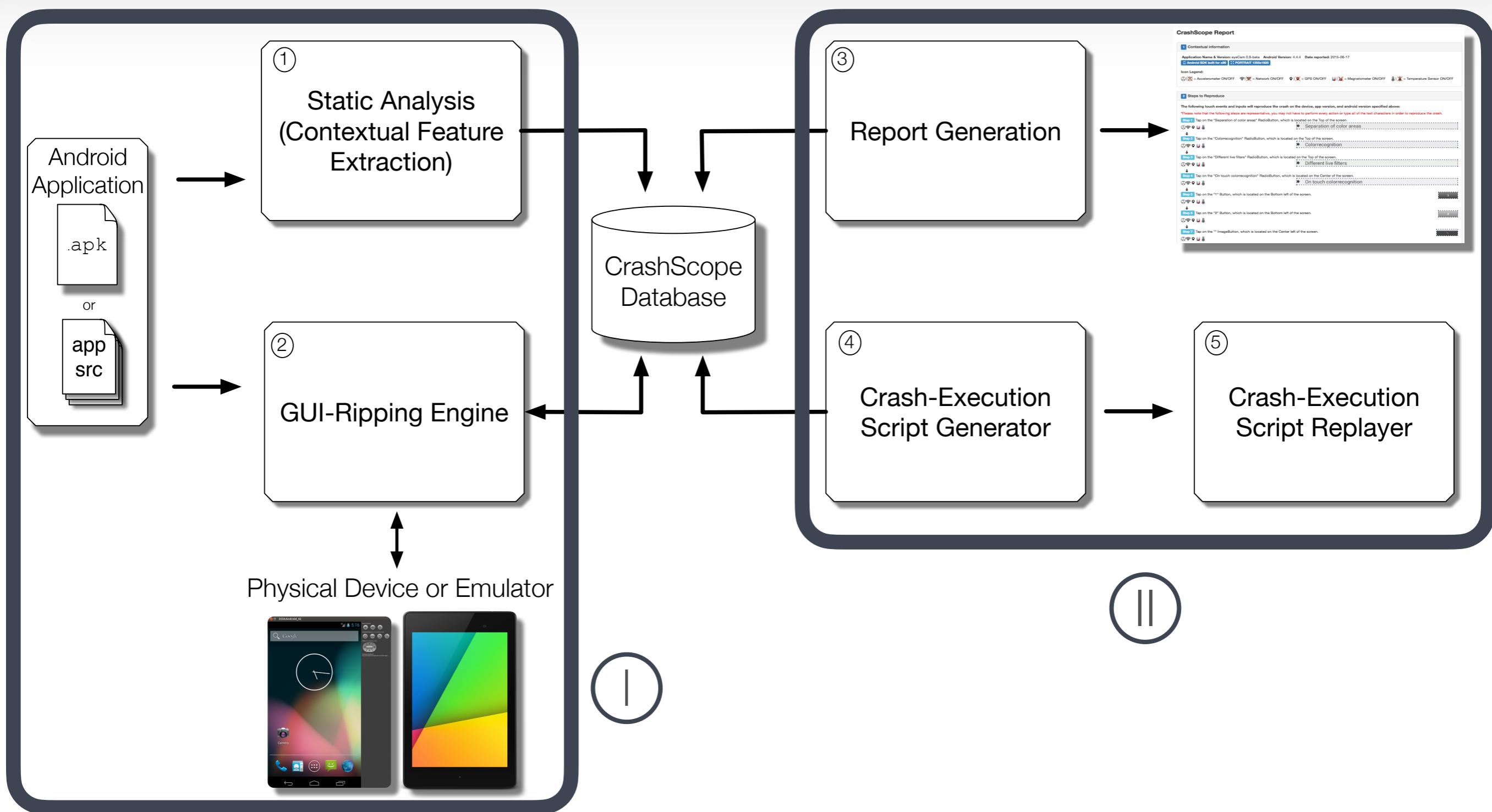
# CRASHSCOPE: Design



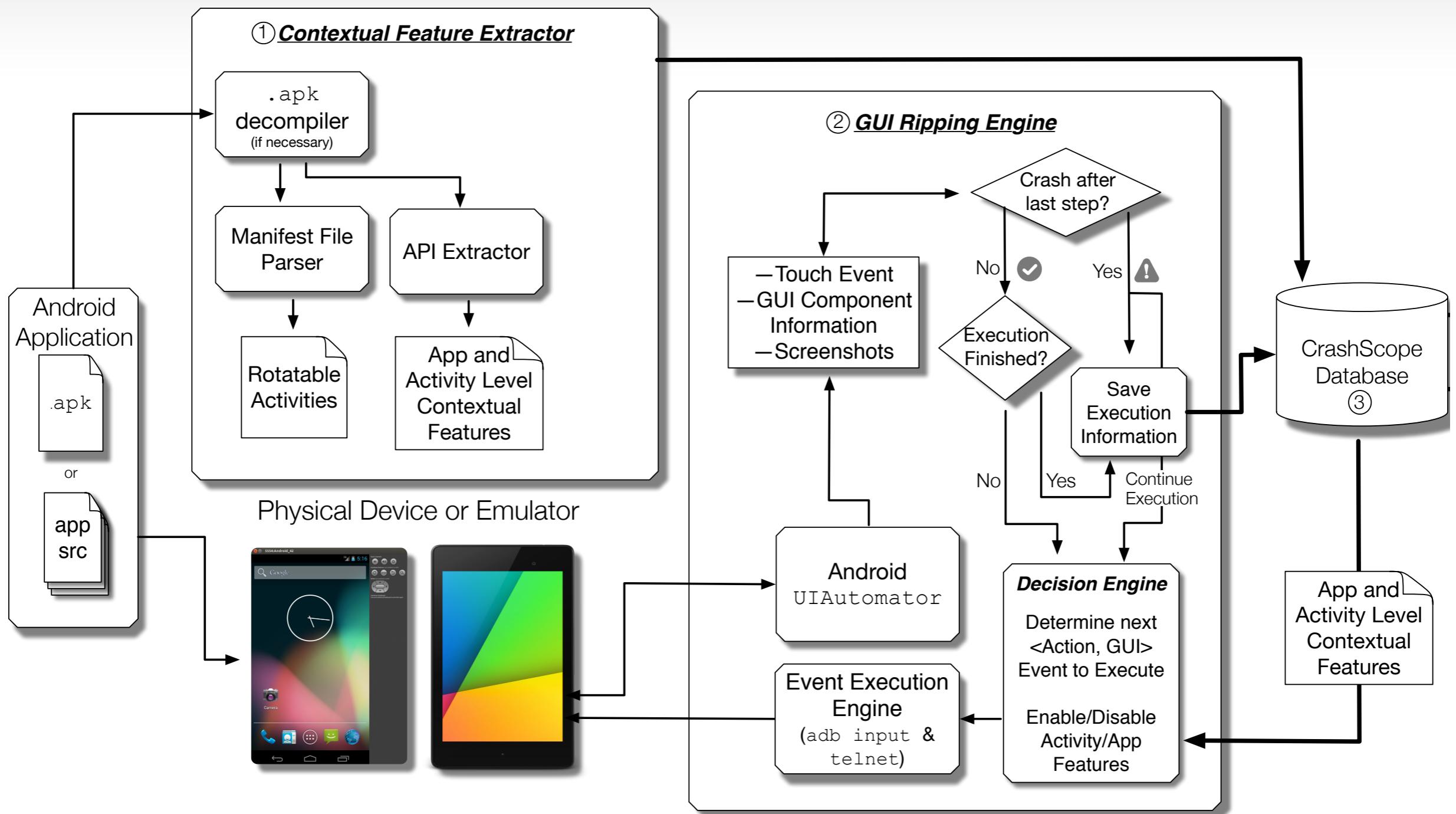
# CRASHSCOPE: Design



# CRASHSCOPE: Design



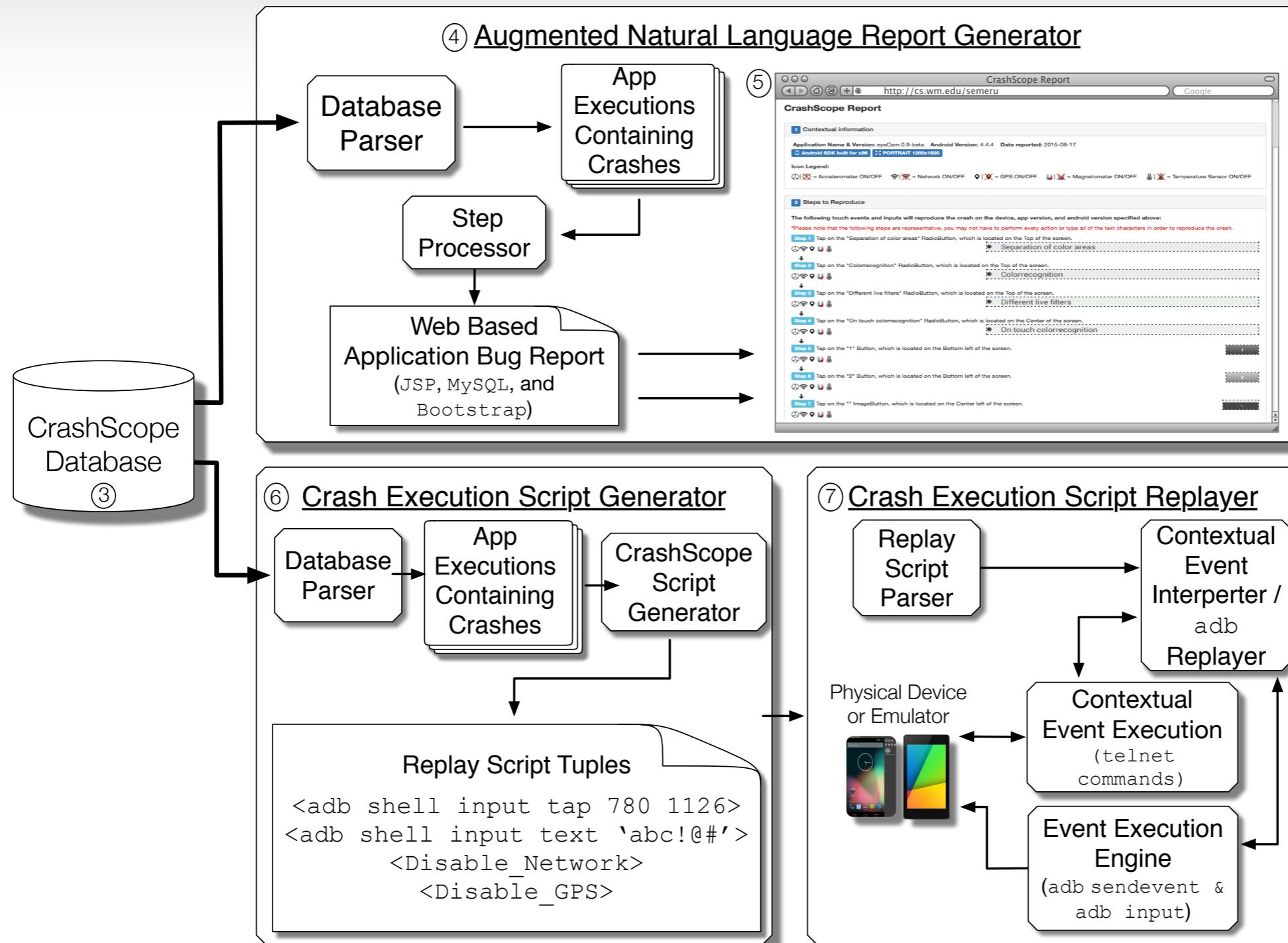
# CRASHSCOPE: Analysis



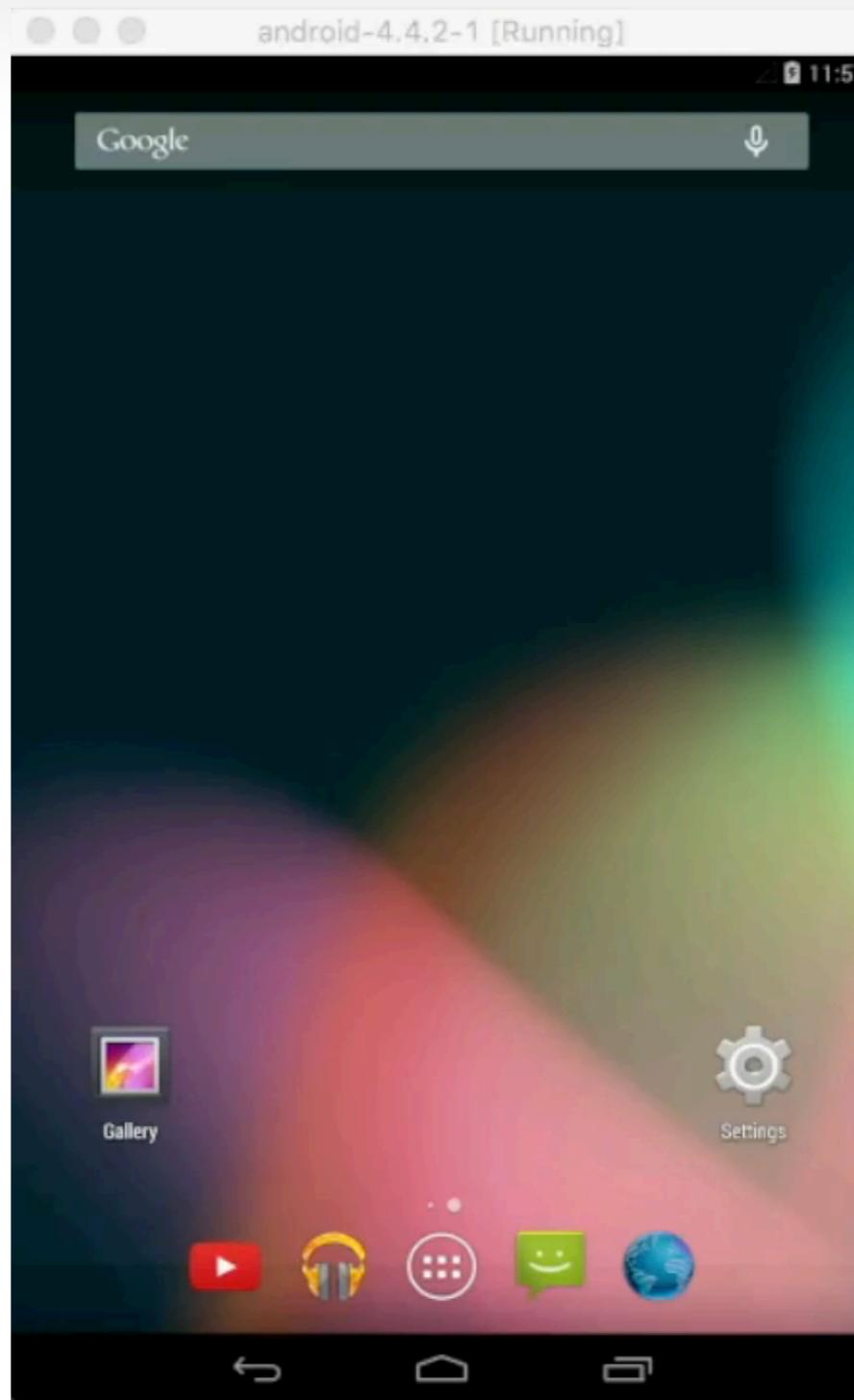
# CRASHSCOPE: Exploration

- *GUI-Traversal:* Top-Down & Bottom Up
- *Text Entry:* Expected, Unexpected, No Text
- *Contextual Features:* Enabled or Disabled

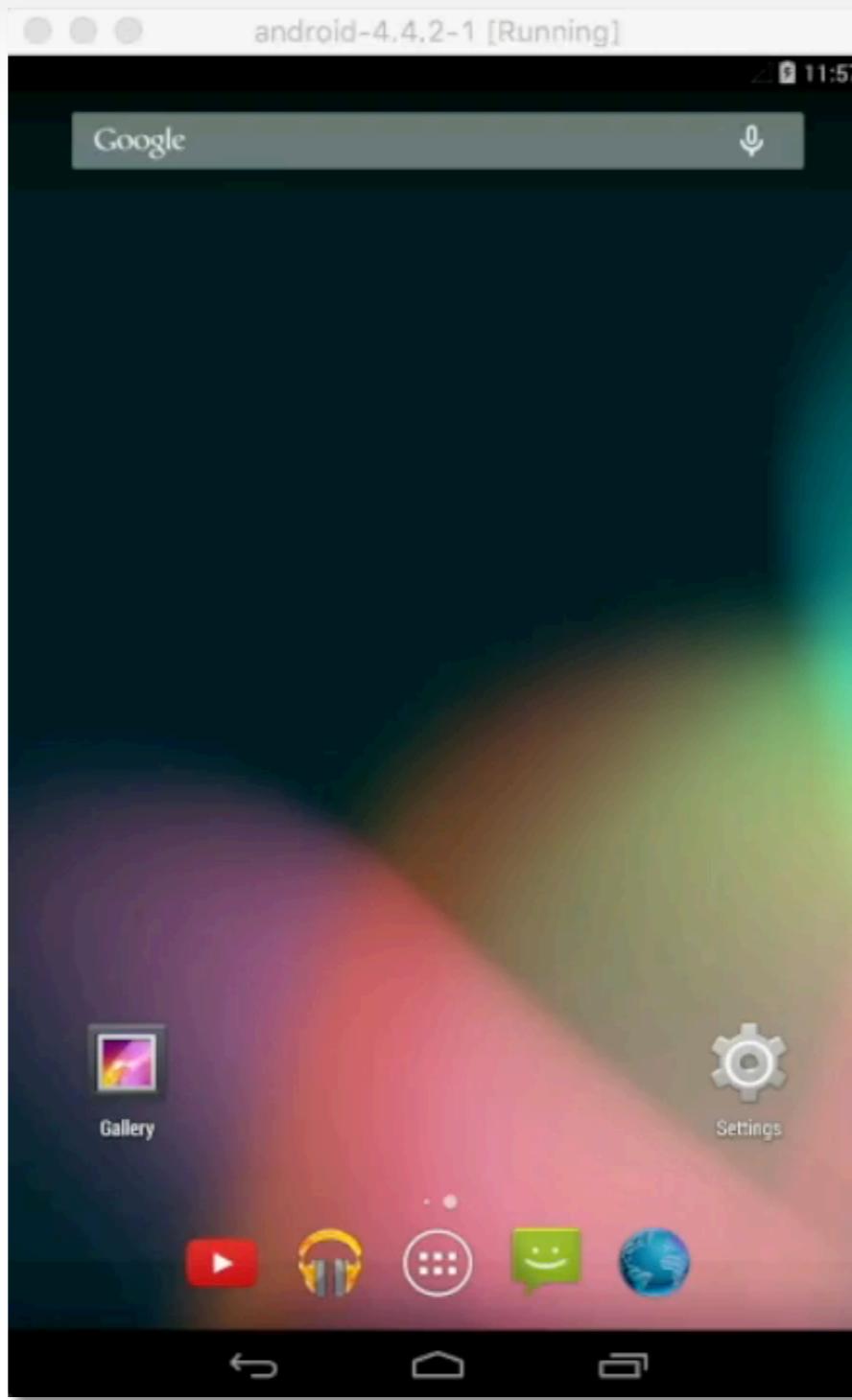
# CRASHSCOPE: Report and Script Generation



# CRASHSCOPE: Exploration Demo



# CRASHSCOPE: Exploration Demo



# CRASHSCOPE: Reports

## CrashScope Report

### 1 Contextual information

Application Name & Version: GnuCash 1.0.3   Android Version: 4.4.4   Date reported: 2015-08-17

Android SDK built for x86   PORTRAIT 1200x1920

Icon Legend:

Accelerometer ON/OFF = Network ON/OFF = GPS ON/OFF = Magnetometer ON/OFF = Temperature Sensor ON/OFF =

### 2 Steps to Reproduce

The following touch events and inputs will reproduce the crash on the device, app version, and android version specified above:

\*Please note that the following steps are representative, you may not have to perform every action or type all of the text characters in order to reproduce the crash.

Step 1 Tap on the "Expenses" CheckedTextView, which is located on the Center of the screen.

Expenses

↓

Step 2 Tap on the "Income" CheckedTextView, which is located on the Center of the screen.

Income

↓

Step 3 Tap on the "Assets" CheckedTextView, which is located on the Center of the screen.

Assets

↓

Step 4 Tap on the "Entertainment" CheckedTextView, which is located on the Center of the screen.

Entertainment

↓

Step 5 Tap on the "Insurance" CheckedTextView, which is located on the Center of the screen.

Insurance

↓

Step 6 Tap on the "Expenses" CheckedTextView, which is located on the Center of the screen.

Expenses

# CRASHSCOPE: Reports

Step 7 Tap on the "Income" CheckedTextView, which is located on the Center of the screen.

↓

Step 8 Tap on the "Assets" CheckedTextView, which is located on the Center of the screen.

↓

Step 9 Tap on the "Entertainment" CheckedTextView, which is located on the Center of the screen.

↓

Step 10 Tap on the "Insurance" CheckedTextView, which is located on the Center of the screen.

↓

Step 11 Tap on the "Cancel" Button, which is located on the Center left of the screen.

↓

Step 12 Tap on the "Accounts" LinearLayout, which is located on the Top left of the screen.

↓

Step 13 Tap on the "" TextView, which is located on the Top right of the screen.

↓

Step 14 Type "sphvxvfriq%sz" on the "Account name" EditText, which is located on the Center of the screen.

↓

Step 15 Tap on the "" Spinner, which is located on the Center of the screen.

↓

Step 16 Tap on the "US Dollar" CheckedTextView, which is located on the Top of the screen.

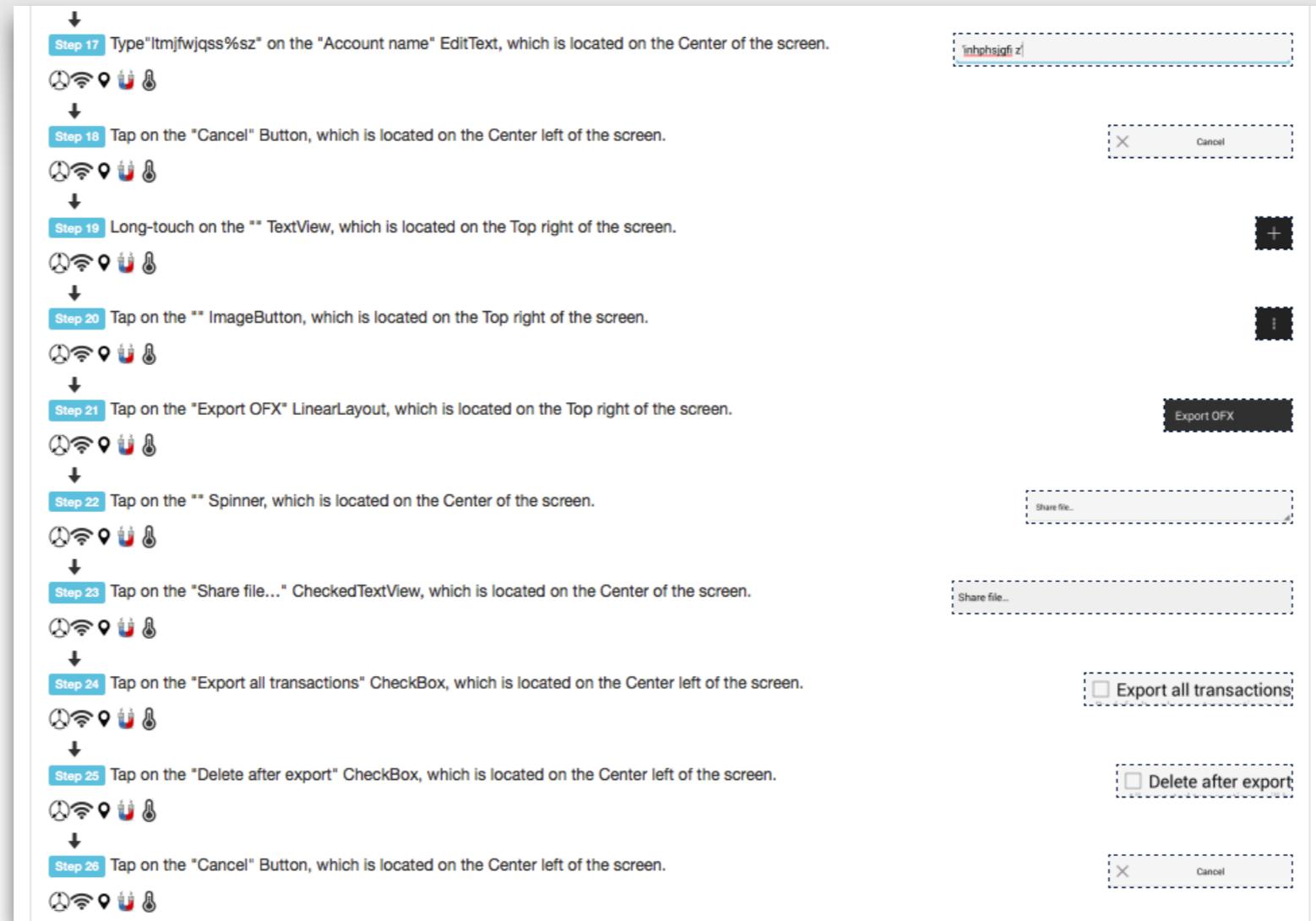
↓

The screenshot shows the CRASHSCOPE app's user interface. It includes several sections with labels and checkboxes:

- "Income" (checkbox checked)
- "Assets" (checkbox checked)
- "Entertainment" (checkbox checked)
- "Insurance" (checkbox checked)
- "Cancel" button
- "Accounts" button with a plus sign
- "US Dollar" spinner

Below the UI, there are 16 steps with icons indicating device status (GPS, WiFi, battery, signal) and a downward arrow between each step.

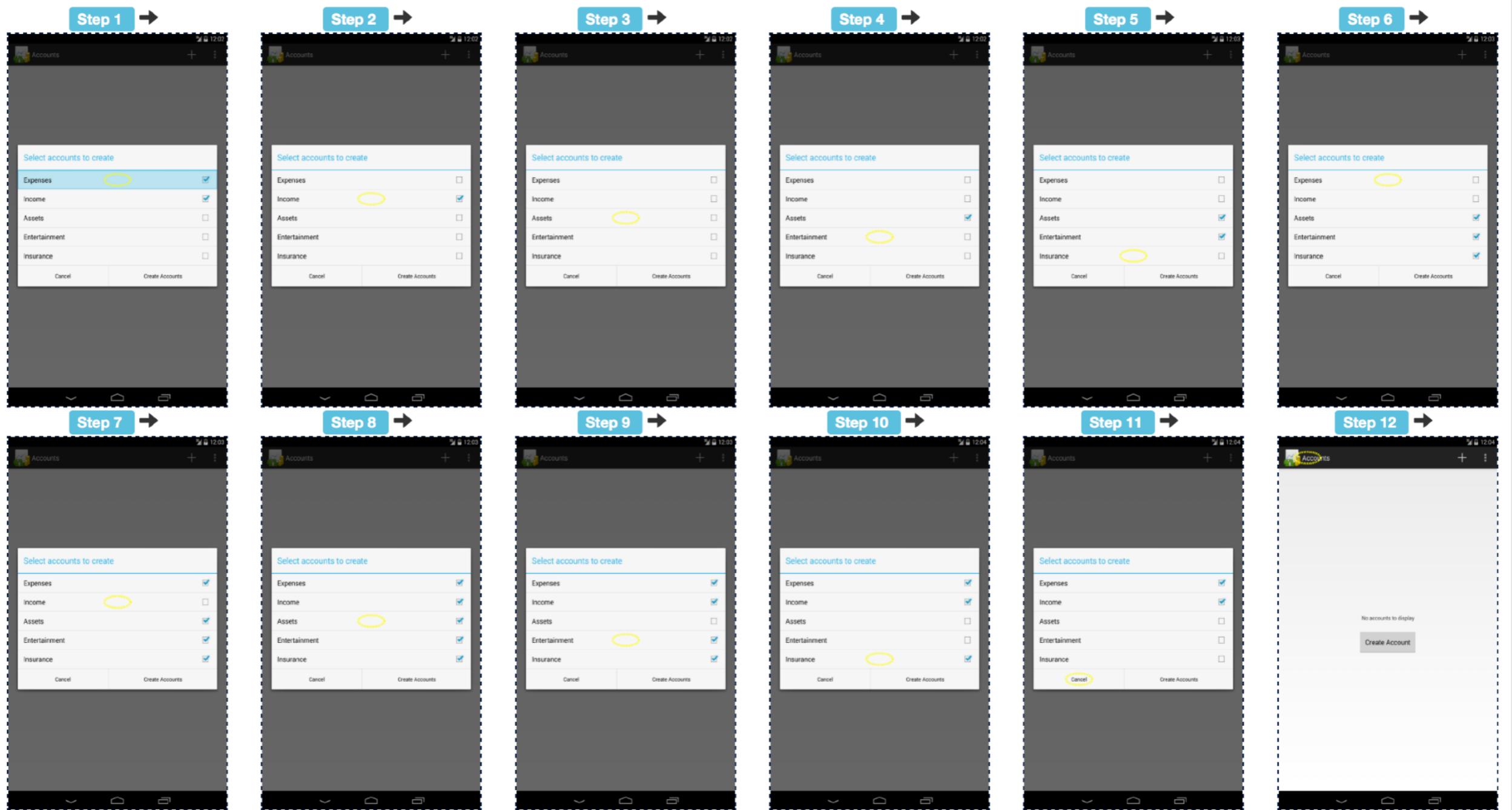
# CRASHSCOPE: Reports



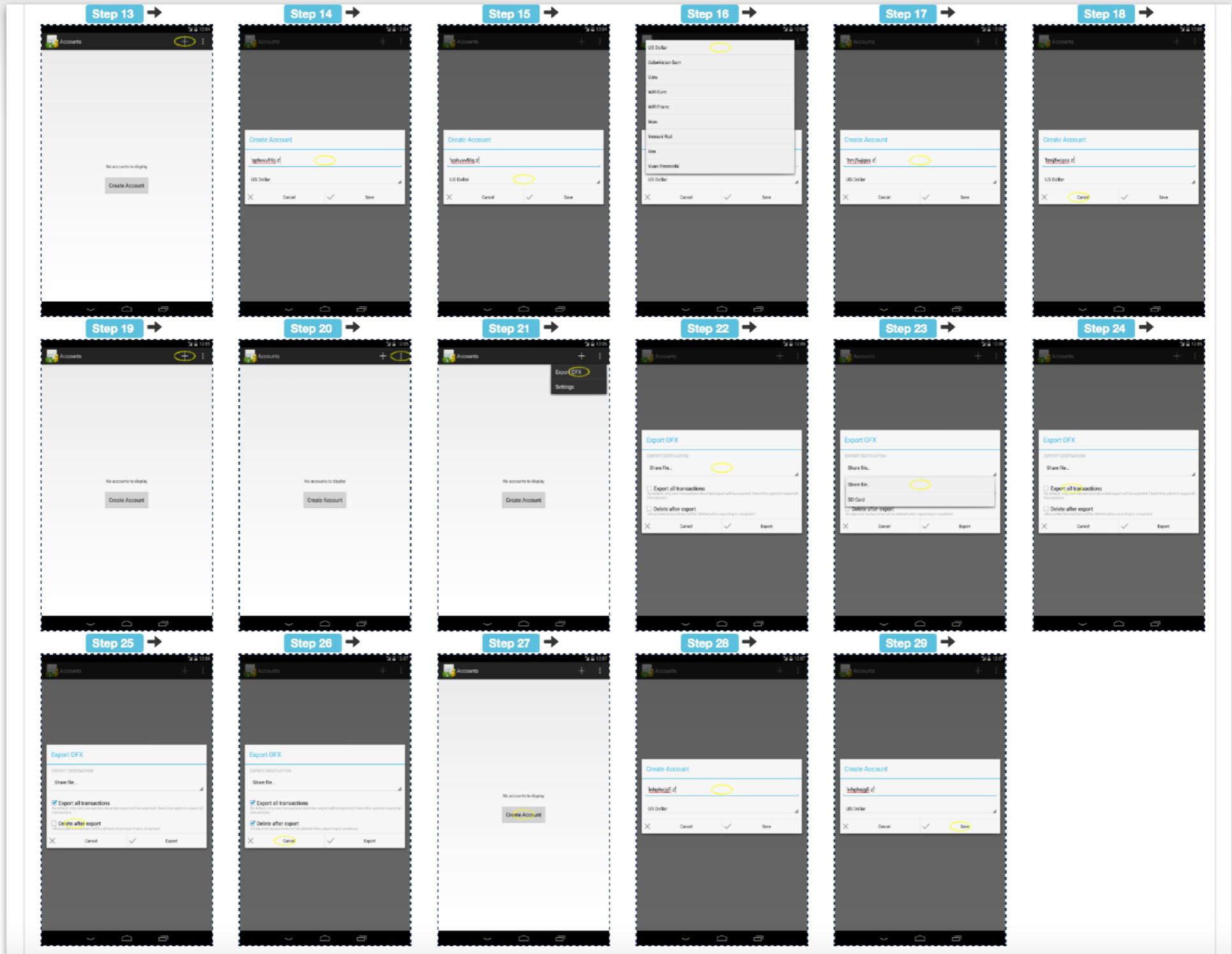
# CRASHSCOPE: Reports

## 3 Crash Application Screen-Flow

(Go top)



# CRASHSCOPE: Reports



# CRASHSCOPE: Reports

## 4 Crash Pruned Stack Trace

[\(Go top\)](#)

```
E/SQLiteLog(17653): (1) near "inhphsjgf": syntax error
E/AndroidRuntime(17653): FATAL EXCEPTION: main
E/AndroidRuntime(17653): Process: org.gnucash.android, PID: 17653
E/AndroidRuntime(17653): android.database.sqlite.SQLiteException: near "inhphsjgf": syntax error (code 1): , while compiling: SELECT _id, uid FROM accounts WHERE uid = ''inhphsjgf-d724114f522e'
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.nativePrepareStatement(Native Method)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.acquirePreparedStatement(SQLiteDatabase.java:889)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.prepare(SQLiteDatabase.java:500)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteSession.prepare(SQLiteSession.java:588)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteProgram.(SQLiteProgram.java:58)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteQuery.(SQLiteQuery.java:37)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDirectCursorDriver.query(SQLiteDirectCursorDriver.java:44)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.rawQueryWithFactory(SQLiteDatabase.java:1314)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.queryWithFactory(SQLiteDatabase.java:1161)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.query(SQLiteDatabase.java:1032)
E/AndroidRuntime(17653):         at android.database.sqlite.SQLiteDatabase.query(SQLiteDatabase.java:1200)
E/AndroidRuntime(17653):         at org.gnucash.android.db.AccountsDbAdapter.getAccountId(AccountsDbAdapter.java:166)
E/AndroidRuntime(17653):         at org.gnucash.android.db.AccountsDbAdapter.addAccount(AccountsDbAdapter.java:76)
E/AndroidRuntime(17653):         at org.gnucash.android.ui.accounts.NewAccountDialogFragment$1.onClick(NewAccountDialogFragment.java:156)
)
E/AndroidRuntime(17653):         at android.view.View.performClick(View.java:4438)
E/AndroidRuntime(17653):         at android.view.View$PerformClick.run(View.java:18422)
E/AndroidRuntime(17653):         at android.os.Handler.handleCallback(Handler.java:733)
E/AndroidRuntime(17653):         at android.os.Handler.dispatchMessage(Handler.java:95)
E/AndroidRuntime(17653):         at android.os.Looper.loop(Looper.java:136)
E/AndroidRuntime(17653):         at android.app.ActivityThread.main(ActivityThread.java:5001)
E/AndroidRuntime(17653):         at java.lang.reflect.Method.invokeNative(Native Method)
E/AndroidRuntime(17653):         at java.lang.reflect.Method.invoke(Method.java:515)
E/AndroidRuntime(17653):         at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
E/AndroidRuntime(17653):         at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
E/AndroidRuntime(17653):         at dalvik.system.NativeStart.main(Native Method)
```

# Evaluation

- Two Empirical Studies
- *Study 1:* Crash Detection Capabilities
- *Study 2:* Crash Report Reproducibility and Readability

# Study 1: Crash Detection & Coverage

- RQ<sub>1</sub>: Crash Detection Effectiveness?
- RQ<sub>2</sub>: Orthogonality of Crashes?
- RQ<sub>3</sub>: Effectiveness of Individual Strategies?
- RQ<sub>4</sub>: Does Crash Detection Correlate with Code Coverage?

# Study 1:Experimental Setup

Tools Used In The Comparative Fault Finding Study

Tool Name	Android Version	Tool Type
Monkey	Any	Random
A3E Depth-First	Any	Systematic
GUI-Ripper	Any	Model-Based
Dynodroid	v2.3	Random-Based
PUMA	v4.1+	Random-Based

- 61 subject applications from the *Androtest*<sup>1</sup> toolset
- Each testing tool was run 5 separate times for 1 hour, whereas CrashScope ran through all strategies
- Monkey was limited by the number of events

---

<sup>1</sup>S. R. Choudhary, A. Gorla, and A. Orso. Automated Test Input Generation for Android: Are we there yet? In 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2015

# Study 1: Crash Results

Unique Crashes Discovered With Instrumented Crashes in Parentheses

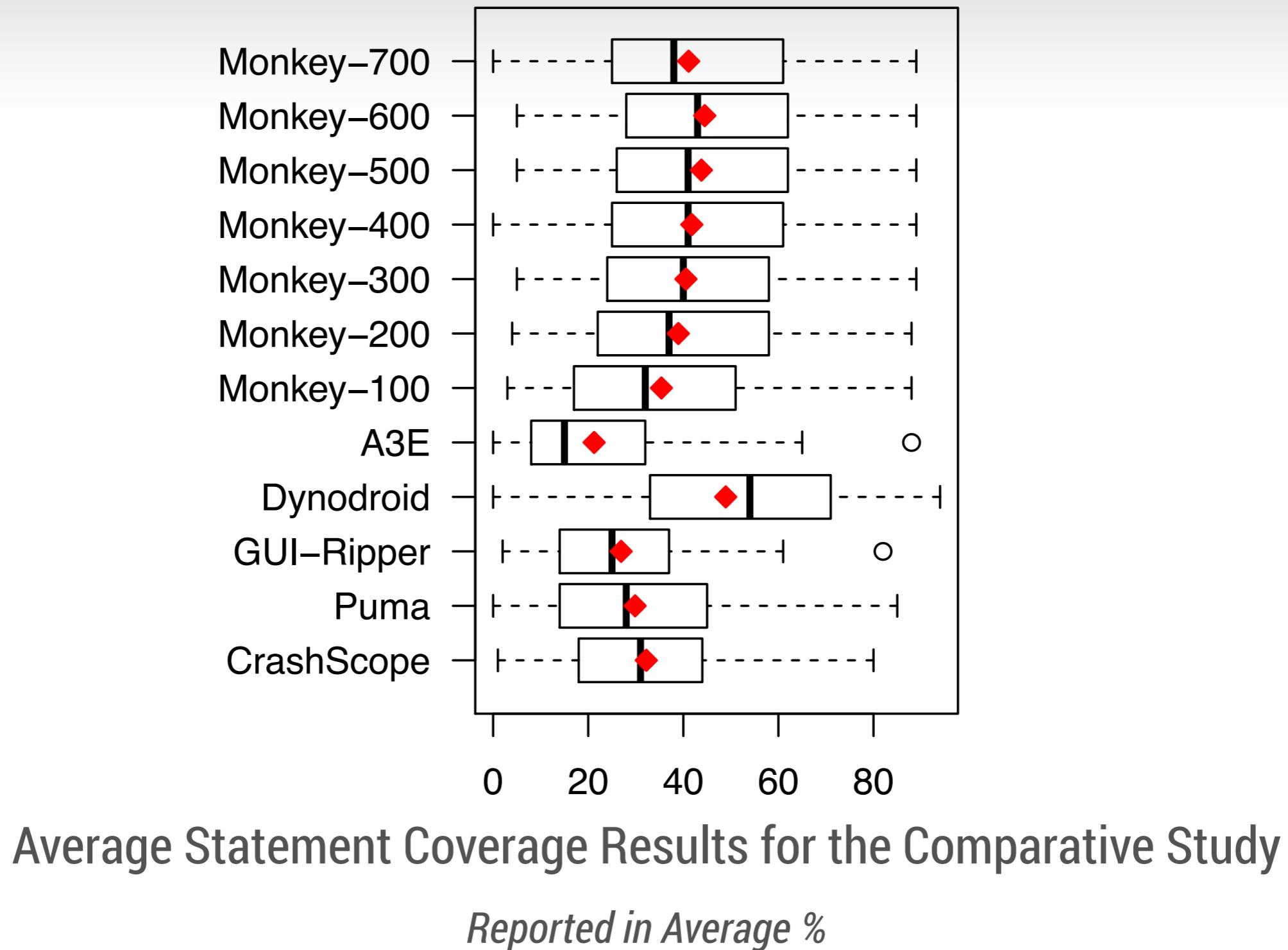
App	A3E	GUI- Ripper	Dynodroid	PUMA	Monkey (All)	CrashScope
<b>A2DP Vol</b>	1	0	0	0	0	0
<b>aagtl</b>	0	0	1	0	1	0
<b>Amazed</b>	0	0	0	0	1	0
<b>HNDroid</b>	1	1	1	2	1	1
<b>BatteryDog</b>	0	0	1	0	1	0
<b>Soundboard</b>	0	1	0	0	0	0
<b>AKA</b>	0	0	0	0	1	0
<b>Bites</b>	0	0	0	0	1	0
<b>Yahtzee</b>	1	0	0	0	0	1
<b>ADSDroid</b>	1	1	1	1	1	1
<b>PassMaker</b>	1	0	0	0	1	1
<b>BlinkBattery</b>	0	0	0	0	1	0
<b>D&amp;C</b>	0	0	0	0	1	0
<b>Photostream</b>	1	1	1	1	1	0
<b>AlarmKlock</b>	0	0	1	0	0	0
<b>Sanity</b>	1	1	0	0	0	0
<b>MyExpenses</b>	0	0	1	0	0	0
<b>Zooborns</b>	0	0	0	0	0	2
<b>ACal</b>	1	2	2	0	1	1
<b>Hotdeath</b>	0	2	0	0	0	1
<b>Total</b>	<b>8 (21)</b>	<b>9 (5)</b>	<b>9 (6)</b>	<b>4 (0)</b>	<b>12 (1)</b>	<b>8 (0)</b>

# Study 1: Crash Results

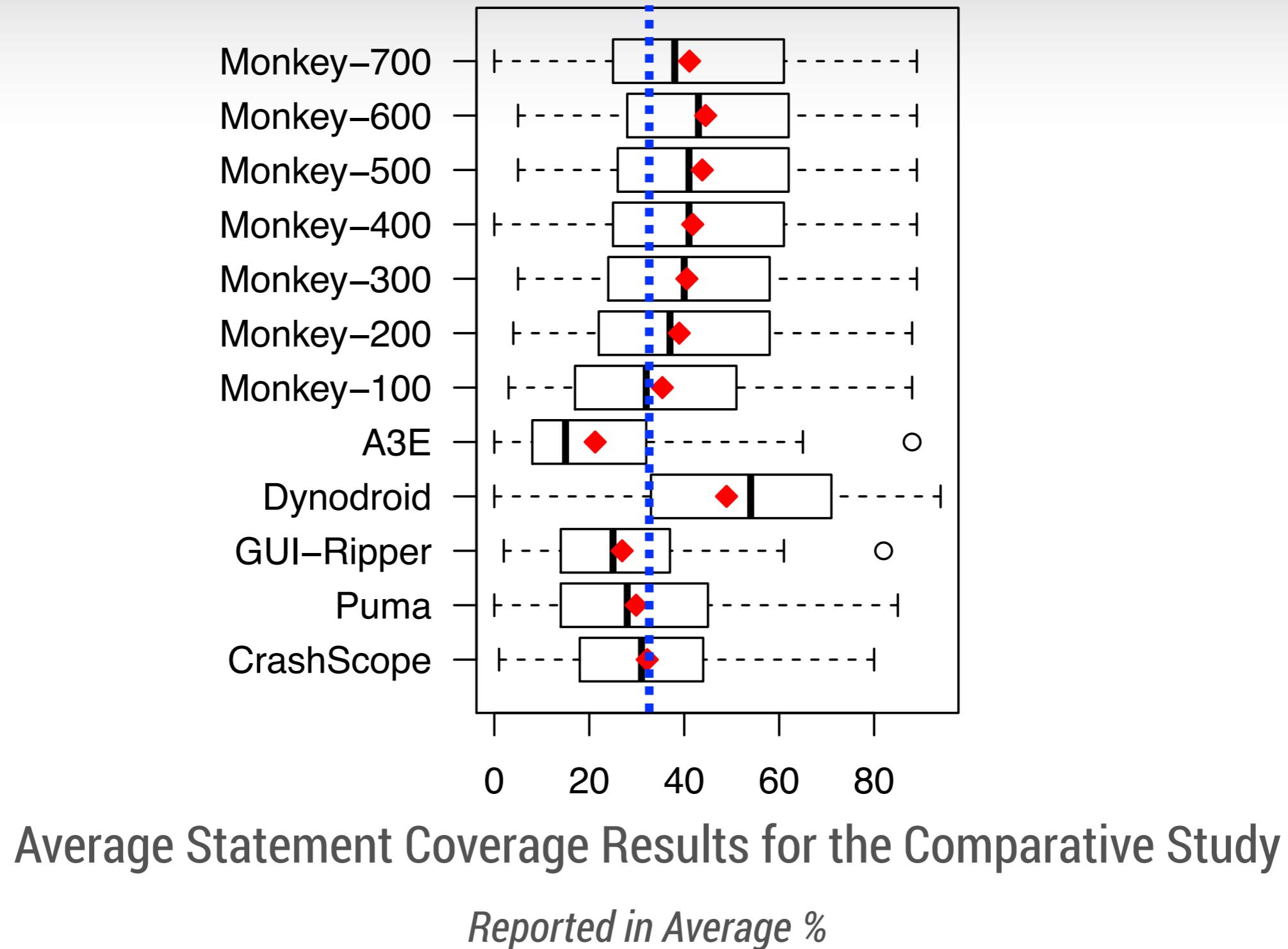
Unique Crashes Discovered With Instrumented Crashes in Parentheses

App	A3E	GUI- Ripper	Dynodroid	PUMA	Monkey (All)	CrashScope
A2DP Vol	1	0	0	0	0	0
aagtl	0	0	1	0	1	0
Amazed	0	0	0	0	1	0
HNDroid	1	1	1	2	1	1
BatteryDog	0	0	1	0	1	0
Soundboard	0	1	0	0	0	0
AKA	0	0	0	0	1	0
Bites	0	0	0	0	1	0
Yahtzee	1	0	0	0	0	1
ADSDroid	1	1	1	1	1	1
PassMaker	1	0	0	0	1	1
BlinkBattery	0	0	0	0	1	0
D&C	0	0	0	0	1	0
Photostream	1	1	1	1	1	0
AlarmKlock	0	0	1	0	0	0
Sanity	1	1	0	0	0	0
MyExpenses	0	0	1	0	0	0
Zooborns	0	0	0	0	0	2
ACal	1	2	2	0	1	1
Hotdeath	0	2	0	0	0	1
Total	8 (21)	9 (5)	9 (6)	4 (0)	12 (1)	8 (0)

# Study 1: Statement Coverage Results



# Study 1: Statement Coverage Results



# Study 1: Summary of Findings

- RQ<sub>1</sub>: CrashScope is nearly as effective at discovering crashes as the other tools, without reporting crashes caused by instrumentation
- RQ<sub>2&3</sub>: CrashScope's differing strategies led to the discovery of unique crashes
- RQ<sub>4</sub>: Higher statement coverage does not necessarily correspond with crash detection capabilities

# Study 2: Reproducibility & Readability

- RQ<sub>5</sub>: Reproducibility of CrashScope Reports?
- RQ<sub>6</sub>: Readability of CrashScope Reports?

# Study 2: Experimental Setup

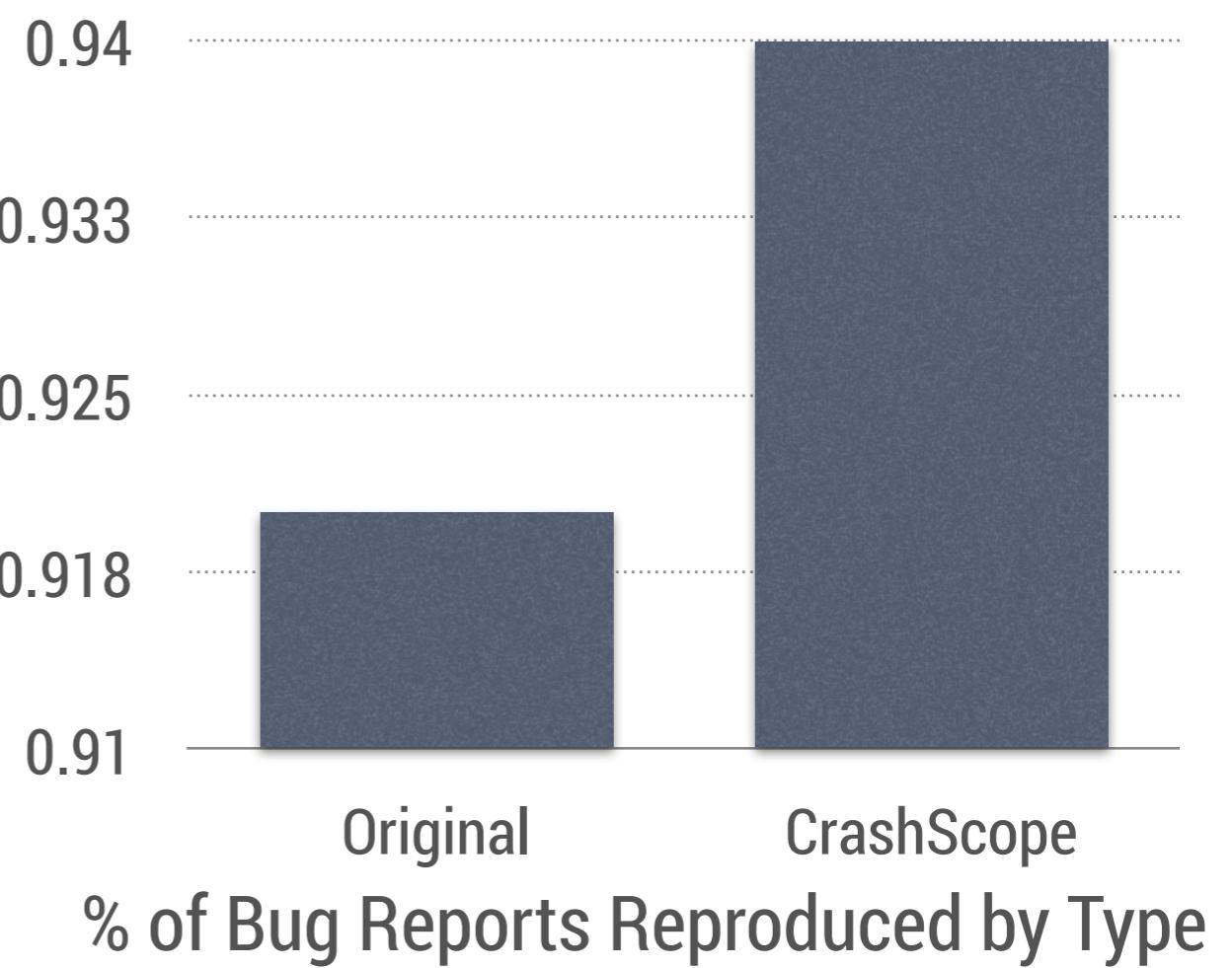
- *8 Real-World Crash Reports from Open Source Apps*
- *16 Graduate Students from the College of William & Mary*

Application Name	# of Reproduction Steps
BMI	4
Schedule	7
adsdroid	2
Anagram-solver	7
Eyecam	14
GNU Cash	29
Olam	2
CardGame Scores	23

- Each student attempted to reproduce 8 bugs: 4 from the original reports, 4 from CrashScope Reports
- Participants used a Nexus 7 tablet for reproduction

# Study 2: Reproducibility Results

Type of Crash Report	# of Total/Non-Reproducible Reports
Original Bug Reports	59/64
CrashScope Bug Reports	60/64



# Study 2: Readability Results

Question	CrashScope Mean	CrashScope StdDev	Original Mean	Original StdDev
<b>UX1: I think I would like to have this type of bug report frequently.</b>	4.00	0.89	3.06	0.77
<b>UX2: I found this type of bug report unnecessarily complex.</b>	2.81	1.04	2.125	0.96
<b>UX3: I thought this type of bug report was easy to read/understand.</b>	4.00	0.82	3.00	0.97
<b>UX4: I found this type of bug report very cumbersome to read.</b>	2.50	1.10	2.44	0.81
<b>UX5: I thought the bug report was very useful for reproducing the crash.</b>	4.13	0.62	3.44	0.89

# Study 2: Readability Results

Question	CrashScope Mean	CrashScope StdDev	Original Mean	Original StdDev
<b>UX1: I think I would like to have this type of bug report frequently.</b>	4.00	0.89	3.06	0.77
<b>UX2: I found this type of bug report unnecessarily complex.</b>	2.81	1.04	2.125	0.96
<b>UX3: I thought this type of bug report was easy to read/understand.</b>	4.00	0.82	3.00	0.97
<b>UX4: I found this type of bug report very cumbersome to read.</b>	2.50	1.10	2.44	0.81
<b>UX5: I thought the bug report was very useful for reproducing the crash.</b>	4.13	0.62	3.44	0.89

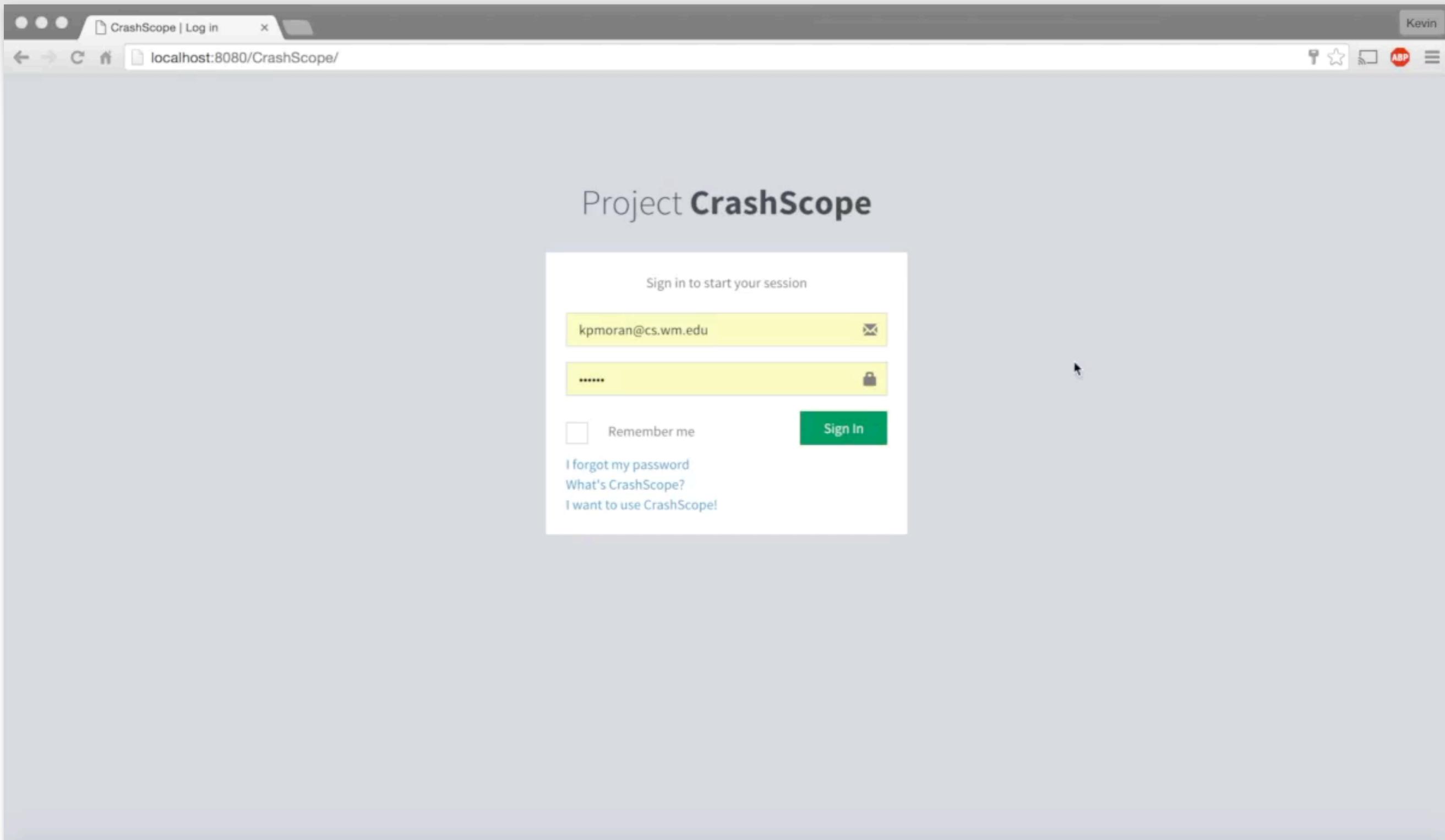
# Study 2: Readability Results

Question	CrashScope Mean	CrashScope StdDev	Original Mean	Original StdDev
<b>UX1: I think I would like to have this type of bug report frequently.</b>	4.00	0.89	3.06	0.77
<b>UX2: I found this type of bug report unnecessarily complex.</b>	2.81	1.04	2.125	0.96
<b>UX3: I thought this type of bug report was easy to read/understand.</b>	4.00	0.82	3.00	0.97
<b>UX4: I found this type of bug report very cumbersome to read.</b>	2.50	1.10	2.44	0.81
<b>UX5: I thought the bug report was very useful for reproducing the crash.</b>	4.13	0.62	3.44	0.89

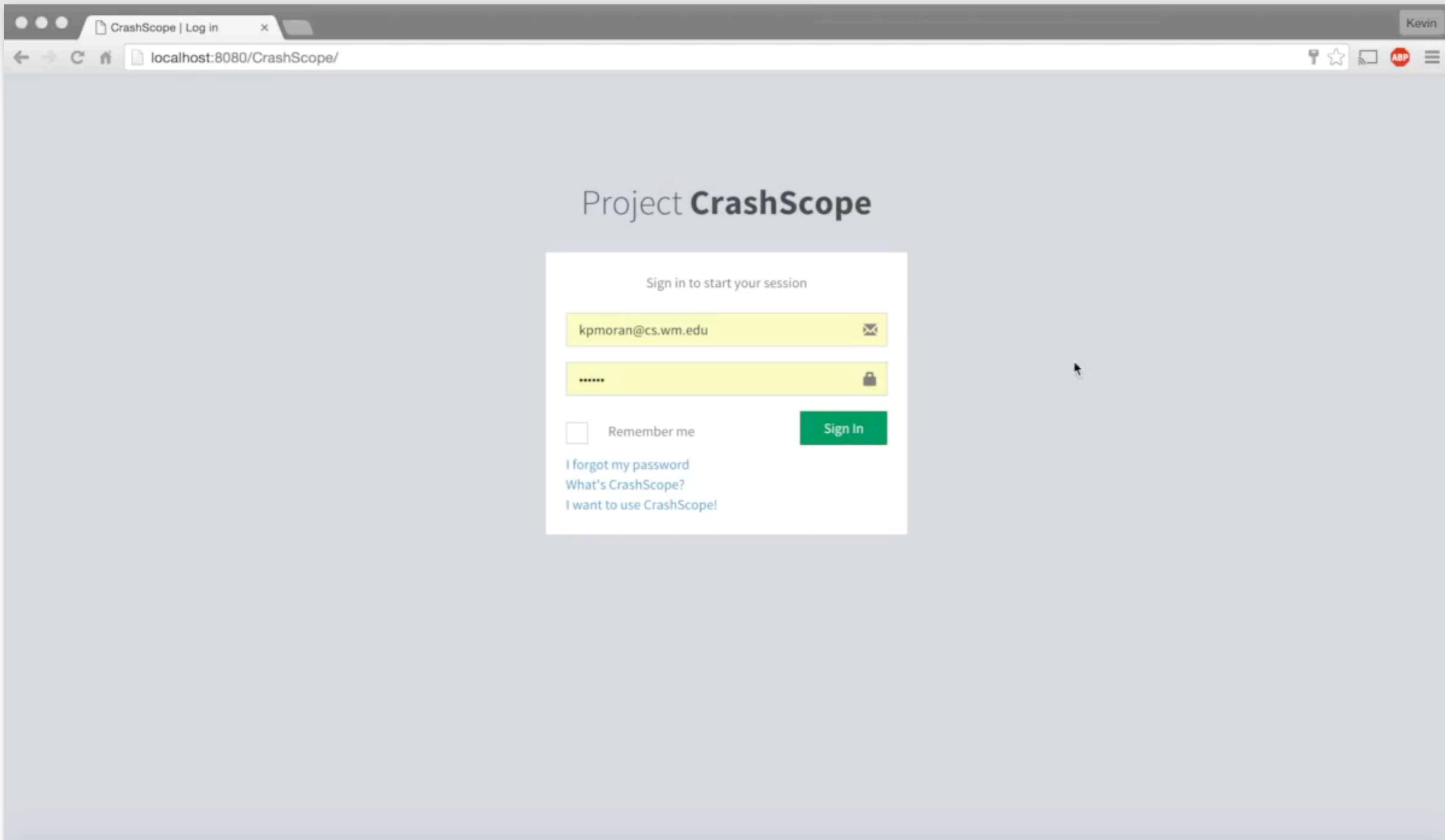
# Study 2: Summary of Findings

- RQ<sub>5</sub>: Reports generated by CrashScope are about as reproducible as human written reports extracted from open-source issue trackers
- RQ<sub>6</sub>: Reports generated by CrashScope are more readable and useful from a developers' perspective compared to human-written reports.

# CRASHSCOPE: A Practical Tool



# CRASHSCOPE: A Practical Tool



# THANK YOU !!

## QUESTIONS/DISCUSSION?



[kpmoran@gmu.edu](mailto:kpmoran@gmu.edu)

# Hands-On Session

<https://sagelab.io/crashscope-tutorial/>



# Discussion Questions

- Potential solutions to challenges we covered?
- Other future research directions?
- How can mobile testing techniques cope with AR/  
VR environments?