Automated Testing of Mobile Apps

Mayur Naik

Georgia Institute of Technology

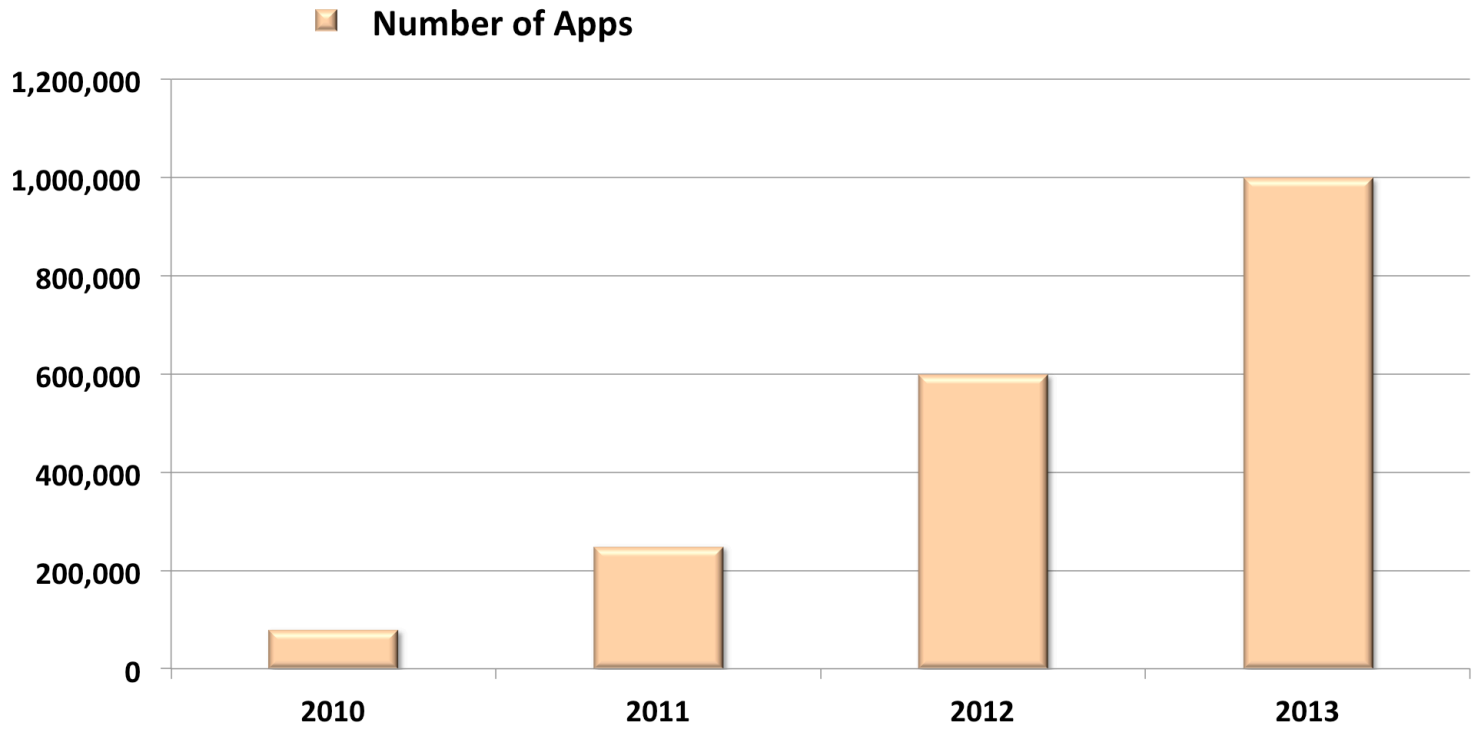
Joint work with:
Aravind Machiry and Rohan Tahiliani

The Growth of Smartphones and Tablets

- 1 million new Android devices activated every day
- 750 million total (March 2013)

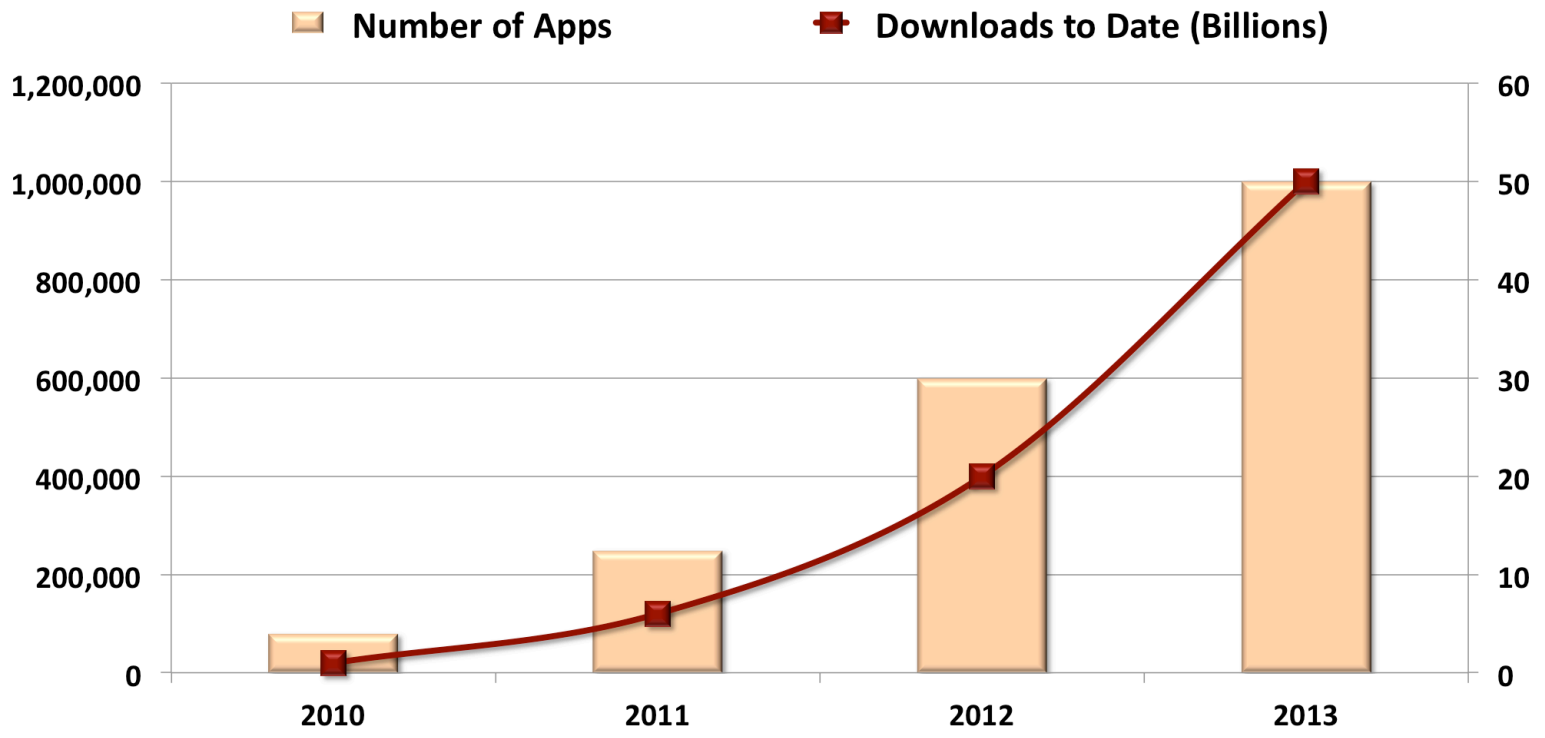
The Growth of Mobile Apps

- 30K new apps on Google Play per month
- 1 million total (July 2013)



The Growth of Mobile Apps

- 1.5 billion downloads from Google Play per month
- 50 billion total (July 2013)



The Life of a Mobile App

Reliability | **Security** | **Performance**

Development and testing

Pre-deployment Certification

Post-deployment Adaptation

- New software engineering problems in all stages

  $\Rightarrow$ need new program analysis-based tools

Program Analysis for Mobile Apps

- Static Analysis
  - Program analysis using program text
  - **Hindered by features common in mobile apps**
    - Large SDK, obfuscated and native code, concurrency, IPC, databases, GUIs, …
- Dynamic Analysis
  - Program analysis using program runs
  - **Needs test inputs yielding high app coverage**
    - Focus of our work

Desiderata for Input Generation System

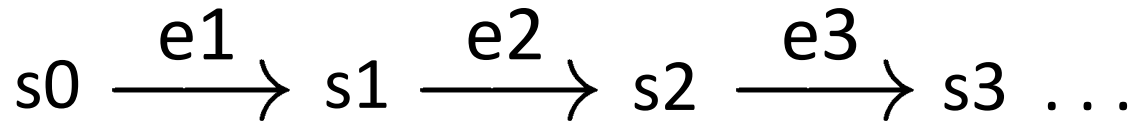
- **Robust**: handles real-world apps
- **Black-box:** does not need sources or ability to decompile binaries
- **Versatile:** exercises important app functionality
- **Automated:** reduces manual effort
- **Efficient:** avoids generating redundant inputs

Our Contributions

- Design of a system Dynodroid satisfying the five desired criteria
- Open-source implementation of Dynodroid on the dominant Android platform
- Evaluation of Dynodroid on real-world apps against state-of-the-art approaches

Our Approach

- View an app is an event-driven program



- Broadly two kinds of events:
  - **UI event:** LongTap(245, 310), Drag(0, 0, 245, 310), …
  - **System event:** BatteryLow, SmsReceived(“hello”), ...

- Assumption: Fixed concrete data in each event and environment (sdcard, network, etc.)
  - May cause loss of coverage

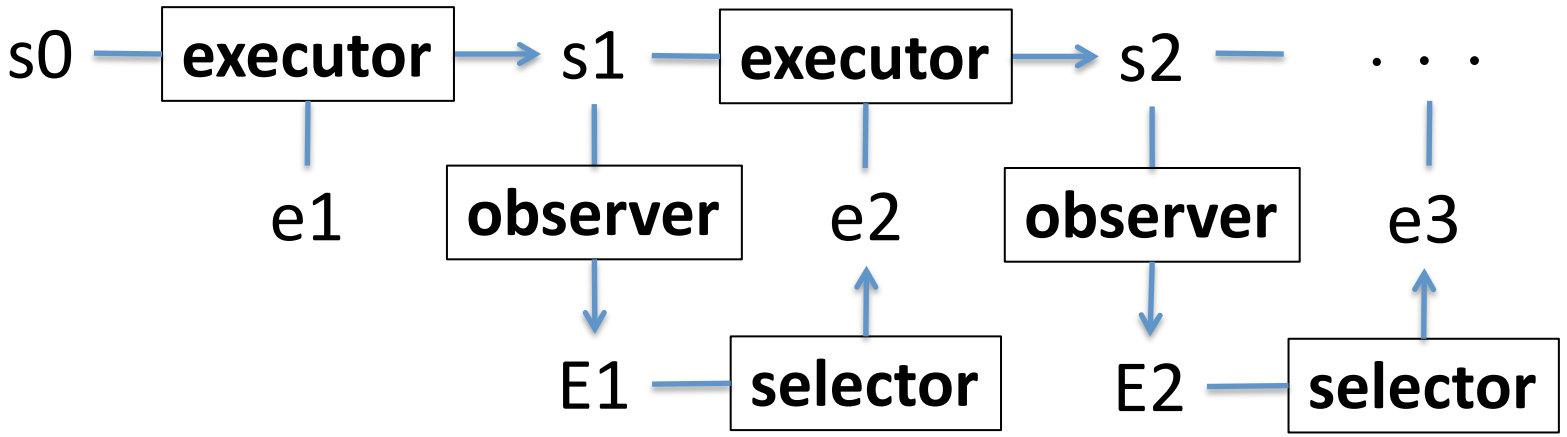
Relevant Events

- Key challenge: Large number of possible events
  - E.g., 108 system events in Android Gingerbread
- Insight #1: In any state, few events are *relevant*

  $\Rightarrow$ vast majority of events are no-ops
- Insight #2: Can identify relevant events by lightly instrumenting SDK once and for all

  $\Rightarrow$ Does not require instrumenting app

Observe-Select-Execute Algorithm



- Statelessness does not cause any coverage loss in principle provided:
  - observer treats “restart app” event always relevant
  - selector is fair

Event Selection Algorithms

- Frequency
  - Selects event that has been selected least often
  - Drawback: deterministic => unfair
- UniformRandom
  - Selects event uniformly at random
  - Drawback: does not consider domain knowledge; no distinction of UI vs. system events, contexts in which event occurs, frequent vs. rare events
- BiasedRandom
  - Combines benefits of above without drawbacks

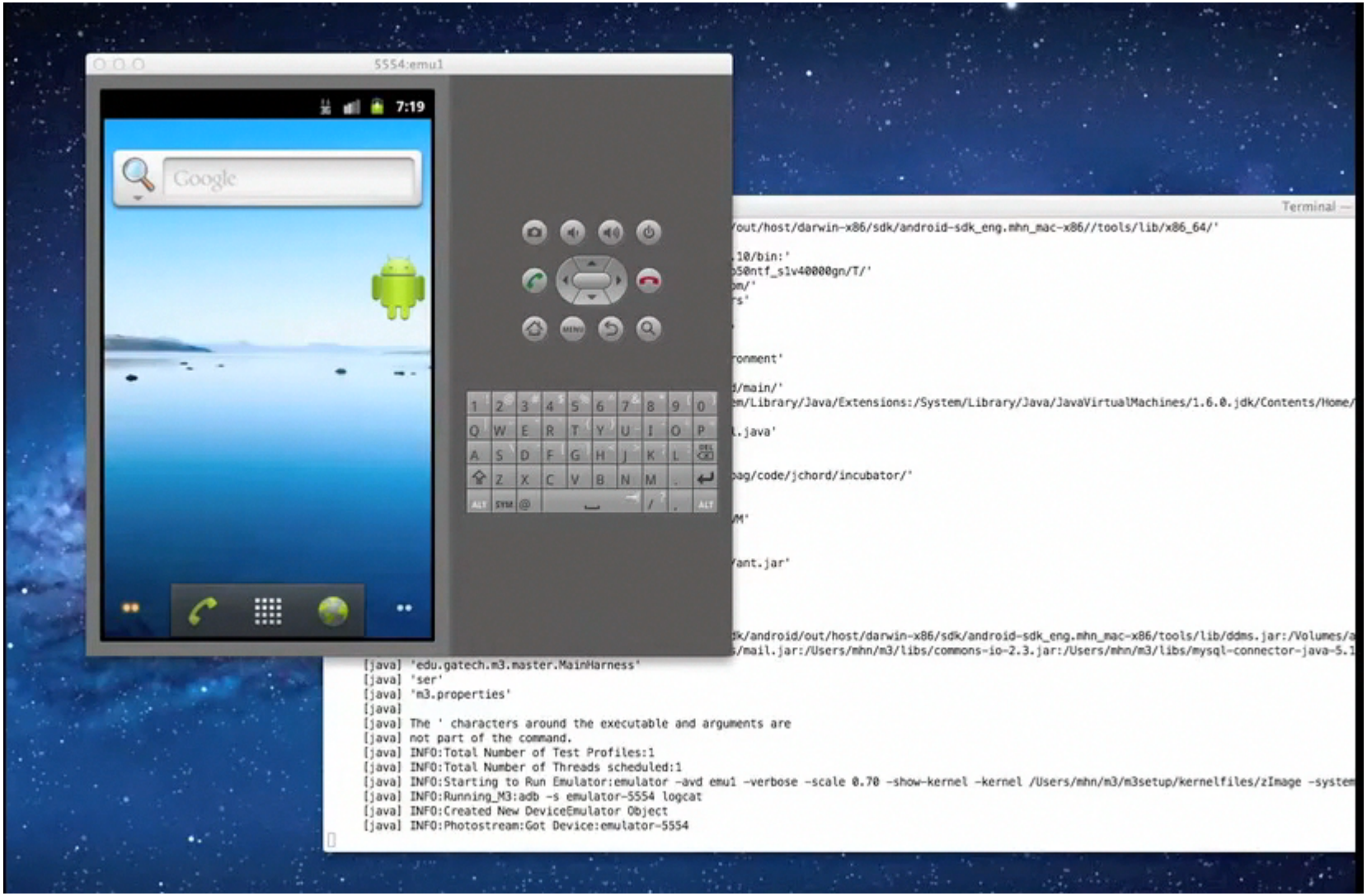
BiasedRandom Event Selection Algorithm

- Global map **G(e, S)** tracks number of times **e** is selected in context **S**
  - Context = set of events relevant when **e** is selected
- Local map **L(e)** computed to select next event from relevant set **S**
  - Initialize: **L(e)** to 0 for each **e** in **S**
  - Repeat:
    - Pick an **e** in **S** uniformly at random
    - If **L(e) = G(e, S)** increment **G(e, S)** and return **e** else increment **L(e)**
- Hallmark: No starvation

Implementation of Dynodroid

- Implemented for Android 2.3.4 (Gingerbread)
  - Covers 50% of all Android devices (March 2013)
- Modified ~ 50 lines of the SDK
  - $\Rightarrow$ Easy to port to other Android versions
- Heavily used off-the-shelf tools
  - HierarchyViewer to observe UI events
  - MonkeyRunner to execute UI events
  - ActivityManager (am) to execute system events
  - Emma to measure source code coverage
- Comprises 16 KLOC of Java
- Open-source: http://dyno-droid.googlecode.com

Demo: Dynodroid on Photostream App



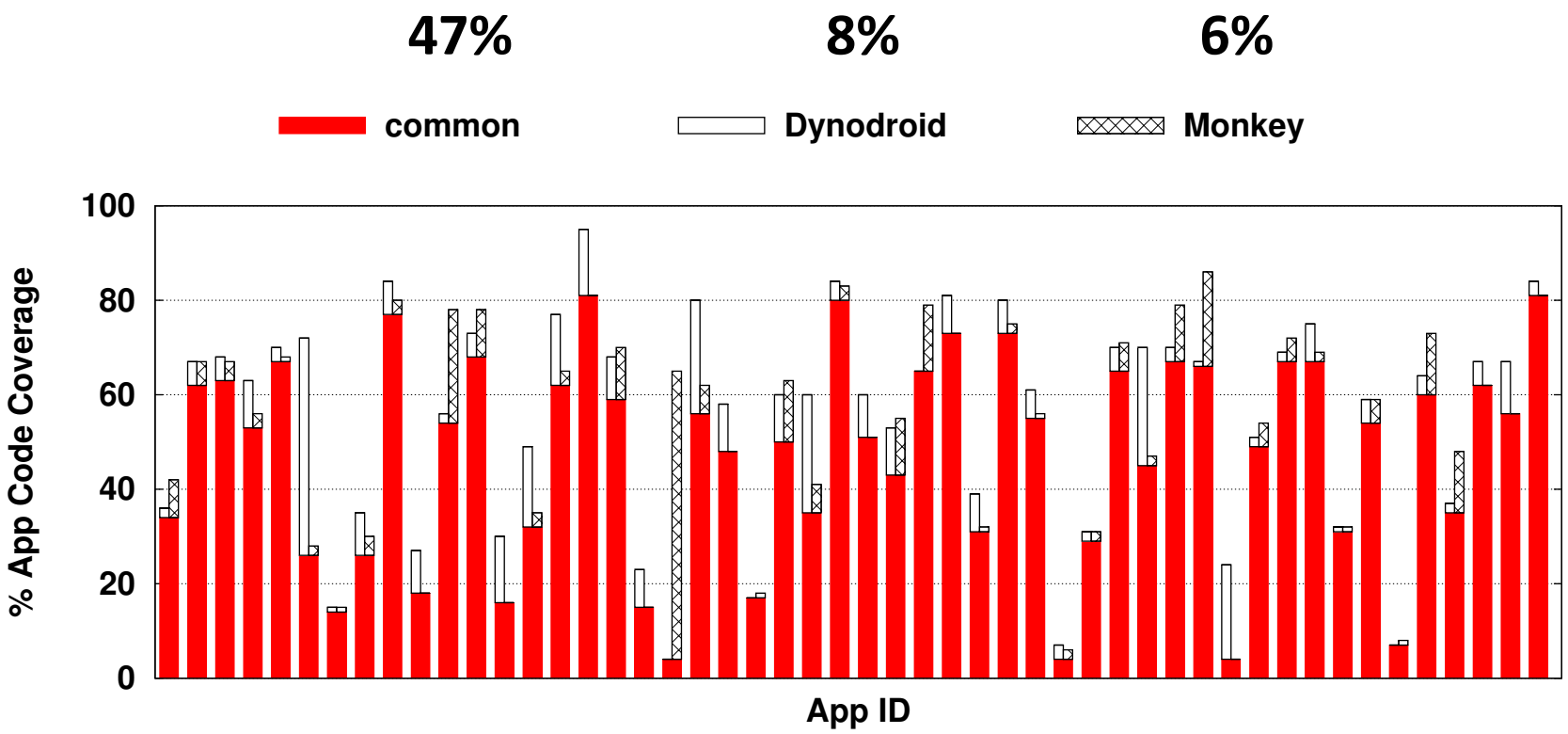
Evaluation Study 1: App Code Coverage

- 50 open-source apps from F-Droid
  - SLOC ranging from 16 to 22K, mean of 2.7K
- Evaluated Approaches:
  - Dynodroid (various configurations)
  - Monkey fuzz testing tool
  - Expert human users
    - Ten graduate students at Georgia Tech
    - All familiar with Android development

Testing Approaches Used in Our Evaluation

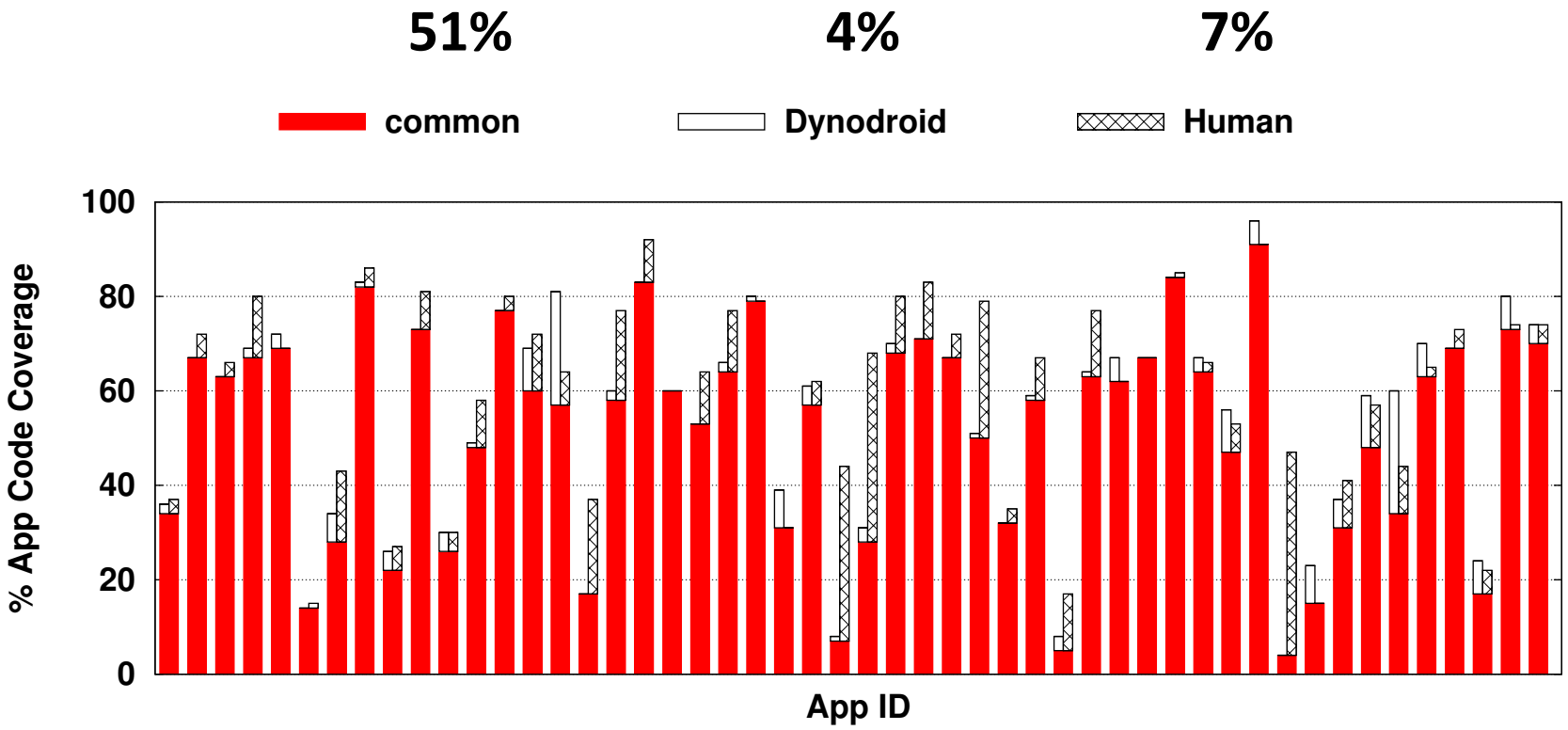
Approach	#Events	#Runs
Dynodroid - Frequency	2,000	1
Dynodroid - UniformRandom	2,000	3
Dynodroid - BiasedRandom	2,000	3
Monkey	10,000	3
Humans	No limit	>= 2

Dynodroid vs. Monkey



Dynodroid achieves higher coverage than Monkey for 30 of the 50 apps.

Dynodroid vs. Humans



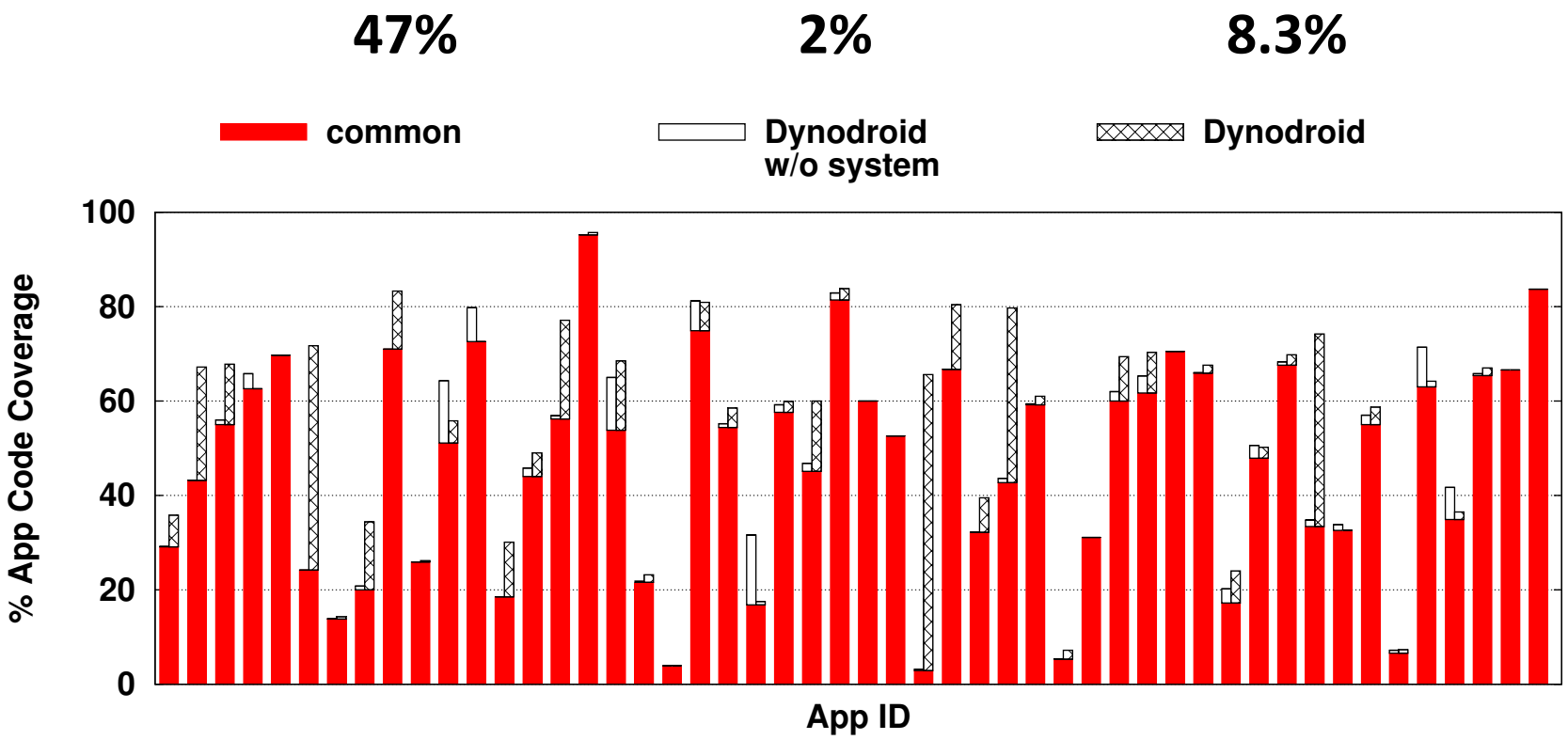
Automation Degree = C(Dynodroid ∩ Human) /C(Human)

Range = **8-100%**, Average = **83%**, S.D. = **21%**

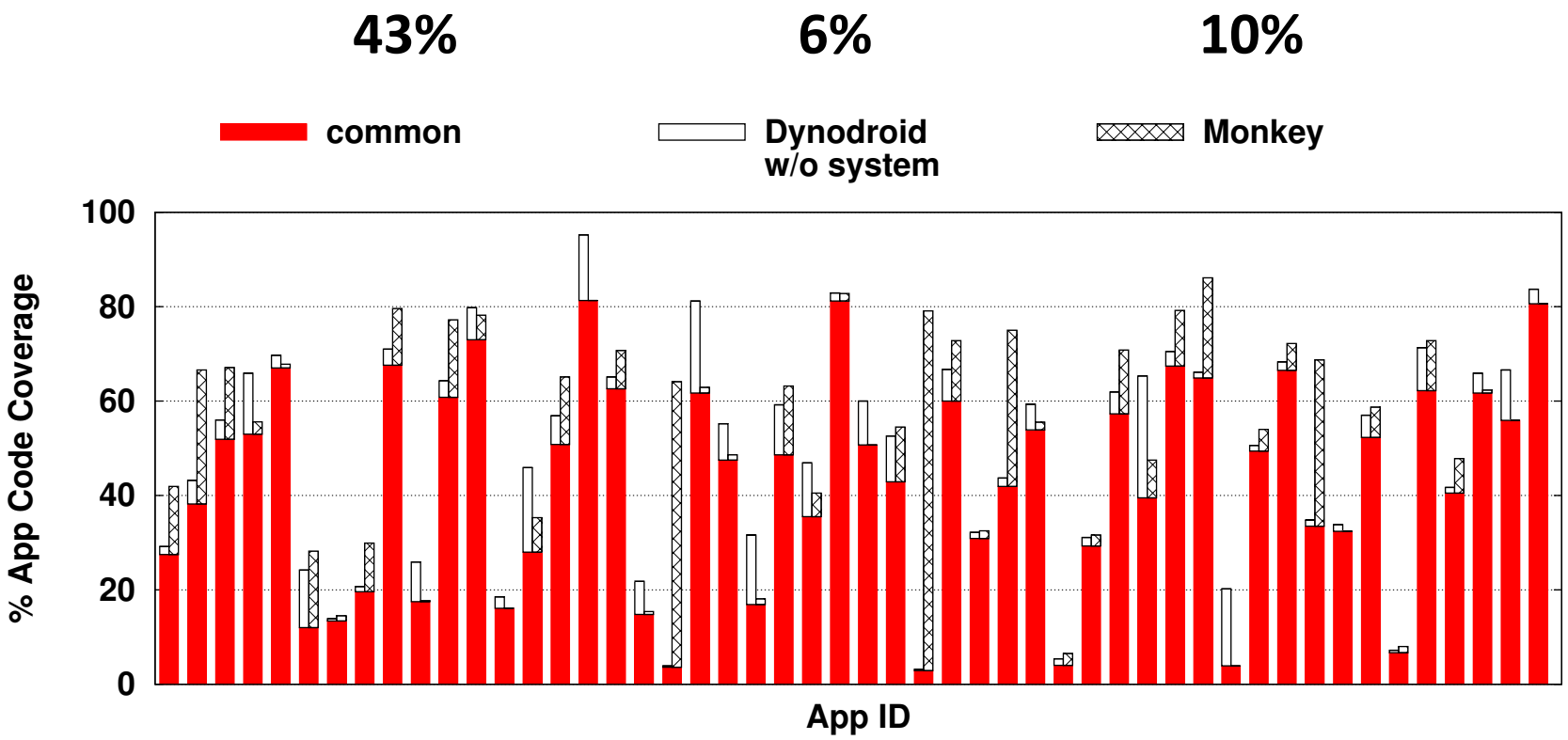
Sample Feedback from Participants

- “Tried to cancel download to raise exception.”
- “Human cannot trigger change to AudioFocus.”
- “Many, many options and lots of clicking but no actions really involved human intelligence.”
- “There are too many combinations of state changes (play -> pause, etc.) for a human to track.”

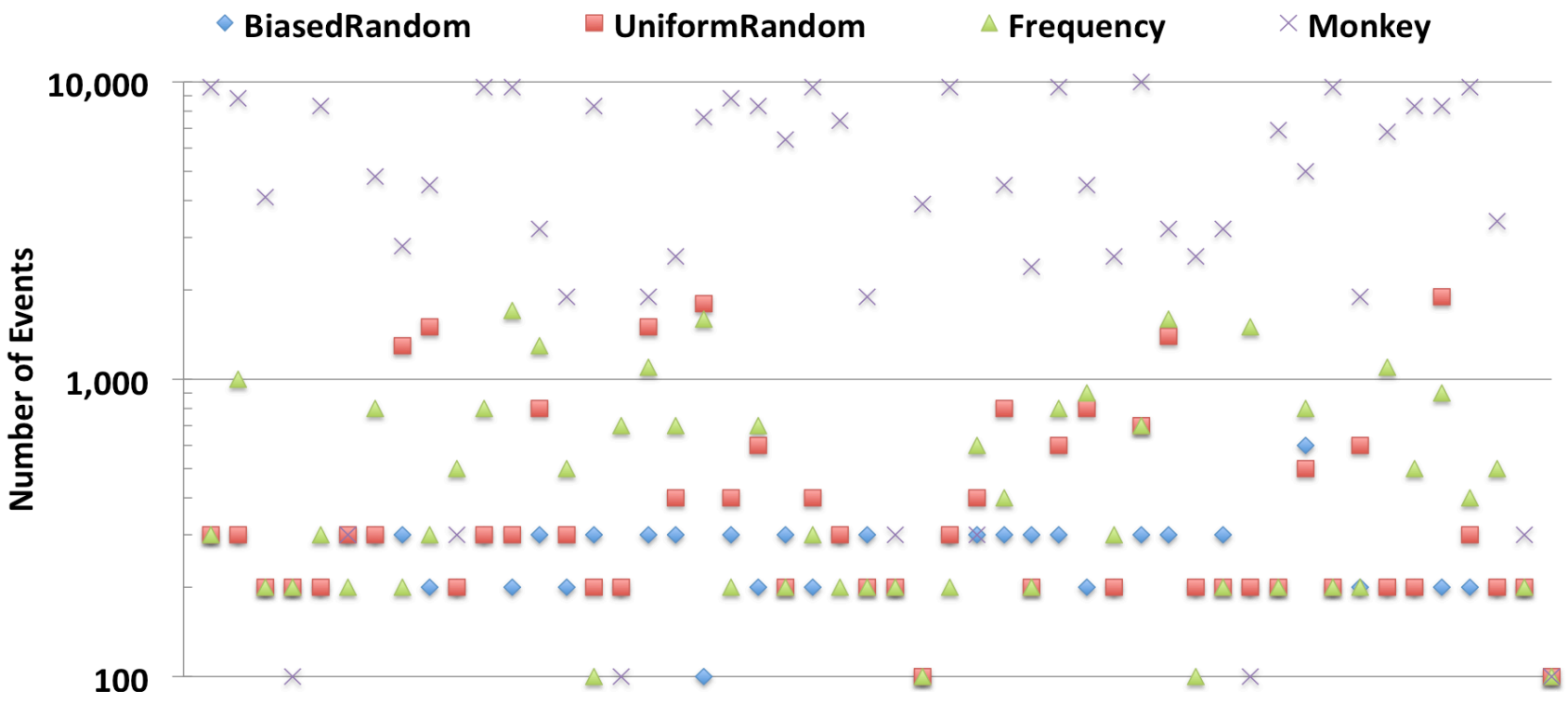
Dynodroid without vs. with System Events



Dynodroid without System Events vs. Monkey



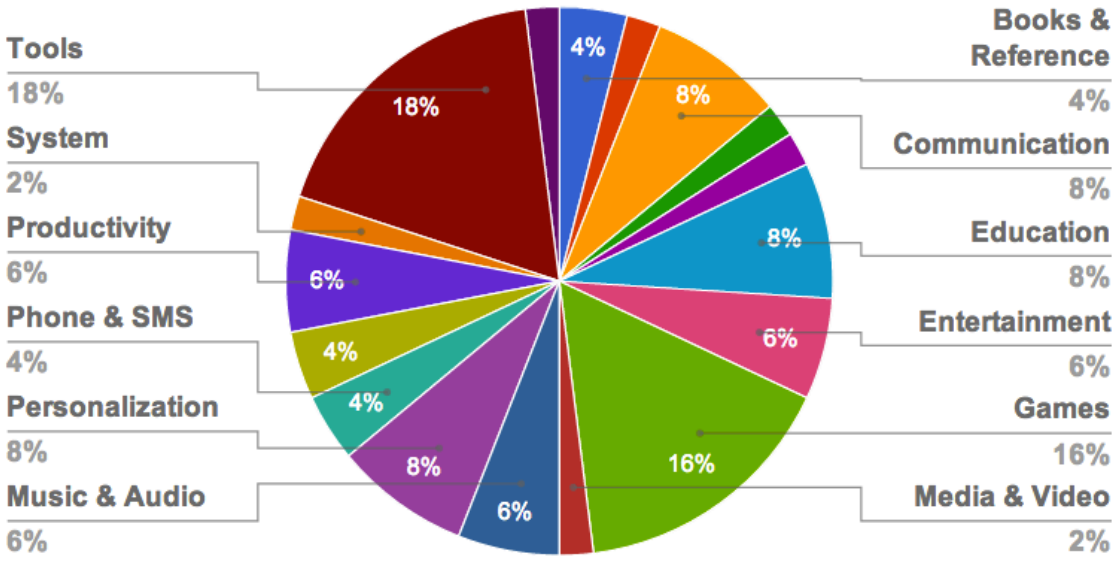
Minimum Number of Events to Peak Coverage



- Monkey requires **20X** more events than BiasedRandom
- Frequency and UniformRandom require **2X** more events than BiasedRandom

Evaluation Study 2: Bugs Found in Apps

- 1,000 most popular free apps from Google Play



- Conservative notion of bug: FATAL EXCEPTION (app forcibly terminated)

Bugs Found in 50 F-Droid Apps

App Name	Bugs	Kind	Description
PasswordMakerProForAndroid	1	Null	Improper handling of user data.
com.morphoss.acal	1	Null	Dereferencing null returned by an online service.
hu.vsza.adsdroid	2	Null	Dereferencing null returned by an online service.
cri.sanity	1	Null	Improper handling of user data.
com.zoffcc.applications.aagtl	2	Null	Dereferencing null returned by an online service.
org.beide.bomber	1	Array	Game indexes an array with improper index.
com.addi	1	Null	Improper handling of user data.

Bugs Found in 1,000 Google Play Apps

App Name	Bugs	Kind	Description
com.ibm.events.android.usopen	1	Null	Null pointer check missed in onCreate() of an activity.
com.nullsoft.winamp	2	Null	Improper handling of RSS feeds read from online service.
com.almalence.night	1	Null	Null pointer check missed in onCreate() of an activity.
com.avast.android.mobilesecurity	1	Null	Receiver callback fails to check for null in optional data.
com.aviary.android.feather	1	Null	Receiver callback fails to check for null in optional data.

Limitations

- Does not exercise inter-app communication
  - Communication via key-value maps ("Bundle" objects)
  - Could synthesize such maps symbolically
- Uses fixed, concrete data for events
  - E.g., geo-location, touch-screen coordinates, etc.
  - Could randomize or symbolically infer such data
- Requires instrumenting the platform SDK
  - $\Rightarrow$ Limited to particular SDK version
  - But lightweight enough to implement for other versions

Related Work

- Model-based Testing
  - GUITAR [ASE'12], EXSYST [ICSE'12], …
- Fuzz Testing
  - Monkey, …
- Symbolic Execution
  - Acteve [FSE'12], Symdroid, …

New Challenges: Client-Driven

- App code typically has far fewer paths than framework and third-party libraries
- Most clients care only about paths in app code


```
private void doTranslate() {
    Language from = (Language) fromButton.getTag();
    Language to = (Language) toButton.getTag();
    String fromName = from.getShortName();
    String toName = to.getShortName();
    String input = fromEditText.getText().toString();
    String result = translateService.translate(input,
          fromName, toName);
    if (result != null)
        setOutputText(result);
    else
        throw new Exception(…);
}
```

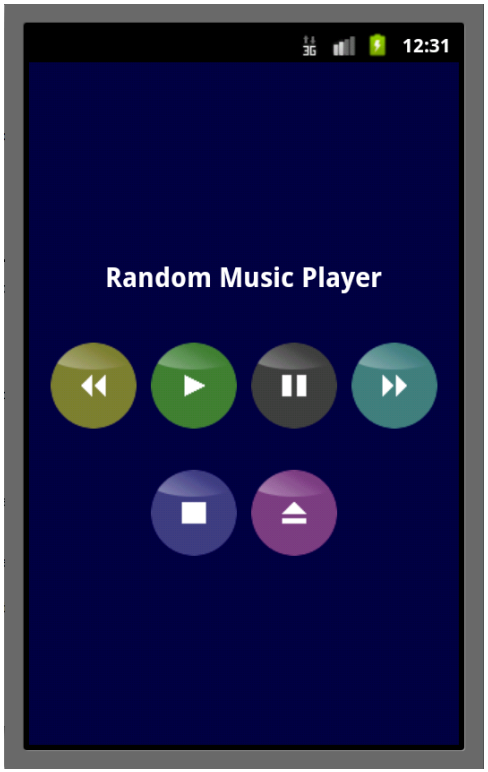
Called by framework

Calls 3rd party library

New Challenges: Mixing Static & Dynamic

Fabricate “target”; not used subsequently

```
public void onClick(View target) {
      if (target == play)
            startService(new Intent(ACTION_PLAY));
      else if (target == pause)
            startService(new Intent(ACTION_PAUSE));
      else if (target == skip)
            startService(new Intent(ACTION_SKIP));
     else if (target == rewind)
            startService(new Intent(ACTION_REWIND));
     else if (target == stop)
            startService(new Intent(ACTION_STOP));
     else if (target == eject)
            showUrlDialog();
}
```

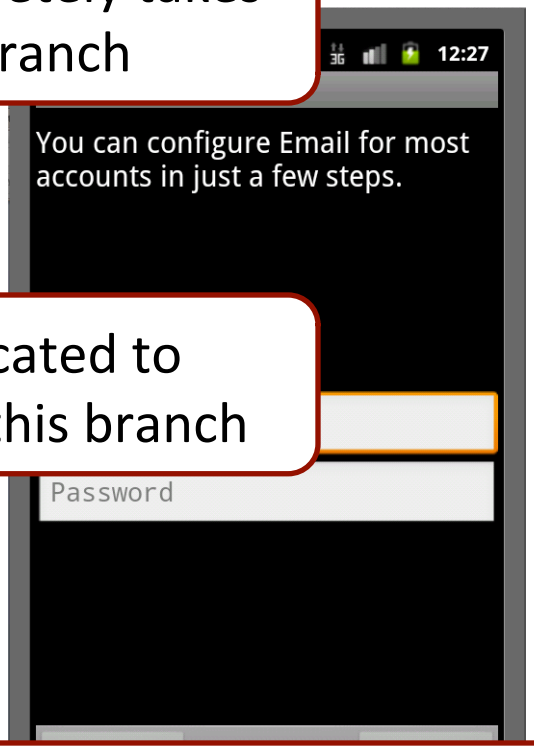


New Challenges: Mixing Static & Dynamic

```
Cursor c = query(Account.ID_PROJECTION);
int numAccounts = c.getCount();
if (numAccounts == 0)
    actionNewAccount();
else if (numAccounts == 1) {
    c.moveToPosition(0);
    long accountId = c.getLong(Account.ID_C
    actionHandleAccount(accountId);
} else
    actionShowAccounts();

public class Cursor {
    public boolean moveToPosition(int pos) {
        // Check position isn't past end of curso
        int count = getCount();
        if (pos >= count) return false;
        return true;
    }
}
```

Concretely takes this branch



Fabricated to take this branch

Symbolic state prevents taking this branch: (c.getCount() == 1 ∧ 0 >= c.getCount()) is unsat

Conclusion

- Proposed a practical system for generating relevant inputs to mobile apps
  - Satisfying the five desirable criteria we identified: robust, black-box, versatile, automated, efficient
- Showed its effectiveness on real-world apps
  - Significantly automates tasks that users consider tedious
  - Yields significantly more concise inputs than fuzz testing
  - Exposed handful of crashing bugs

Thank You!

http://pag.gatech.edu/dynodroid