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
  - Development and testing

- Security
  - Pre-deployment Certification

- Performance
  - Post-deployment Adaptation

- New software engineering problems in all stages ⇒ 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

\[ s_0 \overset{e_1}{\rightarrow} s_1 \overset{e_2}{\rightarrow} s_2 \overset{e_3}{\rightarrow} s_3 \ldots \]

• 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
  ⇒ vast majority of events are no-ops

• Insight #2: Can identify relevant events by lightly instrumenting SDK once and for all
  ⇒ Does not require instrumenting app
• 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
  ⇒ 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

<table>
<thead>
<tr>
<th>Approach</th>
<th>#Events</th>
<th>#Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynodroid - Frequency</td>
<td>2,000</td>
<td>1</td>
</tr>
<tr>
<td>Dynodroid - UniformRandom</td>
<td>2,000</td>
<td>3</td>
</tr>
<tr>
<td>Dynodroid - BiasedRandom</td>
<td>2,000</td>
<td>3</td>
</tr>
<tr>
<td>Monkey</td>
<td>10,000</td>
<td>3</td>
</tr>
<tr>
<td>Humans</td>
<td>No limit</td>
<td>&gt;= 2</td>
</tr>
</tbody>
</table>
Dynodroid vs. Monkey

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

Automation Degree = \( \frac{C(\text{Dynodroid} \cap \text{Human})}{C(\text{Human})} \)

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 System Events vs. Monkey

43% 6% 10%

common Dynodroid
w/o system Monkey

% App Code Coverage

App ID
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

<table>
<thead>
<tr>
<th>App Name</th>
<th>Bugs</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PasswordMakerProForAndroid</td>
<td>1</td>
<td>Null</td>
<td>Improper handling of user data.</td>
</tr>
<tr>
<td>com.morphoss.acal</td>
<td>1</td>
<td>Null</td>
<td>Dereferencing null returned by an online service.</td>
</tr>
<tr>
<td>hu.vsza.adsdroid</td>
<td>2</td>
<td>Null</td>
<td>Dereferencing null returned by an online service.</td>
</tr>
<tr>
<td>cri.sanity</td>
<td>1</td>
<td>Null</td>
<td>Improper handling of user data.</td>
</tr>
<tr>
<td>com.zoffcc.applications.aagtl</td>
<td>2</td>
<td>Null</td>
<td>Dereferencing null returned by an online service.</td>
</tr>
<tr>
<td>org.beide.bomber</td>
<td>1</td>
<td>Array</td>
<td>Game indexes an array with improper index.</td>
</tr>
<tr>
<td>com.addi</td>
<td>1</td>
<td>Null</td>
<td>Improper handling of user data.</td>
</tr>
</tbody>
</table>
# Bugs Found in 1,000 Google Play Apps

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

```java
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(...);
}
```
New Challenges: Mixing Static & Dynamic

```java
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();
}
```

Fabricate “target”; not used subsequently
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_CONTENT);
    actionHandleAccount(accountId);
} else
    actionShowAccounts();

public class Cursor {
    public boolean moveToPosition(int pos) {
        // Check position isn't past end of cursor
        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