APPSTORE 2.0: Improving the Quality of Mobile Apps by Leveraging the Crowds
Motivation

Growing mobile app stores

In 2013, Google Play Store
+2,000,000 apps

+50 billions
app downloads!
Motivation

Negative impact on developer and store reputation!
Motivation

Current App Stores

Developer -> app -> App Repositories

app

User
Motivation

Current App Stores

Developer → app → App Repositories → User

App

6
Problem

In-house testing

Robotium
monkeyrunner
TestDroid
GUITAR
Caipa
Dinodroid
Xamarin Test Cloud
Problem

In-house testing

Robotium
monkeyrunner
TestDroid
Caliipa
Dinodroid
Xamarin Test Cloud

In the wild
Problem

Diagnosing mobile apps is **challenging**

- **Rapid platform evolution**
- **Device fragmentation**
- **Diverse operating context**
- **Privacy issues**
Leveraging Crowds of Apps to Increase Quality

Geoffrey Hecht,
Naouel Moha (UQAM)
Development AntiPattern:

The Blob

- Symptoms
  - Single class with many attributes & operations
  - Controller class with simple, data-object classes.
  - Lack of OO design.
  - A migrated legacy design

- Consequences
  - Lost OO advantage
  - Too complex to reuse or test.
  - Expensive to load
Avoid Internal Getters/Setters

In native languages like C++ it's common practice to use getters (\texttt{i = getCount()} ) instead of accessing the field directly (\texttt{i = mCount} ). This is an excellent habit for C++ and is often practiced in other object oriented languages like C# and Java, because the compiler can usually inline the access, and if you need to restrict or debug field access you can add the code at any time.

However, this is a bad idea on Android. Virtual method calls are expensive, much more so than instance field lookups. It's reasonable to follow common object-oriented programming practices and have getters and setters in the public interface, but within a class you should always access fields directly.

Without a JIT, direct field access is about 3x faster than invoking a trivial getter. With the JIT (where direct field access is as cheap as accessing a local), direct field access is about 7x faster than invoking a trivial getter.
**Step 1: App Analysis**
- Construct model
- Extract metadata

**Step 2: Model Conversion**
- Convert entities
- Convert metrics

**Step 3: Anti-pattern Detection**
- Determine thresholds
- Execute queries

Detected anti-patterns

Graph DB
MATCH (cl:Class)
WHERE cl.lack_of_cohesion_in_methods > HIGH
    AND cl.number_of_methods > HIGH
    AND cl.number_of_attributes > HIGH
RETURN cl

MATCH (m1:Method)-[:CALLS]->(m2:Method),
    (cl:Class)
WHERE m2.is_setter OR m2.is_getter
    AND cl-[:CLASS_OWNS_METHOD]->m1
    AND cl-[:CLASS_OWNS_METHOD]->m2
RETURN m1
Stability
Constant decline

Sudden decline

Constant rise

75 versions
Major refactoring

Controlled additions

Minor refactorings

Minor refactorings

Complex Class

BLOB

LIC

75 versions

Complex Class

Major refactoring
Leveraging Crowds of Devices to Improve UI Performance

María Gómez,
Bram Adams (Poly. Montréal)
To ensure smooth interactions apps should run at 60 *Frames per Second* (fps)
Problem: How to know when janks arise?

Nexus 7 – Android 4.3

Nexus 7 – Android 5.0

Janky!
DUNE: Identifying Janks

Developer → app V1
DUNE: Identifying Janks

1. Run UI Performance Test

Developer

Context Profiles
DUNE: Identifying Janks

1. Run UI Performance Test

Developer

Context Profiles

UI Performance Repository

UI metrics

UI Events Timestamp
DUNE: Identifying Janks

1. Run UI Performance Test

2. Build Model of versions/context Without janks

Developer

Context Profiles

UI Performance Repository

UI metrics

UI Events Timestp.

app V1
DUNE: Identifying Janks

1. Run UI Performance Test

2. Build Model of versions/context Without janks

UI Performance Repository

UI metrics

UI Events Timestp.

Context Profiles

New context

Developer

New app release

New app release

app

app

app

app
DUNE: Identifying Janks

1. Run UI Performance Test

- New UI metrics
- New UI Events Timestamp

2. Build Model of versions/context
   Without janks

Developer

New app release

Context Profiles

New context

UI Performance Repository

New app release

app v1

app v2
DUNE: Identifying Janks

1. Run UI Performance Test
2. Build Model of versions/context Without janks
3. Similarity Measure
DUNE: Identifying Janks

1. Run UI Performance Test

2. Build Model of versions/context Without janks

3. Similarity Measure

4. Detect Outliers

Developer

Context Profiles

UI Performance Repository

New UI metrics

UI metrics

UI Events Timestamp

Similar test runs

28
DUNE: Identifying Janks

1. Run UI Performance Test
2. Build Model of versions/context Without janks
3. Similarity Measure
4. Detect Outliers
5. Context Patterns

Developer

UI Performance Repository

UI metrics
UI Events Timestp.
New UI metrics

Similar test runs
Perform. Outliers

Context Profiles
DUNE: Identifying Janks

1. Run UI Performance Test
2. Build Model of versions/context Without janks
3. Similarity Measure
4. Detect Outliers
5. Context Patterns

Regression!
Context: SDK 5.0
UI Event: Ev1
DUNE: Identifying Janks

1. Run UI Performance Test
2. Build Model of versions/context
   - Without janks
3. Similarity Measure
4. Detect Outliers
5. Context Patterns

Developer

UI Performance Repository

UI metrics
UI Events Timestp.

New UI metrics

Similar test runs

Perform. Outliers

Regression!
Context: SDK 5.0
UI Event: Ev1
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:

- Android device
- Daytime
- High CPU load

Graph showing time (ms) per frame and janky frames. Speed limit is 16 mspf. BUCKET1: #Fr=48, #Janky=3, Smooth=0.94. BUCKET2: #Fr=48, #Janky=0, Smooth=1. BUCKET3: #Fr=28, #Janky=1, Smooth=0.96. BUCKET4: #Fr=24, #Janky=3, Smooth=0.88.
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:

Context

Speed Limit

16 mspf

janky frames

E1

E2

E3

time (ms) per frame

28

21

14

7

0

BUCKET1

#Fr=48

#Janky=3

Smooth=0.94

BUCKET2

#Fr=48

#Janky=0

Smooth=1

BUCKET3

#Fr=28

#Janky=1

Smooth= 0.96

BUCKET4

#Fr=24

#Janky=3

Smooth= 0.88

EXIT
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:

![Android, CPU, Memory icons]

- **Speed Limit**: 16 mspf
- **time (ms) per frame**: 28 to 14
- **janky frames**: E1, E2, E3

**BUCKET1**
- #Fr=48
- #Janky=3
- Smooth=0.94

**BUCKET2**
- #Fr=48
- #Janky=0
- Smooth=1

**BUCKET3**
- #Fr=28
- #Janky=1
- Smooth=0.96

**BUCKET4**
- #Fr=24
- #Janky=3
- Smooth=0.88

**EXIT**
1. Run UI Performance Test

Example. UI metrics collected during a test run

**Test run context:**

- **Speed Limit:** 16 mspf
- **Smooth ratio:**\(\frac{\text{Smooth}}{\text{Frames}} = 1 - \frac{\text{Janky}}{\text{Frames}}\)

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Frames</th>
<th>Janky</th>
<th>Smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUCKET 1</td>
<td>Fr=48</td>
<td>3</td>
<td>0.94</td>
</tr>
<tr>
<td>BUCKET 2</td>
<td>Fr=48</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>BUCKET 3</td>
<td>Fr=28</td>
<td>1</td>
<td>0.96</td>
</tr>
<tr>
<td>BUCKET 4</td>
<td>Fr=24</td>
<td>3</td>
<td>0.88</td>
</tr>
</tbody>
</table>
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:

UI Events

Graph showing time (ms) per frame, speed limit, and janky frames for different buckets:
- **BUCKET1**: 
  - #Fr=48
  - #Janky=3
  - Smooth=0.94
- **BUCKET2**: 
  - #Fr=48
  - #Janky=0
  - Smooth=1
- **BUCKET3**: 
  - #Fr=28
  - #Janky=1
  - Smooth=0.96
- **BUCKET4**: 
  - #Fr=24
  - #Janky=3
  - Smooth=0.88
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:

- **UI EVENTS**
  - E1
  - E2
  - E3

- **UI METRICS**
  - Speed Limit: 16 mspf
  - #Fr=48
  - #Janky=3
  - Smooth=0.94

- **CONTEXT**
  - Android
  - Date
  - Device

Graph showing time (ms) per frame with janky frames highlighted and metrics collected during various bucket stages.
Example. UI metrics collected during a test run

Test run context:
1. Run UI Performance Test

Example. UI metrics collected during a test run

Test run context:
Detecting UI Performance Deviations

2. Building Model of Versions/Contexts

UI PERFORMANCE METRICS

- **E1**
  - T1: [200, 0.9, 25.0], [200, 0.8, 10.1]
  - T2: [190, 0.8, 24.1], [210, 0.9, 10.4]
  - T3: [195, 0.9, 24.1], [205, 0.9, 10.5]
  - T4: [200, 0.9, 27.1], [200, 0.8, 12.1]
  - T5: [195, 0.8, 10.1], [205, 0.8, 10.1]

- **E2**
  - T6: [160, 0.5, 10.1], [95, 0.4, 10.1]

- **E3**
  - [48, 0.9, 10.2]
  - [53, 0.9, 10.0]
  - [50, 0.9, 10.1]
  - [57, 0.9, 11.1]
  - [50, 0.5, 10.1]
2. Building Model of Versions/Contexts
Detecting UI Performance Deviations

2. Building Model of Versions/Contexts

Batch Model

<table>
<thead>
<tr>
<th>CONTEXT</th>
<th>UI PERFORMANCE METRICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1:</td>
<td>[200, 0.9, 25.0], [200, 0.8, 10.1], [48, 0.9, 10.2]</td>
</tr>
<tr>
<td>T2:</td>
<td>[190, 0.8, 24.1], [210, 0.9, 10.4], [53, 0.9, 10.0]</td>
</tr>
<tr>
<td>T3:</td>
<td>[195, 0.9, 24.1], [205, 0.9, 10.5], [50, 0.9, 10.1]</td>
</tr>
<tr>
<td>T4:</td>
<td>[200, 0.9, 27.1], [200, 0.8, 12.1], [57, 0.9, 11.1]</td>
</tr>
<tr>
<td>T5:</td>
<td>[195, 0.8, 10.1], [205, 0.8, 10.1], [30, 0.9, 10.1]</td>
</tr>
<tr>
<td>T6:</td>
<td>[160, 0.5, 10.1], [95, 0.4, 10.1], [50, 0.5, 10.1]</td>
</tr>
</tbody>
</table>
Detecting UI Performance Deviations

2. Building Model of Versions/Contexts

<table>
<thead>
<tr>
<th>UI PERFORMANCE METRICS</th>
<th>CONTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E1</strong></td>
<td>![Context Icon]</td>
</tr>
<tr>
<td>T1: [200, 0.9, 25.0]</td>
<td></td>
</tr>
<tr>
<td>T2: [190, 0.8, 24.1]</td>
<td></td>
</tr>
<tr>
<td>T3: [195, 0.9, 24.1]</td>
<td></td>
</tr>
<tr>
<td>T4: [200, 0.9, 27.1]</td>
<td></td>
</tr>
<tr>
<td>T5: [200, 0.9, 10.1]</td>
<td></td>
</tr>
<tr>
<td><strong>E2</strong></td>
<td>![Context Icon]</td>
</tr>
<tr>
<td>T6: [160, 0.5, 10.1]</td>
<td></td>
</tr>
<tr>
<td><strong>E3</strong></td>
<td>![Context Icon]</td>
</tr>
<tr>
<td>T6: [95, 0.4, 10.1]</td>
<td></td>
</tr>
<tr>
<td><strong>E3</strong></td>
<td>![Context Icon]</td>
</tr>
<tr>
<td>T6: [50, 0.5, 10.1]</td>
<td></td>
</tr>
</tbody>
</table>

Batch Model

New test run

New test run
Detecting UI Performance Deviations

3. Similarity Measure

UI PERFORMANCE METRICS

<table>
<thead>
<tr>
<th></th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>[200, 0.9, 25.0], [200, 0.8, 10.1], [48, 0.9, 10.2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>[190, 0.8, 24.1], [210, 0.9, 10.4], [53, 0.9, 10.0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>[195, 0.9, 24.1], [205, 0.9, 10.5], [50, 0.9, 10.1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>[200, 0.9, 27.1], [200, 0.8, 12.1], [57, 0.9, 11.1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>[195, 0.8, 10.1], [205, 0.8, 10.1], [30, 0.9, 10.1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>[160, 0.5, 10.1], [95, 0.4, 10.1], [50, 0.5, 10.1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTEXT

Rank previous tests with most similar context
Detecting UI Performance Deviations

3. Similarity Measure

Rank previous tests with most similar context
Detecting UI Performance Deviations

4. Performance Outlier Detection

UI PERFORMANCE METRICS

E1
T1: [200, 0.9, 25.0], [200, 0.8, 10.1], [48, 0.9, 10.2]

E2
T3: [195, 0.9, 24.1], [205, 0.9, 10.5], [50, 0.9, 10.1]
T4: [200, 0.9, 27.1], [200, 0.8, 12.1], [57, 0.9, 11.1]

E3
T6: [160, 0.5, 10.1], [95, 0.4, 10.1], [50, 0.5, 10.1]

CONTEXT

NEW TEST
Detecting UI Performance Deviations

4. Performance Outlier Detection

**UI PERFORMANCE METRICS**

- **T1:** [200, 0.9, 25.0], [200, 0.8, 10.1], [48, 0.9, 10.2]
- **T3:** [195, 0.9, 24.1], [205, 0.9, 10.5], [50, 0.9, 10.1]
- **T4:** [200, 0.9, 27.1], [200, 0.8, 12.1], [57, 0.9, 11.1]
- **T6:** [160, 0.5, 10.1], [95, 0.4, 10.1], [50, 0.5, 10.1]

**CONTEXT**

**Interquartile Range (IQR) Filtering technique**

Detect outliers and extreme values
Detecting UI Performance Deviations

4. Performance Outlier Detection

**Interquartile Range (IQR)**
Filtering technique

Detect outliers and extreme values
Detecting UI Performance Deviations

4. Performance Outlier Detection

**Interquartile Range (IQR) Filtering technique**

Detect outliers and extreme values

Outlier with +/- offset Regression/Optimization
Detecting UI Performance Deviations

4. Performance Outlier Detection

Interquartile Range (IQR) Filtering technique

Detect outliers and extreme values

Outlier with +/- offset Regression/Optimization

Flag new test as regression in event E2!
Detecting UI Performance Deviations

4. Identify Context Patterns

• Identify common context patterns that induce outliers
  • Help developers to isolate defective contexts

• Association Rule Mining (Apriori algorithm)

Examples of rules:

{sdk=5.1} => Outlier+
{v=1.2, dev=LG} => E2Outlier-
Leveraging Crowds of Users to Reproduce App Crashes

María Gómez,
Bram Adams (Poly. Montréal),
Walid Maalej (Univ. Hamburg)
Problem: How to reproduce the crash?
Problem: How to reproduce the crash?
Problem: How to reproduce the crash?
Problem: How to reproduce the crash?
Problem: How to reproduce the crash?
Problem: How to reproduce the crash?
MoTiF: Crash Reproduction
1. Collecting
Crash Traces
MoTiF: Crash Reproduction

1. Collecting Crash Traces

2. Identifying Crash Patterns
MoTiF: Crash Reproduction

1. Collecting Crash Traces

2. Identifying Crash Patterns

3. Synthesizing Crash Test Suites
MoTiF: Crash Reproduction

1. Collecting Crash Traces
2. Identifying Crash Patterns
3. Synthesizing Crash Test Suites
4. Assessing Test Suite Effectivity
MoTiF: Crash Reproduction

1. Collecting Crash Traces
2. Identifying Crash Patterns
3. Synthesizing Crash Test Suites
4. Assessing Test Suite Effectivity
2. Identifying Crash Patterns

Crowd Crash Graph
2. Identifying Crash Patterns

**Crowd Crash Graph**

[Diagram showing a network of events (e1, e2, e4, e5) connected to a cloud symbolizing data processing. The network includes a table showing sensor data (wifi, data, mem, bat) with states (ON, OFF, LOW, HIGH). There are nodes representing events with probabilities (0.66, 0.75, 0.25, 0.33) and a final node (c1) indicating a crash pattern.]
2. Identifying Crash Patterns

Crowd Crash Graph

Markov probability

Crowd Crash Graph
2. Identifying Crash Patterns

Crowd Crash Graph

Markov probability

Context
2. Identifying Crash Patterns

Crowd Crash Graph

Synthesize steps to reproduce crashes
2. Identifying Crash Patterns

Crowd Crash Graph

Synthesize steps to reproduce crashes

Shortest path with highest probability

CROWD CONSOLIDATED TRACE!!
2. Identifying Crash Patterns

Learning crash-prone contexts

e1 → e2 → c1

<table>
<thead>
<tr>
<th>wifi</th>
<th>data</th>
<th>mem</th>
<th>bat</th>
<th>wifi</th>
<th>data</th>
<th>mem</th>
<th>bat</th>
<th>wifi</th>
<th>data</th>
<th>mem</th>
<th>bat</th>
<th>sdk</th>
<th>model</th>
<th>manuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>OFF</td>
<td>LOW</td>
<td>HIGH</td>
<td>OFF</td>
<td>LOW</td>
<td>HIGH</td>
<td></td>
<td>OFF</td>
<td>LOW</td>
<td>HIGH</td>
<td></td>
<td>4.1</td>
<td>LG60</td>
<td>LG</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>HIGH</td>
<td>LOW</td>
<td>OFF</td>
<td>HIGH</td>
<td>LOW</td>
<td></td>
<td>OFF</td>
<td>HIGH</td>
<td>LOW</td>
<td></td>
<td>4.1</td>
<td>G3</td>
<td>LG</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>HIGH</td>
<td>HIGH</td>
<td>ON</td>
<td>OFF</td>
<td>HIGH</td>
<td>HIGH</td>
<td>OFF</td>
<td>OFF</td>
<td>HIGH</td>
<td>HIGH</td>
<td>4.1</td>
<td>G3</td>
<td>LG</td>
</tr>
</tbody>
</table>
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

\[ e_1 \rightarrow e_2 \rightarrow c_1 \]
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

e1 -> e2 -> c1
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

Frequent Closed Sequential Pattern Mining (100% support)
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

Frequent Closed Sequential Pattern Mining (100% support)
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

e1 → e2 → c1

Static context

Frequent Closed Sequential Pattern Mining (100% support)
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

Static context

Frequent Closed Sequential Pattern Mining (100% support)
2. Identifying Crash Patterns

Learning crash-prone contexts

Dynamic context

Static context

Frequent Closed Sequential Pattern Mining (100% support)

Union
3. Synthesizing Crash Test Suites

**Consolidated Trace**

- $e_1$ → $e_2$ → $c_1$

**Dynamic Context**

- $\text{wifi}$: ON/OFF
- $\text{data}$: OFF/OFF

**Static Context**

- $\text{sdk}$: 4.1
- $\text{model}$: LG60
- $\text{manuf}$: G3
- $\text{manuf}$: LG
3. Synthesizing Crash Test Suites

Consolidated Trace

Dynamic Context
wifi data wifi data
ON OFF OFF OFF

Static Context
sdk model manuf
4.1 LG60 G3 LG

Synthesize Crash Test

Mapping Rules

<table>
<thead>
<tr>
<th>Button</th>
<th>onClick</th>
<th>clickOnButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>onOrientationChange</td>
<td>setOrientationActivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
3. Synthesizing Crash Test Suites

Consolidated Trace

Dynamic Context

Static Context

Synthesize Crash Test

Mapping Rules

Robotium Black-box test

```java
public void testRun(){
    solo.sleep(2000);
    solo.clickOnView(solo.getView(215));
    solo.sleep(2000);
    solo.setWifiData(false);
    solo.setMobileData(false);
    solo.clickOnMenuItem("Save Page");
}
```

<table>
<thead>
<tr>
<th>Button</th>
<th>onClick</th>
<th>clickOnButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>onOrientationChange</td>
<td>setOrientationActivity</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CrowdSeer: Crash Prevention

1. Collecting Crash Traces
2. Identifying Crash Patterns
3. Synthesizing Crash Test Suites
4. Assessing Test Suite Effectivity
CrowdSeer: Crash Prevention

1. Collecting Crash Traces
2. Identifying Crash Patterns
3. Synthesizing Crash Test Suites
4. Assessing Test Suite Effectivity
5. Synthesizing Preventive patch
developers
CrowdSeer: Crash Prevention

1. Collecting Crash Traces
2. Identifying Crash Patterns
3. Synthesizing Crash Test Suites
4. Assessing Test Suite Effectivity
5. Synthesizing Preventive patch
6. Hot-patching

Crash-triggering feature to disable
Today's featured article

Myles Standish (c. 1584 – 1656) was an English military officer hired by the Pilgrims as military advisor for the Plymouth Colony. One of the Mayflower passengers, Standish played a leading role in the administration and defense of the colony from its inception. On February 17, 1621, the colony militia elected him as its first commander and continued to re-elect him to that position for the remainder of his life. He served as an agent of Plymouth Colony in England, and as assistant governor and treasurer of the colony. He was also one of the first settlers and founders of the town of Duxbury, Massachusetts. As a military leader, Standish favored preemptive action, sometimes angering Native Americans and disturbing more moderate members of the colony. By the 1640s, he relinquished his role as an active soldier and settled into a quieter
CrowdSeer: Crash Prevention
CrowdSeer: Crash Prevention

Today
Read in other languages
Save page
Share

Standish played a leading role in the administration and defense of the colony from its inception. On February 17, 1621, the colony militia elected him as its first commander and continued to re-elect him to that position for the remainder of his life. He served as an agent of Plymouth Colony in England, and as assistant governor and treasurer of the colony. He was also one of the first settlers and founders of the town of Duxbury, Massachusetts. As a military leader, Standish favored preemptive action, sometimes angering Native Americans and disturbing more moderate members of the colony. By the 1640s, he relinquished his role as an active soldier and settled into a quieter
CrowdSeer: Crash Prevention
CrowdSeer: Crash Prevention
CrowdSeer: Crash Prevention
CrowdSeer: Crash Prevention
CrowdSeer: Crash Prevention
Validations

• Lots of interesting insights in the papers based on
  • More than 3,000 apps (some including >100 versions)
  • More than 100 devices
  • More than 500 users (labs, interviews, crowdsourcing)
Questions?

1. M. Gómez, M. Martinez, M. Monperrus, R. Rouvoy: *When App Stores Listen to the Crowd to Fight Bugs in the Wild*. ICSE-NIER’15


3. G. Hecht, O. Benomar, R. Rouvoy, N. Moha, L. Duchien: *Tracking the Software Quality of Android Applications Along Their Evolution*. ASE’15

