Large-scale Analysis of Framework-specific Exceptions in Android Apps

Lingling Fan, Ting Su, Sen Chen, Guozhu Meng, Yang Liu, Lihua Xu, Geguang Pu, Zhendong Su

ACM SIGSOFT Distinguished Paper Award

ICSE 2018
Gothenburg, Sweden
Mobile app is continuously increasing

One of the key priority is to prevent fail-stop errors, e.g., crash

[statista.com]
However…

Apps still suffer from crashes.
Customer complaints

A Lee-Koo
⭐⭐⭐⭐⭐ 30 May 2018
Keeps crashing on Android 9 Developer Preview 2, please fix

walter itsadavisthang Youwouldntunderstand
⭐⭐⭐⭐⭐ 25 May 2018
Keeps crashing, works shotty!

Brittonee Deleveaux
⭐⭐⭐⭐⭐ 27 May 2018
Good for socializing with friends and family but becoming too strict and bias on behavior standards. App now crashes a lot too. Some people are targeted to be banned for 30 days and some are not penalized for the same infraction. Bans cannot be challenged and Face- book carelessly blocks you from
Framework-specific Crash for Android Apps

java.lang.RuntimeException: Unable to resume activity {*}: java.lang.NumberFormatException: Invalid double: ""
  at android.app.ActivityThread.performResumeActivity(…)
  ....
Caused by: java.lang.NumberFormatException: Invalid double:"
  at java.lang.StringToReal.invalidReal(StringToReal.java:63)
  at java.lang.StringToReal.parseDouble(StringToReal.java:248)
  ....

Root exception

Crash signaler

An example of exception trace

Crash Signaler ➤ Framework ➤ App ➤ Lib

NOTE: We do not consider exceptions caused by the bugs of framework itself.
With the understanding of framework crashes

Developers: avoid and fix crashes
Researchers: improve bug detection tools

However, existing studies on functional bugs analysis:

• Small scale (AST’11, ICST’14)
• Different goals (ICST’14, MSR’15)
  (1) generate testing oracles
  (2) investigate bug hazards of exception-handling code
Analyzing (framework-specific) crashes is challenging

- **Lack of comprehensive dataset**
  - No publicly available data
  - Only 16% issues contain exception traces on Github and Google Code

- **Lack of tool support**
  - Crash reproducing tools
  - Failure localization tools

- **Substantial human effort**
  - Require understanding of Android framework
Contribution

- First empirical study to characterize Android framework-specific exceptions
  - 11 fault categories
- Evaluate the state-of-the-art bug detection techniques
  - Static & dynamic tools
- Prototype tools to demonstrate the usefulness of findings
  - Stoat+ & Exlocator
- Publicly available dataset
Data Collection

- **F-Droid** (Github & Google code apps)
- **Crawling**
  - 6,588 unique traces from issues and comments of 583 apps
  - 613 fixing commits
- **Testing**
  - Monkey
  - Sapienz
  - Stoat

Each app runs for 3h 4 months in total
- **2174**
- **9,722** unique traces
Research Questions

RQ1: Are framework exceptions recurring?

RQ2: Fault Patterns?

RQ3: Detected by current techniques?

RQ4: Fixing patterns and effort?
RQ1: Are framework exceptions recurring?

Based on 6,588 unique exceptions from Github and Google code

Yes, framework exceptions are more recurring and pervasive
RQ2: Taxonomy of Framework Exceptions

**Bucket**: repository for exceptions that are thrown from the same location of Android framework.

Top 200 buckets occupy over 80% exceptions

Framework exceptions

2,016
RQ2: Taxonomy of Framework Exceptions

**Bucket**: repository for exceptions that are thrown from the same location of Android framework

11 fault categories covering 84.6% of all framework exceptions
## RQ2: Taxonomy of Framework Exceptions

<table>
<thead>
<tr>
<th>Category</th>
<th>Occurrence</th>
<th>#S.O. posts</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Updates and Compatibility</td>
<td>68</td>
<td>60</td>
</tr>
<tr>
<td>XML Layout Error</td>
<td>122</td>
<td>246</td>
</tr>
<tr>
<td>API Parameter Error</td>
<td>820</td>
<td>819</td>
</tr>
<tr>
<td><strong>Framework Constraint Error</strong></td>
<td><strong>383</strong></td>
<td><strong>1726</strong></td>
</tr>
<tr>
<td>Index Error</td>
<td>950</td>
<td>218</td>
</tr>
<tr>
<td>Database Management Error</td>
<td>128</td>
<td>61</td>
</tr>
<tr>
<td>Resource-Not-Found Error</td>
<td>1303</td>
<td>7178</td>
</tr>
<tr>
<td>UI Update Error</td>
<td>327</td>
<td>666</td>
</tr>
<tr>
<td>Concurrency Error</td>
<td>372</td>
<td>263</td>
</tr>
<tr>
<td><strong>Component Lifecycle Error</strong></td>
<td><strong>608</strong></td>
<td><strong>1065</strong></td>
</tr>
<tr>
<td>Memory/Hardware Error</td>
<td>414</td>
<td>792</td>
</tr>
</tbody>
</table>

Developers make more mistakes on Lifecycle Error, Framework Constraint Error and Memory/Hardware Error.
RQ3: Auditing bug detection tools

![75 different exception instances from 11 categories]

<table>
<thead>
<tr>
<th>Tools</th>
<th>Android support</th>
<th># Detected (out of 75 exceptions)</th>
<th># Rules for Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lint</td>
<td>✔️</td>
<td>4</td>
<td>281</td>
</tr>
<tr>
<td>FindBugs</td>
<td>✔️</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PMD</td>
<td>✔️</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>SonarQube</td>
<td>✔️</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Existing static analysis tools are *ineffective* in detecting framework exceptions
RQ3: Auditing bug detection tools

- 2104 apps (4560 versions)
- Each runs for 3h

<table>
<thead>
<tr>
<th>Tools</th>
<th>Approach</th>
<th># unique framework exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monkey</td>
<td>Random</td>
<td>1842</td>
</tr>
<tr>
<td>Sapienz</td>
<td>Search-based</td>
<td>2342</td>
</tr>
<tr>
<td>Stoat</td>
<td>Model-based</td>
<td>1438</td>
</tr>
</tbody>
</table>

Metrics: Detection time & Occurrence
RQ3: Auditing bug detection tools

Detection time: The time of detecting an exception for the first time

Occurrence: The times of an exception detected during 3 hours

- Dynamic testing tools are still far from effective in detecting database, framework constraint and concurrency errors
RQ4: Fixing Patterns

1. Refine Conditional Checks

```java
+ if(...){
    ....
+ }
```

2. Move Code into Correct Thread

Worker thread  Main thread

UI update  UI update

3. Work in Right Callbacks

onCreate()  onstop()  onStart()  onStop()  
register  unregister  register  unregister

4. Adjust Implementation Choices

Code refactoring
RQ4: Fixing Efforts

- Issue duration: The time cost to fix the issue (day)
- Changed line: Exclude “//...”, “@Override”, “import *::*”
- Closing rate: The percent of issues being closed

<table>
<thead>
<tr>
<th>Category</th>
<th>Closing Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Updates and Compatibility</td>
<td>93.9%</td>
</tr>
<tr>
<td>XML Layout Error</td>
<td>93.2%</td>
</tr>
<tr>
<td>API Parameter Error</td>
<td>88.5%</td>
</tr>
<tr>
<td>Framework Constraint Error</td>
<td>87.7%</td>
</tr>
<tr>
<td>Index Error</td>
<td>84.1%</td>
</tr>
<tr>
<td>Database Management Error</td>
<td>76.8%</td>
</tr>
<tr>
<td>Resource-Not-Found Error</td>
<td>75.3%</td>
</tr>
<tr>
<td>UI Update Error</td>
<td>75.0%</td>
</tr>
<tr>
<td>Concurrency Error</td>
<td>73.5%</td>
</tr>
<tr>
<td>Component Lifecycle Error</td>
<td>58.8%</td>
</tr>
<tr>
<td>Memory/Hardware Error</td>
<td>51.6%</td>
</tr>
</tbody>
</table>

- Lifecycle, Concurrency, UI update and memory errors are more difficult to fix
Applications

(1) Improving Bug Detection

- Meaningful corner cases
  - e.g., “” and “%”

- Enforce environment interplay
  - Screen rotation
  - Start an activity and quickly back
  - Put the app at background for a long time and navigate to it again

3 previously unknown bugs

- Parameter error
- UI update error
- Lifecycle error

https://github.com/tingsu/Stoat
Applications

(2) Enabling Exception Localization

```
java.lang.RuntimeException: Unable to resume activity '{*}':
  java.lang.NumberFormatException: Invalid double: ""
  at android.app.ActivityThread.performResumeActivity(…)
  …
Caused by: java.lang.NumberFormatException: Invalid double:"
  at java.lang.StringToReal.invalidReal(StringToReal.java:63)
  at java.lang.StringToReal.parseDouble(StringToReal.java:248)
  …
```

Report
1. Fault category
2. Root cause
3. Explanations
4. Fixing solutions

- Built on Soot
- Implemented 5 categories

25 out of 27 exceptions (92% precision) are correctly located by comparing the patches from the developers.
Conclusions

RQ1: Are framework exceptions recurring?
  Yes!

RQ2: Fault Patterns?
  Lifecycle Error, Constraint Error…

RQ3: Detected by current techniques?
  Need improvement

RQ4: Fixing practices and effort?
  Lifecycle, Concurrency, UI update

➢ First large-scale analysis of Android framework-specific exceptions
➢ Supporting follow-up research on bug detection, fault localization and patch generation
➢ Large-scale and reusable dataset available on https://crashanalysis.github.io/Dataset-CrashAnalysis
NTU is still hiring PhDs and post docs
Contact: ecnujanefan@gmail.com
Homepage: http://www.sqslab.com/llfan/