Predicting Crashing Releases of Mobile Applications

Xin Xia, Emad Shihab, Yasutaka Kamei, David Lo, Xinyu Wang

eshihab@cse.concordia.ca
das.encs.concordia.ca
Mobile Applications are On the Rise

2 Million apps in Apple’s app store

Mobile Apps are Different

Mobile apps have many releases...

...driven by two key factors:

• Competition
• App stores
Mobile App Crashes are a Challenge

Mobile users are frequently and negatively impacted by app crashes

Short release cycles impact testing schedules for developers
Our Goal is to Predict Crashing Releases

RQ1. Can we effectively predict crashing releases?

RQ2. What are the best indicators of these crashing releases?
How Can we Predict Crashing Releases?

We derive metrics using development history to predict crashing releases.
Approach
Approach Overview

- Mine Apps (F-droid)
- Extract & label crashing releases
- Extract factors to predict crashing releases
- Predict & evaluate performance
Mobile App Dataset

Mined 900 mobile apps from F-droid
466 of these use Git
22 apps are active & have 2+ yrs of dev history
10 have more than 100 releases

For each app, we collect:
1. Source code
2. Repository meta-data
3. Wiki page
Determining Crashing Releases

Determine all releases from manifest

Determine releases & release dates

Mine & group commits of the specific releases
Determining Crashing Releases

Search commits logs to determine crashing fixes

Mark releases as crash-inducing releases

Manually examine (flagged) crashing releases
## Determining Crashing Releases

<table>
<thead>
<tr>
<th>Project</th>
<th># Releases</th>
<th># Crashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>597</td>
<td>97</td>
</tr>
<tr>
<td>App2</td>
<td>149</td>
<td>23</td>
</tr>
<tr>
<td>App3</td>
<td>156</td>
<td>28</td>
</tr>
<tr>
<td>App4</td>
<td>392</td>
<td>19</td>
</tr>
<tr>
<td>App5</td>
<td>230</td>
<td>36</td>
</tr>
<tr>
<td>App6</td>
<td>233</td>
<td>26</td>
</tr>
<tr>
<td>App7</td>
<td>262</td>
<td>34</td>
</tr>
<tr>
<td>App8</td>
<td>241</td>
<td>34</td>
</tr>
<tr>
<td>App9</td>
<td>205</td>
<td>39</td>
</tr>
<tr>
<td>App10</td>
<td>123</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2,638</strong></td>
<td><strong>344</strong></td>
</tr>
</tbody>
</table>
Factors Used in Prediction

**Code & Complexity**
- Lines added
- Lines del.
- Cyclomatic comp.
- Rel. size
- No. files (curr & prev) release

**Diffusion**
- No. subsystems modified
- Entropy of modified files
- Entropy of code churn

**Commit**
- No. of commits
- No. bug fixing commits

**Time & Text**
- Days since last rel.
- Fuzzy and Naïve Bayes scores of commit messages from prior release
Evaluating Prediction Models

Actually Crashing

Recall: “How small is FN”

Precision: “How small is FP”

F-measure: Harmonic mean of Precision and Recall
Empirical Results
RQ1. How well can we predict crashing release?

*Evaluated on a 100 times 10-fold cross validation.
RQ2. What are the best indicators of a crashing release?

<table>
<thead>
<tr>
<th>Project</th>
<th>Factor 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>Text</td>
</tr>
<tr>
<td>App2</td>
<td>Text</td>
</tr>
<tr>
<td>App3</td>
<td>Text</td>
</tr>
<tr>
<td>App4</td>
<td>Text</td>
</tr>
<tr>
<td>App8</td>
<td>Text</td>
</tr>
<tr>
<td>App9</td>
<td>Code &amp; Comp</td>
</tr>
<tr>
<td>App10</td>
<td>Text</td>
</tr>
</tbody>
</table>

**Text and code & complexity factors are the best indicators of crashing releases**
Longitudinal Analysis

Train on the **earliest 70%** of the data and test on the last 30%
Majority of developers indicated that the factors and machine learning technique are practical

“we can easily extract these factors from commits in a release...”

“given that releases tend to be small, inspecting an entire release is not impossible...”

“although F1-score is not high, recall score is good, in practice we are interested to find all the crash releases as possible, thus recall is more important...”
Limitations

• Our approach is **heavily dependent on commit logs**

• Examine what **textual features best indicates crashing releases**

• **Improve prediction accuracy**, possibly by adding more/better factors

• **Replicate** on more applications
Mobile App Crashes are a Challenge

Mobile users are **frequently and negatively impacted** by app crashes

Short release cycles impact testing schedules for developers

---

**RQ1. How well can we predict crashing release?**

![Graph showing improvement in F-measure and AUC](image)

*We improve avg. F-measure by 50% and AUC by 28%*

---

**RQ2. What are the best indicators of a crashing release?**

<table>
<thead>
<tr>
<th>Project</th>
<th>Factor 1</th>
<th>Factor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>Text</td>
<td>Complexity</td>
</tr>
<tr>
<td>App2</td>
<td>Text</td>
<td>Complexity</td>
</tr>
<tr>
<td>App3</td>
<td>Text</td>
<td>Code</td>
</tr>
<tr>
<td>App7</td>
<td>Text</td>
<td>Code</td>
</tr>
<tr>
<td>App8</td>
<td>Text</td>
<td>Code</td>
</tr>
<tr>
<td>App9</td>
<td>Code</td>
<td>Text</td>
</tr>
<tr>
<td>App10</td>
<td>Text</td>
<td>Text</td>
</tr>
</tbody>
</table>

*Text, code and complexity factors are the best indicators of crashing releases*

*Evaluated on a 100 times 10-fold cross validation.*