Empir Software Eng (2017) 22:505–546
DOI 10.1007/s10664-016-9435-7

An empirical study of emergency updates for top android mobile apps

Safwat Hassan[1] · **Weiyi Shang**[2] · **Ahmed E. Hassan**[1]

Published online: 25 June 2016

Abstract The mobile app market continues to grow at a tremendous rate. The market provides a convenient and efficient distribution mechanism for updating apps. App developers continuously leverage such mechanism to update their apps at a rapid pace. The mechanism is ideal for publishing emergency updates (i.e., updates that are published soon after the previous update). In this paper, we study such emergency updates in the Google Play Store. Examining more than 44,000 updates of over 10,000 mobile apps in the Google Play Store, we identify 1,000 emergency updates. By studying the characteristics of such emergency updates, we find that the emergency updates often have a long lifetime (i.e., they are rarely followed by another emergency update). Updates preceding emergency updates often receive a higher ratio of negative reviews than the emergency updates. However, the release notes of emergency updates rarely indicate the rationale for such updates. Hence, we manually investigate the binary changes of several of these emergency updates. We find eight patterns of emergency updates. We categorize these eight patterns along two categories "Updates due to deployment issues" and "Updates due to source code changes". We find that these identified patterns of emergency updates are often associated with simple

Communicated by: Denys Poshyvanyk

✉ Safwat Hassan
shassan@cs.queensu.ca

Weiyi Shang
shang@encs.concordia.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

[1] Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Kingston, Ontario, Canada

[2] Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada

mistakes, such as using a wrong resource folder (e.g., images or sounds) for an app. We manually examine each pattern and document its causes and impact on the user experience. App developers should carefully avoid these patterns in order to improve the user experience.

Keywords Android mobile apps · Emergency updates · Patterns · SDK version · Permissions · Empirical study · Software engineering

1 Introduction

The mobile app market is continuously growing and evolving. Research on mobile app markets reports that in July 2015 more than 3.9 million mobile apps are available for users across the different mobile app stores (Statista 2016a). There exist more than 1.2 billion mobile app users worldwide and the number of users continues to grow at a very fast pace (mobiThinking 2016). ABI research estimates that mobile users downloaded 70 billion apps in 2013 (ABI Research 2016).

Mobile app stores, such as the Google Play Store, provide a unique distribution mechanism to facilitate the release and deployment of app updates. When a developer publishes an update for their app, all the current users of the app can automatically receive the update within the same day (Google Support 2016). Developers extensively leverage this low cost distribution mechanism in order to rapidly publish updates. Such a distribution mechanism eases the shift towards faster update cycles (Khomh et al. 2012; Mäntylä et al. 2013). However, frequent updates may disturb users, such that some corporations (like Microsoft (Wikipedia 2016)) opted to reduce the frequency of their updates based on user feedback.

The distribution mechanism enables the rapid release and deployment of emergency updates for mobile apps. Emergency updates are updates that are released soon after the previous update. For example, the *"OPM Alert"*[1] app has an update on February 4th 2014 and an emergency update on the following day (February 5th 2014).

However, to the best of our knowledge, there exist no studies that explore such emergency updates. In this paper, we perform an empirical study on the emergency updates for the top apps in the Google Play Store. Our study focuses on the top 12,000 free-to-download apps. The top apps list is according to the Distimo's top popular apps report in 2013 (Distimo 2013). These top free-to-download apps are distributed among 25 different categories. We choose to study these apps, since updates to these top apps impact a large number of users. Moreover, such mature apps are less likely to exhibit very frequent updates. Hence, we can easily identify emergency updates. We rank the emergency nature of each update by measuring the ratio of the lifetime of its preceding update versus the median lifetime of an update for that particular app (we call this metric, the emergency ratio of an update).

We study the top 1,000 emergency updates (according to our aforementioned emergency ratio metric). Our study of the characteristics of the top 1,000 emergency updates, shows that emergency updates often have a long lifetime (i.e., they are rarely followed by another emergency update). The lifetime of an emergency update is on average 2.25 times longer than the median lifetime of all non-emergency updates of an app. Hence, users should

[1] https://play.google.com/store/apps/details?id=gov.opm.status

install these emergency updates and not worry about another emergency update following the emergency update. In addition, we find that developers rarely mention the reasons of emergency updates in their release notes. 63.4 % of the emergency updates do not include any useful information about the rationale for the updates in their release notes. We find that the ratio of negative reviews for the update preceding an emergency update is often higher than the ratio for the emergency update.

To further understand emergency updates, we manually examine the decompiled code and files in the binaries (apk files) that are associated with such emergency updates and their preceding updates. Our manual analysis of 361 emergency updates leads us to identify several common patterns for emergency updates. We document eight patterns of emergency updates. These patterns belong to two categories: 1) Updates due to deployment issues and 2) Updates due to source code changes. We document the details of each pattern with its root-causes, example updates, speed of repair, examples of user complaints and lessons learned from the pattern.

The contributions of this paper are as follows:

1. This paper is the first study to empirically examine the characteristics and patterns of emergency updates for mobile apps.
2. Our detailed documentation of emergency updates can help mobile app developers avoid these patterns before releasing updates for their mobile apps.

Our work is a first step towards documenting such patterns and we expect that future studies will extend these patterns and uncover new ones.

The rest of this paper is organized as follows. Section 2 describes the studied apps and illustrates our data collection process and study methodology. Section 3 discusses the characteristics of emergency updates and Section 4 defines our approach for identifying patterns of emergency updates. Section 5 describes the identified patterns for emergency updates. Section 6 outlines the limitations and threats to the validity of our study. Section 7 describes the related work. Finally, Section 8 concludes our study.

2 Methodology

In this section, we describe the methodology of our study. First, we collect apps from the Google Play Store. Then, we identify emergency updates from the collected apps. Figure 1 illustrates an overview of the methodology of our study.

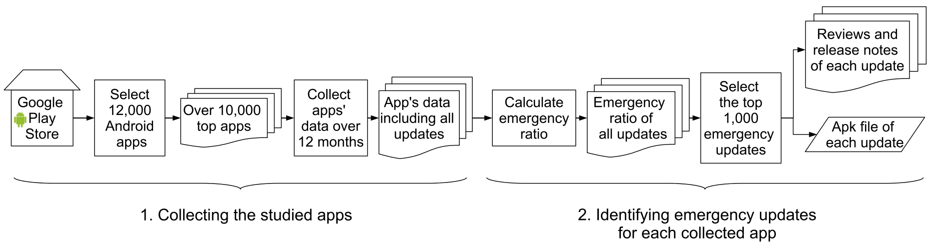
Google Play Store
Select 12,000 Android apps
Over 10,000 top apps
Collect apps' data over 12 months
App's data including all updates
Calculate emergency ratio
Emergency ratio of all updates
Select the top 1,000 emergency updates
Reviews and release notes of each update
Apk file of each update
1. Collecting the studied apps
2. Identifying emergency updates for each collected app

Fig. 1 An overview of the methodology of our study

2.1 Collecting the Studied Apps

We study the top free-to-download mobile apps in the Google Play Store. Google Play Store is one of the world's largest mobile app stores with millions of apps and billions of downloads (Statista 2016a, b). We study free-to-download apps because the majority of the apps in the Google Play Store are free-to-download (AppBrain 2016). In addition, we can only download binary files (apk file) from such apps due to our limited budget. We focus on the most popular free-to-download apps since these apps have many users and contain more updates than the less popular apps. We select the top free-to-download mobile apps using the Distimo's top popular apps report in 2013 (Distimo 2013). Distimo's report provides a list of the 400 top apps for each of the 24 non-Game categories. In addition, Distimo's report provides a list of 400 top apps for each of the 6 Game categories. In total, we collect 12,000 top free-to-download apps. We use the Distimo's published popular apps ranking in 2013 (one year before we start our data collection) in order to ensure that the apps are more stable and that the collected updates are not early updates of an app (such early updates are likely to be rapid in nature).

In order to collect daily data about the studied apps, we crawl the Google Play Store using a specialized store crawling library (Akdeniz 2013). We collect the general information of an app on a daily basis, such as its name, category, uploaded apk file, and user reviews. We select the Samsung S3 as the mobile device to download apps from the Google Play Store, as Samsung S3 is a very popular mobile device (at the time of our study). If the crawler interacts with the Google Play Store frequently, the crawler may be blocked by the Store due to too many requests. Therefore, we use a timer to pause the crawler periodically and visit the store page of a particular app only once a day.

The crawler runs during the study period of around 12 months starting from November 26th 2013 to November 18th 2014. During the study period, some apps were removed from the Google Play Store. Hence, we only collect data for the 10,747 available top apps. During the study period, we collect the apk files and user reviews for 44,113 updates. On average, each app has 4.11 updates during the study period and 6,894 apps published at least one update during the study period.

2.2 Identifying Emergency Updates

In order to identify emergency updates, for each update U_i, we calculate the *lifetime of the update (U_i)*, as the time difference (in days) between the update date of the update U_i and the update date of the next update U_{i+1}.

In order to identify emergency updates, intuitively we consider an update U_i as an emergency update, if the lifetime of the update U_{i-1} is less than one day. However, some apps have a more frequent update cycle than other apps. For example, The *"Hola Free VPN"*[2] app has 69 updates during the study period (the median update lifetime for this app is three days), while the *"Stock Watch: BSE / NSE"*[3] app has only three updates during the study period (the median update lifetime for this app is 202 days). There are 200 apps that have a median update lifetime of less than a week.

[2]https://play.google.com/store/apps/details?id=org.hola

[3]https://play.google.com/store/apps/details?id=com.snapwork.finance

Table 1 Mean and five-number summary of emergency ratio of the top 1,000 emergency updates. The larger the emergency ratio, the longer the lifetime of the update preceding the emergency update

Update category	Mean	Min.	1st Qu.	Median	3rd Qu.	Max.
The emergency update (U_i)	0.03	0.00	0.02	0.03	0.04	0.05

Therefore, we cannot use one value as a threshold to determine whether an update is an emergency update. Instead, we quantify the emergency nature of an update using a metric named the *Emergency ratio* of an update. We define the *Emergency ratio* of an update U_i as the ratio between the lifetime of the U_{i-1} update and the median lifetime of all the updates for that app.

$$Emergency\ ratio\ (U_i) = \frac{lifetime\ (U_{i-1})}{Median\ lifetime\ of\ the\ updates\ for\ that\ app} \tag{1}$$

The lower the emergency ratio, the higher the emergency nature of a particular update. We rank the updates by their emergency ratio and we focus on the top 1,000 emergency updates with the lowest emergency ratios. Table 1 represents the mean and five-number summary of emergency ratio of the top 1,000 emergency updates. For example, the *"Tweakker APN INTERNET MMS"*[4] app has a median update lifetime of 124 days and the lifetime of one update (*"version 1.8.1"*) is only one day (i.e., the emergency ratio for the following update is 0.008). Such a low emergency ratio indicates that the following update is very likely an emergency update.

Table 2 represents the mean and five-number summary for lifetime (in days) of the updates that precede the 1,000 top most updates according to their emergency ratio. We notice that 688 of the updates are followed by an emergency update within one day and a large portion of updates are followed with an emergency update within two days.

3 Characteristics of Emergency Updates

In this section, we study the characteristics of the top 1,000 emergency updates. In particular, we focus on five aspects of emergency updates: their lifetime, the content of their release notes, their rating and their version numbering. Table 3 summarizes the major findings and implications of our findings.

3.1 Lifetime of Emergency Updates

We would like to examine whether emergency updates last for a long time, and whether emergency updates are likely to be followed by additional emergency updates. First, we compare the lifetime of emergency updates and the median lifetime of an update of an app. We then study whether there is a statistically significant difference between the emergency ratios of the update that follows an emergency update and the update that follows a non-emergency update. We use the *MannWhitney U test* (Wilcoxon rank-sum test) (Gehan

[4] http://tweakker.com, the app was available during the study period but the Google Play Store no longer hosts this app at the time of the writing of this paper.

1965), since the emergency ratio is highly skewed. The *MannWhitney U test* is a non-parametric test which does not have any assumptions about the distribution of the sample population. A p-value of ≤ 0.05 means that the difference between the emergency ratios of the updates that follow an emergency update and the emergency ratios of the updates that follow a non-emergency update is statistically significant and we may reject the null hypothesis. By rejecting the null hypothesis, we can accept the alternative hypothesis, which shows that there is a statistically significant difference between the emergency ratios of the updates that follow an emergency update and the emergency ratios of the updates that follow a non-emergency update.

Emergency Updates Have a Long Lifetime We find that that on average the lifetime of an emergency update is 2.4 times the median lifetime of all non-emergency updates of an app. The p-value of the *MannWhitney U test* Test is 0.01271, (i.e., less than 0.05) which shows that there is a statistically significant difference between the emergency ratios of the updates that follow an emergency update and the emergency ratios of the updates that follow a non-emergency update.

Nevertheless, we find 40 emergency updates (out of the 1,000 studied emergency updates) that are followed by another emergency update. We find that the minimum value of the emergency ratio of the update after the emergency updates is only 0.01. For example, *"Horoscope HD Free"*[5] app has an update on March 30th 2014, the development team published an emergency update on the next day (March 31st 2014). However, the development team forgot to add the needed permissions for the modified code. Therefore, they publish yet another emergency update on the next day on April 1st 2014 to add the missing permissions.

Our results suggest that users should update their mobile apps if an emergency update is published as developers rarely follow an emergency update with yet another one.

3.2 Release Notes of Emergency Updates

We would like to better understand the rationale for emergency updates and whether developers do inform their users about the rationale for such emergency updates. We read the release notes for all 1,000 identified emergency updates (U_i) and the updates preceding the emergency updates (U_{i-1}). Then we identify the differences in the release notes between an emergency update (U_i) and the preceding update (U_{i-1}). Based on the differences between the two release notes, we determine whether the release note provides useful information about the rationale for the emergency update.

Around a Third of the Updates Explain the Rationale for the Emergency Update in the Release Notes We find that 36.6 % of the release notes explain what is fixed in the emergency update. For example, the *"Body Fat Calculator"*[6] app has an update on October 10th 2014 and an emergency update on the next day. The emergency update added text in the release notes that describes the update as follows: *"Version 3.2.1: Revised on-line guide did not load properly on some old devices. Issue Fixed"*.

[5] https://play.google.com/store/apps/details?id=ch.smalltech.horoscope.free

[6] https://play.google.com/store/apps/details?id=com.fullquieting.android.FatCalc

Table 2 Mean and five-number summary for lifetime (in days) of the top 1,000 updates (according to the emergency ratio of their following update)

Update category	Mean	Min.	1st Qu.	Median	3rd Qu.	Max.
The lifetime of the update preceding the emergency update (U_{i-1})	1.79	1.00	1.00	1.00	2.00	21.00

Two Third of the Updates do not Have a Clear Description Regarding the Rationale for the emergency update We find that 63.4 % of the release notes do not include any useful information about the rationale for the emergency update. 59.8 % of the emergency updates use the same release note as the previous update and 3.7 % of the emergency updates changed their release notes, however with very general descriptions, such as *"Fix bugs and add new features"*.

We find that both apps with long or short update cycles rarely include the rationale for the emergency update in their release notes. For example, the *"Emojidom: Chat Smileys and Emoji"*[7] app has a median update lifetime of 20 days. An update of the *"Emojidom: Chat Smileys and Emoji"* app was published on September 16th 2014 and an emergency update was published on the next day. Both updates have the exact same release notes. Similarly, the *"Sushi Bar"*[8] app with a long update lifetime of 409 days has one emergency update without updating the release notes that are associated with that emergency update.

We also investigate the release notes that are from the websites of the 361 studied apps with emergency updates. We find that 279 apps provide a link to their website on their store page in the Google Play Store. We manually checked the websites for these 279 apps and we find that only 18 apps provide release notes on their web site. We find that such online release notes provide no additional information over the release notes that are posted on the app's store page.

Our analysis suggest that developers should consider highlighting emergency updates in their release notes in order to encourage users to download these emergency updates.

3.3 Version Numbering of an Emergency Update

We would like to understand whether the update version numbers of emergency updates follow a certain format, in order to ease the further investigation of emergency updates. We study the difference between the version numbers for the emergency update (U_i) and the update preceding the emergency update (U_{i-1}).

There is no Fixed Numbering Convention for Emergency Updates We define *Version level* as the number of digits that are separated by dots (".") in a version number. In some cases, developers change the version numbers while keeping the same number of version levels (e.g., from version 2.32 to version 2.33). In other cases, developers use an additional version level for their emergency updates relative to the version number of the preceding update (e.g., from version 2.3 to version 2.3.1). Table 4 summarizes the different changes to version numbers that we find in the emergency updates.

From Table 4, we summarize the results as follows:

[7] https://play.google.com/store/apps/details?id=com.plantpurple.emojidom

[8] https://play.google.com/store/apps/details?id=com.roidgame.sushichain.activity

Table 3 Our major findings (and their implications) on the characteristics of emergency updates for top Android apps

Lifetime of emergency updates	**Implications**
Emergency updates have a long lifetime.	Users should update mobile apps with the emergency updates without being concerned bout another update showing up soon afterwards.
Release notes of emergency updates	**Implications**
Release notes of emergency updates rarely provide a clear description about the rationale for the update.	Developers should highlight the emergency nature of an update and encourage users to download it.
Version numbering of an emergency update	**Implications**
There is no fixed numbering convention for the emergency updates.	Developers may consider using version numbers to indicate whether an update is a major update or an emergency update.
Ratings of emergency updates	**Implications**
The updates preceding the emergency updates have more negative reviews than the emergency updates.	It would be beneficial if users are told about the recent rating of a newly available update (relative to the apps rating for its prior update) when users are informed of the availability of the update.

- In 11 % of the emergency updates, the update preceding the emergency update and the emergency update both have the **same exact version number**.
- In 89 % of the emergency updates, the update preceding the emergency update and emergency update have **different version numbers**. In such cases:
  - 71 % of the emergency updates have the **same version level** as the update preceding the emergency update (e.g., version 5.77 and 5.78).
  - 18 % of the emergency updates have an **additional version level** than the update preceding the emergency update (e.g., version 7.3 and 7.3.1).

Table 4 Study of version numbers in emergency updates

Changes in the version numbers of emergency updates	% of emergency updates
The update preceding the emergency update and the emergency update have the same exact version number.	11 %
The emergency update has the same version level as the update preceding the emergency update (e.g., version 5.77 and 5.78).	71 %
The emergency update has an additional version level than the update preceding the emergency update (e.g., version 7.3 and 7.3.1).	18 %

Based on the obtained results, we find that users cannot know whether an update is an emergency update just based on changes to version numbers. However, developers may consider making better use of version numbering patterns (and best practices) to indicate whether an update is a major update or an emergency update.

3.4 Ratings of Emergency Updates

In order to study changes in the ratings to the updates preceding the emergency updates. We focus on negative reviews (reviews with one or two stars rating) as these reviews mainly contain user complaints (Khalid et al. 2015; Martin Poschenrieder 2016). Among the total 1,000 emergency updates, 363 updates have no reviews for the update preceding the emergency update and 280 updates have too few reviews (with a median of two received reviews for the update preceding the emergency update). Therefore, we focus on the rest 357 updates to study the negative reviews.

We define the ratio of negative reviews (RNR) for an app A at date d as follows:

$$RNR\,(A,\ d)\ =\frac{N(A,\ d)}{T(A,\ d)} \tag{2}$$

where $N(A,\ d)$ is the number of negative reviews that are received at date d for an app A, and $T(A,\ d)$ is the total number of reviews that are received at date d for an app A.

For each app A that has an emergency update, as illustrated in Fig. 2, we calculate the ratio of negative reviews for the app during the following five dates:

- Date d_{2BE} is the deployment date of the second update preceding the emergency update.
- Date d_{BE} is the deployment date of the update preceding the emergency update.
- Date d_{ED1} is the deployment date of the emergency update.
- Date d_{ED2} is the following day to the deployment date of the emergency update.
- Date d_{ED3} is the second following day to the deployment date of the emergency update.

In order to compare the ratio of negative reviews, we use the $RNR\ (A,\ d_{BE})$ i.e., the ratio of negative reviews on the deployment date of the update preceding the emergency update as a baseline. We compare the ratios of negative reviews as follows:

$$Comparison_{2BE}\ (A)\ =\frac{RNR\,(A,\ d_{2BE})}{RNR\,(A,\ d_{BE})} \tag{3}$$

$$Comparison_{ED1}\ (A)\ =\frac{RNR\,(A,\ d_{ED1})}{RNR\,(A,\ d_{BE})} \tag{4}$$

$$Comparison_{ED2}\ (A)\ =\frac{RNR\,(A,\ d_{ED2})}{RNR\,(A,\ d_{BE})} \tag{5}$$

$$Comparison_{ED3}\ (A)\ =\frac{RNR\,(A,\ d_{ED3})}{RNR\,(A,\ d_{BE})} \tag{6}$$

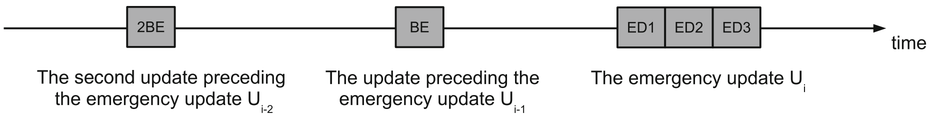
2BE
BE
ED1
ED2
ED3
time
The second update preceding the emergency update U_{i-2}
The update preceding the emergency update U_{i-1}
The emergency update U_i

Fig. 2 An overview of the studied dates for an app A

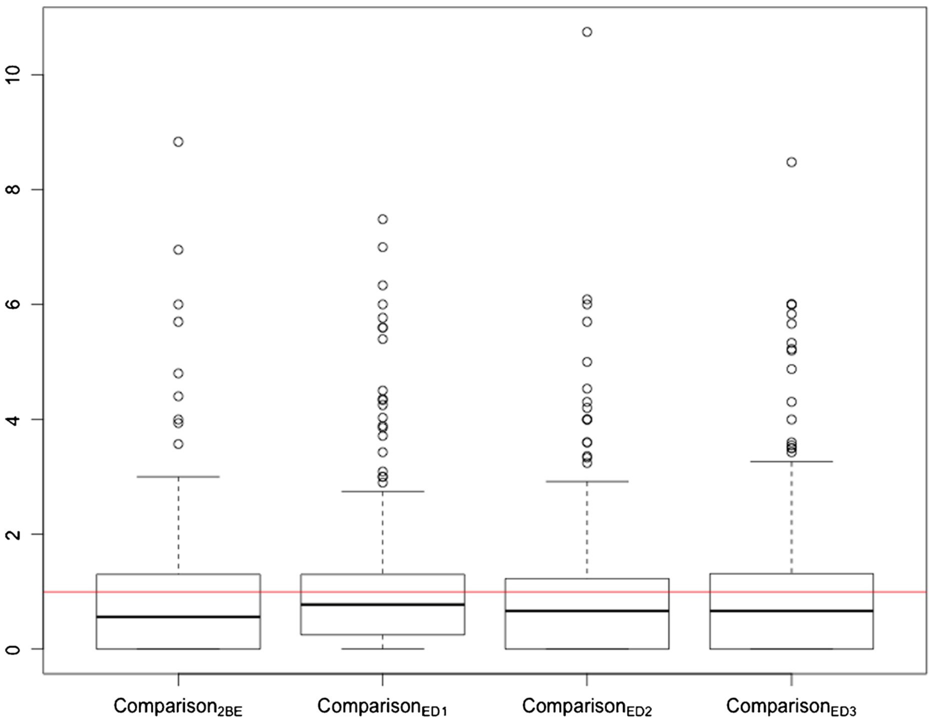

Fig. 3 Boxplot for the calculated comparison metrics for emergency updates. The red line in the figure shows the metric value 1. Metric values lower than 1 indicate that the update preceding the emergency update has a higher ratio of negative reviews than this emergency update

where $Comparison_{2BE}$ (A) compares the ratio of negative reviews between dates d_{2BE} and d_{BE}. The rest of the equations follow similar comparisons.

The Updates Preceding the Emergency Updates Have More Negative Reviews than the Emergency Updates We compare the ratio of negative reviews in different days as illustrated in equations (3, 4, 5 and 6) for all emergency updates (Fig. 3).

As shown in Table 5, the update preceding the emergency update (U_{i-1}) has a higher ratio of negative reviews than the 2nd update preceding the emergency update (U_{i-2}). Also

Table 5 Mean and five-number summary of comparison metrics for emergency updates

Metric name	Mean	Min.	1st Qu.	Median	3rd Qu.	Max
$Comparison_{2BE}$	0.89	0.00	0.00	0.56	1.29	8.83
$Comparison_{ED1}$	1.02	0.00	0.00	0.78	1.30	7.48
$Comparison_{ED2}$	0.91	0.00	0.00	0.67	1.22	10.74
$Comparison_{ED3}$	0.97	0.00	0.00	0.67	1.31	8.48

A value higher than one means that the ratio of negative reviews in the corresponding date is higher than the date of the deployment of the update preceding the emergency update we find that the update preceding the emergency update (U_{i-1}) has a higher ratio of negative reviews than the emergency update (U_i) in the 1st, 2nd, and 3rd deployment days. We notice that users still complain during the days after issuing the emergency update since users may have not downloaded the new update (the emergency update) yet. Interestingly, for some emergency updates, users may prefer the issues not being fixed, such as the emergency updates that are published to ensure that the app is able to continue displaying advertisement (based on our manual reading of user reviews).

Based on the observed results, it would be beneficial if users are told about the recent rating of a newly available update (relative to the rating of the prior update of the app) when users are informed of the availability of the update. This would permit users to decide on whether they wish to update their apps or not. For example, users might even configure their automated updaters to only install updates with an improved user rating.

4 Our Approach for Identifying the Patterns of Emergency Updates

In this section, we present our approach for identifying the patterns of emergency updates. In order to identify the patterns of emergency updates, as illustrated in Fig. 4, we leverage four data sources as follows:

1. The apk file for each update.
2. The release notes for each update.
3. The reviews associated with each update.
4. The F-Droid apps repositories data (F-droid 2016).

We explain the content of each source in the rest of this section.

4.1 The apk File for Each Update

First we unarchive the apk files of the emergency update (U_i), the two updates preceding the emergency update (U_{i-2} and U_{i-1}) and the update following the emergency update (U_{i+1}).

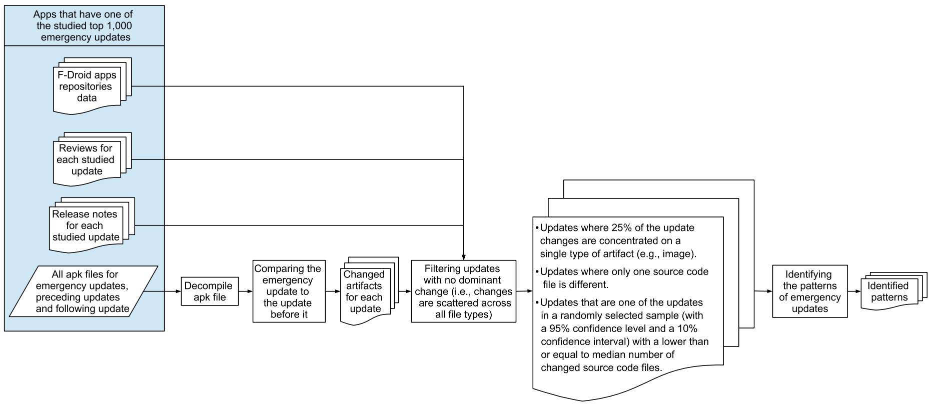
Apps that have one of the studied top 1,000 emergency updates
F-Droid apps repositories data
Reviews for each studied update
Release notes for each studied update
All apk files for emergency updates, preceding updates and following update
Decompile apk file
Comparing the emergency update to the update before it
Changed artifacts for each update
Filtering updates with no dominant change (i.e., changes are scattered across all file types)
•Updates where 25% of the update changes are concentrated on a single type of artifact (e.g., image).
•Updates where only one source code file is different.
•Updates that are one of the updates in a randomly selected sample (with a 95% confidence level and a 10% confidence interval) with a lower than or equal to median number of changed source code files.
Identifying the patterns of emergency updates
Identified patterns

Fig. 4 Process for identifying patterns of emergency updates

We unarchive each apk file using the Android-Apktool (Apktool 2016). Each unarchived apk file contains four folders and one AndroidManifest.xml file as follows (Sofia 2016):

- **Smali:** This folder contains the source code files (Sofia 2016; Smali2java 2016). In order to obtain readable source code, we converted the apk to jar using Dex2jar tool (Dex2jar 2016). We decompiled the generated jar into java source code using the Class File Reader (CFR) tool (CFR 2016).
- **Libs:** This folder contains third-party libraries that are used by the mobile app (Sofia 2016).
- **Res:** This folder stores the different resources that are needed by the app, such as images (stored in "drawable" folders), layout, style, colors, and configuration files for the various customizations that are used across the app (e.g., the displayed text) (Sofia 2016).
- **Assets:** This folder contains raw files that a developer may want to use in the app, such as texture, audio, text, fonts, and game data files. The raw data files can be stored either in the sub folder raw in the Res folder (res/raw) or in the assets folder: Files in the raw folder need to be accessed via the resource identifier, while files in the assets folder are accessed using the Java filesystem API without constraints on the file names (Sofia 2016; Stackoverflow 2016c, d; Android Studio 2016; Houston 2016).
- **AndroidManifest.xml:** The manifest file specifies the required configuration for the Android platform for the proper execution of the app (Google Android 2016d). For example, the manifest file specifies the required Android SDK version for the mobile app to work properly (Google Android 2016a). Developers also configure the needed permissions (Google Android 2016c) and the needed software or hardware features (such as camera, Bluetooth, app widgets) (Google Android 2016b) that are required by the app.

In total, we only encounter 56 apks where the unarchiving or decompilation failed in our experiment.

We compare the decompiled binaries in the emergency update (U_i) and the preceding update (U_{i-1}) to find what is changed in the emergency updates. We define ten types of files: code, third-party libraries, images, layout, style, colors, audio and video, displayed text, application configurations, and changes in the AndroidManifest.xml file. Then we categorize the similarly changed files together into one type, e.g., the images files are grouped together into image type, all source code files are grouped into the code type and changed XML elements in AndroidManifest.xml file as the AndroidManifest.xml change type. After that, we count the number of changes in each type. For the artifacts that are typically at the file level, e.g., figures and source-code files, we count the number of files that are changed. For the artifacts that are not at the file level, e.g., permissions and SDK version, we count the number of individual items of the artifact. In particular, changes to the the AndroidManifest.xml are not counted as only one change. Then we calculate the total number of changes in the apk file as the total number of items changed in the AndroidManifest.xml file and the total number of changed files (other than the AndroidManifest.xml file). Finally, we calculate the percentage of changes in each type relative to the total number of changes in the apk file.

We examine the decompiled source code files that are different between the emergency update (U_i) and the preceding update (U_{i-1}). We find that the median percentage of files that are the same between the two updates is 99.7 %, and the minimum percentage is 1.2 %. We manually investigated the updates with a very large number of source code files that are different to the files in the proceeding updates (i.e., low percentage of unchanged files). In particular, we rank the updates with the percentage of unchanged files across the two updates. We find that the updates in the first quartile have up to 98.3 % of the files that are the same between the update and the previous one. We manually investigate several of the updates in the first quartile. We find that there are two reasons for the low percentage of unchanged code files across updates:

1. The low percentage of unchanged source code files is due to code obfuscation, instead of actual code changes.
2. The app has too few app-specific code files and the rest of the code files are library files (e.g., advertisement library files). Hence, the adoption of a different library would lead to a very large percentage of changed source code files in that particular update. For instance, the update of the *"Fingerprint Lock Free"*[9] app on May 3rd 2014 contains only 1.2 % of the code files that are unchanged relative to its prior update. This update involves changes to four out of nine app-specific source code files and 651 out of 654 third party advertisement libraries files.

Since around two thirds of the emergency updates do not specify the rationale for their update (see Section 3.2), we need to manually analyze some of these emergency updates. In order to ease our understanding of the rationale for an emergency update, we manually examine an emergency update if one of the following three selection criteria hold:

– At least 25 % of the update changes are concentrated on a single type of artifact. For example, all of the files (100 %) in the images type is changed in the *"Pediatrics"*[10] app on January 8th 2014. Therefore, we manually examine this emergency update. If there are no major changes to the artifacts, it would be very difficult for us to deduce the root-cause of the emergency update.
– There is only one code file that differs (i.e., a new file, a removed old file or a changed old file) between the emergency update (U_i) and its preceding update (U_{i-1}), as any large amounts of source code differences between the two updates are likely due to code obfuscation. Furthermore, it is very challenging for use to deduce the rationale for such large changes given that we are using decompiled code and we have very limited knowledge of the apps and their codebase.
– The update is one of the updates in a random sample of updates (with a 95 % confidence level and a 10 % confidence interval) that have lower than or equal to median number of changed source code files. In order to create the random sample, we count the median number of changed source code files in the emergency updates and find that the median number of changed source code files is four. We collect all 376 updates that have less than or equal to four changed source code files and randomly select a sample of 77 updates for manual inspection. The size of our random sample achieves a 95 % confidence level and a 10 % confidence interval.

In our study, we examine the updates for which we can identify the rationale for the changes without the need for the release notes. E.g., 1) updates where at least 25 % of

[9]https://play.google.com/store/apps/details?id=com.nb.fingerprint.lock.free

[10]https://play.google.com/store/apps/details?id=com.texterity.android.Pediatrics

the update changes are concentrated on a single type of artifact (such as images, requested permissions, layout, colors, Android SDK versions), 2) updates with only one source code file being different, or 3) the random sample of updates for manual inspection.

In total, we select 361 emergency updates to manually examine based on our aforementioned selection criteria.

4.2 The Reviews Associated with Each Update

Google Play Store enables users to provide their reviews for each update of an app. We manually examine the reviews that are associated with each studied emergency update (U_i) and the preceding update (U_{i-1}).

4.3 The Release Notes for Each Update

Release notes are also one of the data sources that are used to study the emergency updates. However, about 60 % of the emergency updates use the same release notes as their preceding update, and 3.7 % of the emergency updates only have general words for the emergency updates, such as: *Hot-fix and Various Bug Fixes* (see Section 3.2).

In our manual process for identifying patterns of an emergency update, first we start by reading the release notes for the emergency update and comparing the details that are mentioned in the release notes to the changed files in the emergency update. If the release notes do not mention any useful details about the rationale for the emergency update, then we manually read all the reviews associated with the emergency update (U_i) and the preceding update (U_{i-1}). We excluded reviews that contain generic user complaints and considered reviews that mention details that are related to each identified pattern.

If the emergency update does not include useful release notes or reviews that are explicitly related to the identified patterns, we manually compare the changed files in the four updates (the emergency update (U_i), the update following emergency update (U_{i+1}), and the two updates preceding the emergency update (U_{i-1}) and (U_{i-2})) in order to identify the rationale for the change. For example, in order to identify that incorrect images were included in a particular update (U_i), we compare the images content in the two preceding updates (U_{i-1}) and (U_{i-2}) in order to see the content before the emergency update, then we compare the emergency update with the preceding update (U_{i-1}) in order to identify the changed images. Finally, we compare the content of the updates (U_i) and (U_{i+1}) in order to ensure that the updated files are the correct images that are still used in the following updates.

4.4 The F-Droid apps Repositories Data

The release notes of app updates may be short and with limited details (see Section 3.2). In order to have more information about the update, we explore another source of data to understand the patterns of emergency updates. In particular, we find that 11 apps with emergency updates are hosted in a software repository named F-Droid (F-Droid 2016). F-Droid provides a public collection of different FOSS (Free and Open Source Software) apps. We collect all the available releases for the 11 apps from F-Droid. We search for the emergency update (U_i) and the preceding update (U_{i-1}) releases. Then we collect the code comment in the source code files and the commit messages, for the changed files between the two releases for the updates (U_i) and (U_{i-1}). We collect the repository release notes, which are different from the release notes shown in the Google Play Store,[11] for the emergency update (U_i).

The releases of emergency update (U_i) or the release of the preceding update (U_{i-1}) are not always explicitly tagged in the repository. In such cases, we use the update time to find the related code commits for the emergency update (U_i) and the preceding update (U_{i-1}). Then we use the commit messages and code comments that are associated with these particular code commits to understand the root-cause for the emergency updates.

We have the following three cases for the 11 apps that are hosted on F-Droid:

1. The repository is not available. We find two updates where either the repository URL is not working or the repository does not store the commits for the code changes during our study period.
2. The repository is available but we cannot identify the code changes to the emergency update. We find three updates in this case.
3. The repository is available and we could benefit from the releases and code changes in order to map the studied updates to the actual changed code. We find six updates in this case.

Such results show that even though F-Droid provides much detailed development information about mobile apps, data analytics on mobile apps cannot rely solely on F-Droid due to its small scale and the quality of its data.

4.5 Manual Inspection

We manually inspected all emergency updates that meet our aforementioned selection criteria. In our process, we go through the following two steps:

- **First step – identifying patterns of emergency updates.** The first and second author of this paper work together by manually examining all differences between all the associated updates. We examine the differences between the downloaded apks, release notes, F-Droid repositories data and app reviews in order to know the issue that is fixed in each update. If the issue is a new issue then we add it to the list of identified issues. We iteratively examine all the updates until there are no more identified issues. If there is disagreement during the issue identification process, the first two authors together come to a consensus. In particular, we have 20 updates which there is disagreement in the identified issues. After finalizing the list of identified issues, we consider only issues that are fixed in more than one update as patterns for emergency updates. The output of the first step is the list of the identified patterns along with the updates that are related to each pattern.
- **Second step – identifying root-causes of each pattern.** Similar to the first step, the first and second author of this paper examine all the updates associated with each pattern in order to study the root-causes for each pattern. For each inspected emergency update, we use all available data in order to identify the root-cause for the update. If we identify a new root-cause for a certain pattern, we add this root-cause to the list of the root-causes that are related to this pattern. We iteratively examine all the updates in the pattern until there are no more identified root-causes.

[11] We refer to the release note collected from the Google Play Store as “release notes” and the release notes collected from the apps repositories as “repository release notes”.

Table 6 Patterns of deployment issues emergency updates

Updates due to deployment issues				
Pattern name	Description	Root-cause	Identified updates	App names
Including low quality images	An update is published to repair image quality issues (image resolution or image brightness).	1) Developers do not test the quality of images on devices with varying screen resolution. 2) Developers forget to test newly added images.	9	"*Read Unlimitedly! Kids'n Books*", "*Chomp SMS theme add-on*", "*Learn Portuguese with Babbel*", "*Learn French with Babbel*", "*Learn German with Babbel*", "*Our Groceries Shopping List*", "*Curso de Ingles Gratis*", "*Naver*" and "*Baby FlashCards for Kids*"
Including incorrect images	An update is published to replace the incorrect image with the appropriate one.	1) Developers of multiple mobile use the wrong images from their other apps. 2) Developers miss updating old images.	18	"*Camera ZOOM FX Buddy Pack*", "*Pediatrics*", "*Camera ZOOM FX Halloween Pack*", "*Navmii GPS USA (Navfree)*", "*Davis's Drug Guide*", "*10 K Runner Trainer FREE*", "*C25 K ©- 5 K Runner Trainer FREE*", "*OnLive*", "*Camera ZOOM FX Composites*", "*Camera ZOOM FX New Composites* ", "*Camera ZOOM FX Extra Props*", "*Camera ZOOM FX Props Pack*", " *Camera ZOOM FX More Composites*", "*Camera ZOOM FX Picture Frames*", "*Camera ZOOM FX Cool Borders*", "*Brain Age Test Free*", "*Police Lights and Sirens Pt.*" and "*Safeway*"
Having inconsistent permissions	An update is published to remove the request for not needed permissions or to request the needed permissions that are missing.	1) Developers use some permissions during development but some of these permissions are no longer needed for the most recent app update. 2) Developers forget to add some of the needed permissions. 3) Developers make mistakes when defining customized permissions.	13	"*Easy Uninstaller App Uninstall*", "*Free Sports Radio*", "*Mobiletag QR & product Scanner*", "*Higher One Mobile Banking App*", "*Mixology ™Drink Recipes*", "*Bingo Fever - Free Bingo Game*", "*Spanish Translator*", '*One More Clock Widget Free*", "*OPM Alert*", "*Evernote Widget*", "*Horoscope HD Free*", "*AroundMe*" and "*AnyTimer Pill Reminder*"

Table 6 (continued)

Updates due to deployment issues

Pattern name	Description	Root-cause	Identified updates	App names
Having inappropriate SDK versions	An update is published to fix the needed SDK version of the app.	1) Developers mistakenly leverage new features from a new version of the SDK without specifying the need of the new SDK version in the most recent app update. 2) Developers make a code change and downgrade or upgrade the target SDK version, which is discovered to lead to problems in the field.	18	"*Guitar Lessons Free*", "*Horoscope and Tarot*", "*Convert video to mp3*", "*Mp3 Converter Free*", "*Chinook Book*", "*Stock Watcher*", "*Voxer Walkie Talkie Messenger*", "*Pyramid Spirits 3 - Slots*" "*Simple Notepad*", "*Useful Knots*", "*DJ*", "*FVD - Free Video Downloader*", "*BoothStache*", "*UglyBooth*", "*MixBooth*", "*AgingBooth*", "*FatBooth*" and "*BaldBooth*"
Incorrect debugging mode	An update is published to enable or to disable the debugging mode.	Developers need to enable or disable the debugging mode.	10	"*Office Calculator Free*", "*Mixology^{TM} Drink Recipes*", "*Offroad Legends*", "*Foods That Burn Fat*", "*Real Time CPR Guide*", "*Prenatal Ultrasound Lite*", "*TAGstagram - IG TAG searcher*" and "*VAT CALCULATOR*"

Table 7 Patterns of source code changes emergency updates

Updates due to source code changes				
Pattern Name	Description	Root-cause	Identified updates	App names
Invoking unavailable APIs	An update is published to check the availability of APIs before calling the APIs.	Developers find that their app crashes due to calling certain features in Android APIs that may not exist in some SDK versions on user devices.	5	"*justWink Greeting Cards*", "*Couple Tracker -Mobile monitor*", "*DJ Control*", "*Free Golf GPS APP - FreeCaddie*" and "*Word Learner Vocab Builder GRE*"
Advertisement issues	An update is published to ensure the correct display of advertisements.	1) Developers do not set the correct identifier for the advertisement. 2) Developers do not validate how advertisements are loaded and displayed in the app.	19	"*Tractor Pull*", "*KWCH 12*", "*Pocket Tanks*", "*Baja Trophy Truck Racing*", "*Make Up Salon!*", "*LoveCycles - Period Tracker*", "*Drift Mania Championship Lite*", "*Update Samsung Android Version*" and "*Crazy Grandpa*"
Un-handled exceptions	An update is published to handle exceptions.	Developers do not handle all possible exceptions that can occur when users run the app.	54	"*Photosphere Free Wallpaper*", "*SlenderMan LIVE*", "*FrostWire - Torrent Downloader*", "*GROWLr: Gay Bears Near You*", "*Fast Burst Camera Lite*", "*Zen Pinball*", "*Hersheypark*", "*Harvest Time & Expense Tracker*", "*First Aid Emergency & Home*", "*Flashlight + Clock*","*FeaturePoints: Free Gift Cards*", "*GPS Phone Tracker Pro*", "*Followers+ for Twitter*", "*Buycott - Barcode Scanner Vote*", "*Highway Crash Derby*", "*Next Music Widget*", "*GO Cleaner & Task Manager*" and "*Speed Card Free*"

Table 8 The median speed of repair for all patterns of emergency updates

Category	Pattern name	Median speed of repair (days)
Updates due to deployment issues	Including low quality images	1
	Including incorrect images	3
	Having inconsistent permissions	1
	Having inappropriate SDK versions	1
	Incorrect debugging mode	2
Updates due to source code changes	Invoking unavailable APIs	2
	Advertisement issues	1
	Un-handled exceptions	1

5 Identified Patterns for Emergency Updates

We could map 146 updates to certain patterns and we cannot identify the patterns for the remaining 215 updates (361 in total). We identify two categories of emergency patterns including "Updates due to deployment issues" and "Updates due to source code changes". Each category consists of different patterns. Tables 6 and 7 summarize the identified patterns for emergency updates. For each pattern, we discuss the description of the pattern and the root-cause of the pattern with some real-life examples. We also discuss how fast developers address the root-cause problem of the pattern (as shown in Table 8) and show some examples of user complaints. Finally, we discuss the lessons learned from each pattern.

5.1 Updates Due to Deployment Issues

We identify 68 emergency updates that are due to deployment issues: nine updates that are done to address image quality issues, 18 updates that are done to address image content issues, 13 updates that are done to address inconsistent permissions, 18 updates that are done to address inappropriate SDK versions and ten updates that are done to address incorrect debugging mode.

5.1.1 Including Low Quality Images

Pattern description:

An update is published to repair image quality (e.g., image resolution or image brightness). If an emergency update (U_i) fixes the quality of the displayed images in the update preceding the emergency update (U_{i-1}), we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find three root-causes of this pattern:

1. Developers may not have the resources to verify the quality of images on every single device on which their apps may run. The images may not be of a suitable quality (e.g., resolution) for all mobile devices (especially for mobile devices with high screen resolution).

2. Developers may forget that they included new images in an app and they forget to test the quality of such new images.
3. Developers add images and icons without knowing whether users would like the new images and icons.

Example updates:

- An update of the "*Chomp SMS theme add-on*"[12] app on August 17th 2014 has icons that are not comfortable for users (very bright). An emergency update adjusts the images to be less fluorescent.
- An update of the "*Baby FlashCards for Kids FREE*"[13] app on November 25th 2013 has images with sizes that are not suitable for all devices. An emergency update adjusts the images sizes to be suitable for all devices, especially for the devices with high-resolution screens.

Examples of user complaints:

We find user complaints (in the app reviews) for only one out of the nine updates that are related to this pattern. The users become frustrated with the image quality issues and complain about this pattern. The update of "*Chomp SMS theme add-on*" app on August 17th 2014 has an issue in the displayed icons. Users complain about the icons in the app reviews, such as "*Love the app ever since I purchased it a while back ! The icon not so much.*" and "*Love the update but not a big fan of the new icon*". The development team repairs the icon images and publishes a new update with release notes "*Thanks for valuable user feedback, we appreciate it and have updated the icon to be less fl(u)orescent!*".

Speed of repair:

As shown in Table 8, this pattern requires a median of one day to repair (i.e., the median lifetime for the update that precedes the emergency update is one day). The short time to repair indicates that it is not hard for developers to replace the image with ones of higher quality.

Lessons learned for developers:

Developers should examine the quality of the images when new images are added to the mobile app or when the mobile app starts to support new devices with different screen resolutions. Having an image of low resolution may not be suitable for some devices with higher resolution screens. Developers should always consider making different versions of images with different quality to suit the different devices. A recent study of user reviews have proposed approaches to prioritize the devices that need to be tested based on the reviews (Khalid et al. 2014). Developers may leverage such an approach to prioritize their effort in order to test the quality of images on various devices.

Prior research studies the relationship between the colors that are used in products and the successfulness of the products (Alvarez 2016; Smith 2016; Apple Developer 2016; Arakelyan 2016). Tools have been proposed to assist in evaluating whether the used colors

[12] https://play.google.com/store/apps/details?id=com.p1.chompsms

[13] https://play.google.com/store/apps/details?id=au.com.alexooi.android.flashcards.alphabets

in a product are comfortable for end users (Colblindor 2016; Color Oracle 2016). Developers may leverage results and tools of the prior studies to select the most suitable colors for their apps.

Lessons learned for store owners:

Mobile app store owners can enhance the current review mechanism by enabling users to upload screenshots for their complaints. Moreover, app store owners can augment their existing tooling to notify app developers about images that may not appear well on some devices so developers are aware of the issue earlier (e.g., as part of the automated verification of an app that is published by the store owners for each new update of an app before making the update available on the store).

5.1.2 Including incorrect images

Pattern description:

An app has incorrect images. An update is published to replace the incorrect images with the appropriate ones. If an emergency update (U_i) fixes the content of the displayed images in the update preceding the emergency update (U_{i-1}), we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find two root-causes of this pattern:

1. Developers of multiple mobile apps mistakenly package incorrect images across their other apps.
2. Developers forget to include updated images in a new update.

Example updates:

– The *“Camera ZOOM FX Buddy Pack”*, *“Camera ZOOM FX Halloween Pack”*, *“Camera ZOOM FX Composites”*, *“Camera ZOOM FX New Composites”*, *“Camera ZOOM FX Extra Props”*, *“Camera ZOOM FX Props Pack”*, *“Camera ZOOM FX More Composites”*, *“Camera ZOOM FX Picture Frames”*, and *“Camera ZOOM FX Cool-Borders”* apps have issues in their updates on June 10th 2014 when all these apps use the same set of images. Developers had mistakenly replaced the images with Halloween photos from the *“Camera ZOOM FX Halloween Pack”* app. The developers repair this issue by updating the images for each of their apps.
– Developers forgot to update the main introductory image for the app portfolio of the *“Pediatrics”* app on January 6th 2014. Developers replace the wrong image with the correct image in an emergency update.
– The “Brain Age Test Free”[14] app stopped using the Heyzap advertisement network (Hyzap 2016) but the developers forgot to remove the Heyzap advertisement in their

[14] https://play.google.com/store/apps/details?id=brain.age.analyzer

image. They published an emergency update on July 31st 2014 that removes all images that represents Heyzap. The release notes for this update are *“No more Heyzap”*.

Examples of user complaints:

We find user complaints (in the app reviews) for only one out of the 18 updates that are related to this pattern. The *“Camera ZOOM FX Extra Props”*[15] app has a user complaining that the different *“Androidslide”* apps have the same content. The small number of reviews may be because not all users realize the wrong image content.

Speed of repair:

The median time to repair the updates with this pattern is three days (shown in Table 8). In comparison to the pattern of “Including low quality images”, this pattern takes a longer time to repair. We believe that the slower repair pace may be due to the users not complaining about this pattern as often as the pattern of including low quality image. The fewer complaints may cause the developers to not realize the issue right after the update, or the developers realize the issue but are not as hard pressed to repair it, given the low number of complaints.

Lessons learned for developers:

There exist automated GUI testing tools for mobile apps, such as Robotium (2016). However, automated GUI testing on mobile apps is effort consuming and challenging (Joorabchi et al. 2013). In order to avoid this pattern, developers need better automated testing tools that can reduce their testing efforts. Developers might wish to consider using different build environments for their different apps to ensure that each app has separate resources.

Developers need better code analysis tools that track the dependency between two changed elements in the app (e.g., which parts of the code that are related to which resources in the app) (Hassan and Holt 2004; Zimmermann et al. 2004). For example, If developers change source code, tools can notify developers with a list of non-code resources (e.g., images) that should be updated.

Lessons learned for store owners:

Mobile app store owners should augment their automated verification process of new updates so that the process would examine all recent updates across a single organization not just for the current app update. The app store may then warn app developers about the repeated content (e.g., resources and configurations) across their different apps. Such automated analysis might also help flag spam apps (i.e., apps with similar content and very slight variations) (Ruiz et al. 2012, 2014).

[15] https://play.google.com/store/apps/details?id=slide.cameraZoom.extraprops

5.1.3 Having inconsistent permissions

Pattern description:

A mobile app requests a list of permissions. Some of these permissions may not be actually needed or, on the other hand, some needed permissions are not requested. An update is published to remove the request for the not needed permissions or to request the needed permissions that are missing. If an emergency update (U_i) fixes the requested permissions in the AndroidManifest.xml file, we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find four root-causes for this pattern:

1. Developers do not examine whether all the requested permissions by the app are actually needed. For example, developers sometimes use tools to assist during development but these tools may require permissions to function correctly, however such permissions are not needed once the app is published (Telerik 2016).
2. Developers may use some permissions during development but forget to remove them before issuing their update.
3. Developers forget to add some needed permissions.
4. Developers restrict the permissions for certain SDK versions and discover after issuing an update that the permission needs to be requested for all SDK versions.

Example updates:

– An update of the *"Free Sports Radio"*[16] app on March 8th 2014 requests several not needed permissions. The developers removed the not needed permissions "Read External Storage", "Write External Storage", "Read User Directory", and "Write User Directory" in an emergency update.
– To repair an issue in an update of the *"Higher One Mobile Banking App"*[17] app on April 29th 2014, developers needed the "Read Phone State" permission, which requests read access to the phone state. This permission is not needed any more after the fix. However, developers of the *"Higher One Mobile Banking App"* app forgot to remove the permission for this update. An emergency update removes the permission since it is no longer needed.
– The *"Spanish Translator"*[18] app has an update on November 11th 2014 and the update has unused permissions *Read Phone State* and *Access Network State*. On the next day, the development team publishes a repair that removes these permissions that are not needed.

[16]https://play.google.com/store/apps/details?id=com.MyIndieApp.FreeSportsRadio

[17]https://play.google.com/store/apps/details?id=com.higherone.mobile.android

[18]https://play.google.com/store/apps/details?id=pl.pleng.spanish

- An update of the *"One More Clock Widget Free"*[19] app on November 4th 2014 was restricting the "Write External Storage" permission to certain SDK versions. Developers discover that they need to request the "Write External Storage" permission for all SDK versions. To repair this issue, developers correctly reconfigure the permission request in an emergency update by removing the "maxSdkVersion" attribute so the "Write External Storage" permission is requested for all SDK versions.
- An update of the *"Evernote Widget"*[20] app on May 16th 2014 misses to request the "Read Data" and "Write Data" permissions. Developers publish an emergency update to request these missing permissions.

Examples of user complaints:

We find that two out of 13 updates that are related to this pattern (the updates for the *"One More Clock Widget Free"* and the *"Evernote Widget"* apps) have user complaints.

The missed permissions introduces high severity issues as the app may not work because of these missed permissions. Below are examples of user complaints:

- The update of the *"Evernote Widget"* app on May 16th 2014 misses to request the "Read Data" and "Write Data" permissions. Users start to complain that the updated app is not working. Users leave the following reviews on the same day of the update, such as a review *"Application No longer working correctly since evernote update"*, and another review with title "Stopped working" and content *"The new update does not work"*. Developers publish the emergency update that requests the missing "Read Data" and "Write Data" permissions.
- The update for the *"One More Clock Widget Free"* app on November 4th 2014 restricts the request for the "Write External Storage" permission to some SDK versions. Users start complaining that they cannot open the app. For example, user wrote a review on November 4th 2014 with title *"Was good"* and comment text *"App was great till latest update, then wouldn't open. Will try again later"*. Developers publish an emergency update on the next day to fix this permission issue.

Speed of repair:

As shown in Table 8, the median time to repair the updates related to this pattern is one day. The reason for this fast repair can be explained as follows:

- In the case where the emergency update removes the unneeded permissions, we think the fast turnaround may be due to developers taking permission requests very seriously. For example, according to a recent survey of 2,272 users, 49 % of mobile app users do not download at least one app due to privacy concerns (Stuart Dredge 2016).
- In the case where the emergency update adds the missing permissions, we find that reviews often report app crashes for missing to request the needed permissions. The app crashing may be the reason for this fast speed of repair.

[19]https://play.google.com/store/apps/details?id=com.sunnykwong.freeomc

[20]https://play.google.com/store/apps/details?id=com.evernote.widget

Lessons learned for developers:

Developers should better track the requested permissions and their corresponding source code or third party libraries. With strong and up to date traceability links between requested permissions and the source code of an app or third party libraries, developers can ensure that all requested permissions are needed and all needed permissions are requested correctly (Stackoverflow 2016a, b, e).

In order to track the inconsistency in the requested permissions, many researchers introduced tools to verify the needed permissions (Felt et al. 2011; Xu et al. 2013; Pandita et al. 2013; Gorla et al. 2014). Developers should leverage the existing permissions tracking tools to identify any inconsistency between the requested permissions and the app behavior.

Lessons learned for store owners:

Mobile app store owners should leverage the existing permission tools in order to prevent app developers from issuing updates with unneeded or missing permissions.

5.1.4 Having Inappropriate SDK Versions

Developers define the minimum and target SDK versions in the Android Manifest file. The minimum SDK version represents the minimum SDK version that needs to be installed on the mobile device in order to assure that the app runs properly (Google Android 2016a). Users, who installed a later version of the Android platform, can run the updated app. For example, if a user has SDK version 5.0 installed on his mobile device and the app has a minimum SDK version 7.0, this update will not appear to this user (Google Android 2016a; Simon 2016). Developers change (usually increase) the minimum SDK version because they use new features that are introduced in a certain SDK version. By upgrading the minimum SDK version to a higher value, developers prevent the update from being installed on devices with lower SDK versions. If the app runs on an older SDK version, the app may crash because the app may call features that are not supported by the older SDK version (Simon 2016).

The target SDK version represents the SDK version that is targeted by the development team (Google Android 2016a). Adjusting the target SDK version to a certain value means that this SDK version is the version that development team use to test the app. For example, if an app has a target SDK version lower than the user's installed SDK version, the Android platform may apply compatibility behavior in order to make the app run in the same way as expected by app developers (Google Android 2016a). For example, The "Honeycomb" version of Android provides a set of themes called "Holo" themes, by identifying the target SDK version to a lower version than Android Honeycomb, the Android platform will not enable the "Holo" themes for these apps in order to prevent issues like drawing a black text on a black background (Google Android 2016a; Simon 2016).

We find 18 emergency updates that are due to issues related to SDK versions (either minimum or target SDK versions). Eight updates are done to address the issue of missing to update an old SDK version and ten updates are done to address the issue of using a wrong SDK version.

Pattern description:

A mobile app requires a certain SDK version to run, however the app fails to specify the need for this SDK version in its AndroidManifest.xml file. An update is published to fix the needed SDK version of the app. If an emergency update (U_i) fixes the required SDK version of the app, we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find two root-causes for this pattern:

1. A new version of the SDK often provides new features, such as additional APIs. Developers may leverage such new features from a new version of SDK. However, an update might miss specifying the need for a new SDK version in the AndroidManifest.xml file.
2. Developers make a code change and downgrade or upgrade the SDK version. After the update, developers discover that the SDK version change introduces issues (e.g., menus do not appear) on some devices. Developers perform an emergency update to correct the SDK version to the version that is suitable for the new changes, or to revert back their new changes and to continue using the original SDK version.

Example updates:

– An update of the *"Stock Watcher"*[21] app on September 25th 2014 misses to change the target SDK version. Developers release an emergency update on the next day in order to update the SDK version with release notes *"Fix the disappearing menu button on some devices"*.
– An update of the *"DJ"*[22] app on November 5th 2014 increases the target SDK version from 10 to 21. The developers discover an issue that the menu key does not appear, such that users cannot shut down the app. The developers repair the issue by reverting the *SDK* version back to 10.

Examples of user complaints:

We find that four out of the 18 updates that are related to this pattern (for the *"Voxer Walkie Talkie Messenger"*[23], *"DJ"*, *"AgingBooth"*[24] and the *"FatBooth"*[25] apps) have users complaining about symptoms that are related to this pattern:

– The *"Voxer Walkie Talkie Messenger"* app on September 15th 2014 does not update the minimum SDK version. A user posts a review with the title *"Voxer freezing"* and the

[21] https://play.google.com/store/apps/details?id=com.mobileappsresearch.stockwatcher

[22] https://play.google.com/store/apps/details?id=com.spartacusrex.prodjlite

[23] https://play.google.com/store/apps/details?id=com.rebelvox.voxer

[24] https://play.google.com/store/apps/details?id=com.piviandco.agingbooth

[25] https://play.google.com/store/apps/details?id=com.piviandco.fatbooth

following review content *"After new update voter is freezing and not letting me listen to full vox messages"*. Developers publish an emergency update on the following day to increase the minimum SDK version from version 8 to version 10.

– The *"AgingBooth"* app has an update on June 20th 2014 and users report issues about this update. For example, a user leaves a review with title *"Crashed on first use"* and content *"Cannot use it. Let's me take pic and then adjust markers. Then it closes unexpectedly for no apparent reason. Uninstalling"*, another user leaves a review with content *"Force closes after I take a picture, as I can see it(')s doing the same for everyone else too"*. Because of the large impact on the user experience, the development team publishes an emergency update to repair this pattern in two days.
– Before updating the *"DJ"* app with the emergency update, we observe that the users complain that the app cannot shutdown. For example, a user leaves the review *"Since it(')s for free but with add - you cannot shut down anymore this app. Craps!!"*. After the emergency update, the same user confirms that the app is working well *"NOW it works PERFECT! GREAT technical support and best DJ app. THANKS"*.
– The app *"FatBooth"* has an update on June 20th 2014 and users complain about this update. For example, a user writes a comment with the title *"Force close"* and the following text: *"Since last update it automatically force closes"*. Another user left a comment *"I (t)take a picture, then go to do it and it force close every single time. Stupidity"*. The development team published an emergency update to upgrade the minimum SDK version from version 9 to version 10 and the target SDK version from version 20 to version 21.

Speed of repair:

The median time to repair the SDK version issues pattern is one day (as shown in Table 8). The reason of this pattern being repaired fast is that this pattern is easy to fix. Moreover, this pattern has a large impact on users since every user with the inappropriate SDK version is impacted.

Lessons learned for developers:

To ensure the correct running environment of apps, developers must correctly specify the minimum requirement of SDK version in the AndroidManifest.xml file. However, there exist no automated techniques to ensure that the specified minimum requirement of SDK version is the correct one. Therefore, for every update of an app, developers need to verify the correctness of the specified minimum SDK version. Automated techniques are needed to analyze the app source code and identify the minimum SDK version that is needed for the current code, then to notify the developers with any inconsistency between the needed and the requested minimum SDK versions.

Prior research finds that updating to a new SDK version may be harmful since the API change may lead to defects in mobile apps, leading to a negative impact on the user ratings (Bavota et al. 2015; Linares-Vásquez et al. 2013). Before updating the SDK version, developers need to understand the impact of the update. A thorough regression testing process is needed to compare the behavior of an app with the old and the new SDK versions.

Lessons learned for store owners:

Mobile app store owners need automated tools that warn developers if they change the source code without updating the needed SDK version, or if the new source code is not compatible with the specified minimum SDK version.

5.1.5 Incorrect Debugging Mode

Pattern description:

A mobile app uses debugging functionality to store information about the app behavior. Developers find a need to enable or disable the debugging mode. An update is published to change the debugging mode. If the emergency update (U_i) changes app source code in order to enable or disable the debugging mode, we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find two root-causes of this pattern:

- Developers forgot to disable the debugging mode. Developers publish an emergency update to disable the debugging mode.
- Developers find that there is a need to store the debugging information in order to track the app behavior, so an emergency update is published to enable the debugging mode.

Example updates:

- Developers of the *"Office Calculator Free"*[26] app notice that the app was published with the debugging mode mistakenly enabled. An emergency update is published on June 30th 2014 to disable the debugging mode.
- Developers of the *"Mixology^{TM} Drink Recipes"*[27] app need to track debugging information about the app. An update is published on July 2nd 2014 to enable the debugging mode.

Examples of user complaints:

We did not find user complaints about this pattern. The lack of user complaints is likely due to users not being aware of the debugging and with debugging not having a large impact on the user experience with the app (e.g., performance).

Speed of repair:

As shown in Table 8, this pattern requires a median of two days to repair. The short time to repair is most likely due to enabling or disabling debugging information not requiring much effort from developers.

[26] https://play.google.com/store/apps/details?id=net.taobits.officecalculator.android

[27] https://play.google.com/store/apps/details?id=com.digitaloutcrop.mixology

Lessons learned for developers:

Developers should have a checklist to review the app configurations (such as enabling or disabling the debugging mode) before issuing the updates, in order to avoid the need for an emergency update.

5.2 Updates Due to Source Code Changes

We identify 78 emergency updates that are due to source code changes: five updates are done to address invoking unavailable APIs, 19 updates are done to address advertisement issues and 54 updates are done to address un-handled exceptions.

5.2.1 Invoking Unavailable APIs

Pattern description:

A mobile app does not consider the users' installed SDK versions while calling certain API features. App developers publish an emergency update that enables different behaviors based on the availability of API. If an emergency update (U_i) changes the source code to address the invocation of an unavailable API issue, we consider that the emergency update (U_i) belongs to this pattern.

Root-cause:

The root-cause for this pattern is that developers fail to consider the availability of an API before invoking it in their code.

Example updates:

An update of the *"Word Learner Vocab Builder GRE"*[28] app on June 19th 2014 has an issue in getting the app data using the Android "Context" object. The issue exists because the app invokes APIs that are not available in some versions of the Android SDK . An emergency update with version *"2.3"* is published to handle the described issue with release notes *"Version 2.3: Fixed crash on Android 4.2 and above"*.

Examples of user complaints:

We find complaints about the app crashing but we cannot be sure that the complaints about app crashing are related to the code changes in the emergency update.

Speed of repair:

As shown in Table 8, the "Invoking unavailable APIs" pattern requires a median of two days to repair. The "Invoking unavailable APIs" pattern needs longer time to fix than the other code related patterns (e.g., "Un-handled exceptions"). The reason may be that the

[28]https://play.google.com/store/apps/details?id=com.wordLearner.Free

"Invoking unavailable APIs" pattern needs more investigation and development effort than the other source code related patterns.

Lessons learned for developers:

We notice that developers handle this pattern by adapting the code to behave in a different ways depending on the availability of APIs on the user device. Development tools (e.g., development IDEs) can warn developers when their code calls certain methods that are not provided for some SDK versions that are configured by the app. Similarly, apps store owners can easily warn developers about such issues as part of their automated verification of new updates.

5.2.2 Advertisement Issues

Pattern description:

A mobile app uses third party libraries to display advertisement. Developers find an issue that is related to displaying advertisement. An update is published to ensure the correct display of advertisement. If an emergency update (U_i) changes app source code to fix issue in the calls to the ad libraries, we consider that the emergency update (U_i) belongs to this pattern.

Root-causes:

We find two root-causes of this pattern:

1. Developers use a pseudo value for the ad identifier and forget to update the ad identifier with the correct value.
2. Developers do not validate how the ad will be loaded and displayed in the app.

Example updates:

- An update of the "*Tractor Pull*"[29] app on May 11st 2014 has an issue in setting the ad identifier for the displayed advertisement. An emergency update is published on the next day to fixe the ad identifier value.
- An update of the "*KWCH 12*"[30] app on June 5th 2014 does not handle issues related to the rendering of advertisement. An emergency update is published on the next day to handle such issues. The emergency update adds retry mechanisms to load advertisement.

Examples of user complaints:

We did not find user complaints about advertisement in the updates that precede the emergency update. On the other hand, we find 128 recent reviews for the emergency updates

[29]https://play.google.com/store/apps/details?id=com.anddgn.tp.main

[30]https://play.google.com/store/apps/details?id=com.newssynergy.kwch

that are related to this pattern where users complain about the intensive existence of the advertisement in the app. Some users threaten to uninstall the app. For example, on the same day of the emergency update of the "*Tractor Pull*" app, users posted negative reviews such as "*Fun game but since last update get killed with ads! Hard to run gas peddle when a stupid ad covers right half of screen! Uninstall!!!*".

Speed of repair:

As shown in Table 8, this pattern requires a median of one day to repair. The short time to repair may be because displaying advertisement is one of the main sources of revenue for the free-to-download app developers.

Lessons learned for researchers:

More research needs to be done in order to provide recommendation about the common usability issues (e.g., best practices and common pitfalls) surrounding the integration of advertisements in mobile apps.

5.2.3 Un-handled Exceptions

Pattern description:

Mobile app code does not handle all possible scenarios and app crashes in certain scenario. An update is published to handle exceptions. If an emergency update (U_i) changes app source code to handle some previously unhandled exceptions that may be thrown by the code, we consider that the emergency update (U_i) belongs to this pattern.

Root-cause:

The root-cause for this pattern that developers do not handle all possible raised exceptions that may occur when users run the apps.

Example updates:

– An update of the *"Photosphere Free Wallpaper"*[31] app on June 27th 2014 has an issue in setting the scroll speed for users if the mode is auto scroll. The app code does not handle the case if the user does not provide scroll speed value. An emergency update is published on the next day (on June 28th 2014) to handle the exception by setting the scroll speed to a default value with release notes *"Fixed issue with auto scroll"*.
– An update of the *"SlenderMan LIVE"*[32] app on April 3rd 2014 has an issue in the code that opens the camera and sets the orientation of the displayed preview of the camera images. The app code does not handle the case of an exception occurring while opening and setting the camera preview. An emergency update is published on the next day (on April 4th 2014) to handle the exception of the un-handled cases with release notes *"Fixed upside down camera !"*.

[31] https://play.google.com/store/apps/details?id=fishnoodle.photospherewp_free

[32] https://play.google.com/store/apps/details?id=www.agathasmaze.com.slendermanlive

- An update of the *"First Aid Emergency & Home"*[33] app on December 8th 2013 has an issue in loading the In Case of Emergency (ICE) profile data. The issue occurs because the code does not handle the case of the user not providing all the requested information. An update is published on the next day to handle the null cases with release notes *"Fixed crash related ICE Profile"*.
- An update of the *"Flashlight + Clock"*[34] app on June 9th 2014 has an issue in setting the visibility of component as the code does not handle case if component is null. An update with version *"1.1.2"* is published on the next day to handle the null cases with release notes *"Version 1.1.2 Crash bug fixed"*.
- An update of the *"FeaturePoints: Free Gift Cards"*[35] app on April 30th 2014 has an issue in connecting to Google+ as the code does not handle the case of the user data being null. An update is published to handle the null cases with release notes *"Fixed crash when connecting to Google+"*.

Examples of user complaints:

We find that some user complaints about the app crashing but similar to the "Invoking unavailable APIs" pattern, it is difficult to be sure that these complaints are related to exceptions.

Speed of repair:

As shown in Table 8, this pattern requires a median of one day to repair. The short time to repair can be explained as this pattern causes the app to crash with the un-handled exceptions.

Lessons learned for developers:

Although un-handled exceptions may produce critical issues, research illustrates that un-handled exceptions are not often identified during code review. For example, Bacchelli and Bird (2013) study code review comments for Microsoft code. Bacchelli et al. find that only 4 % of code review comments are about handling exceptions. Developers need to perform more efficient code review mechanism and possibly automated tools in order to avoid the occurrence of this pattern.

We find 35 of the 54 updates that are related to this pattern are due to unhandled null pointer exceptions. There exists a slew of tools that can identify possibly unhandled null pointer exceptions (FindBugs 2016). Developers and store owners should make use of static analysis tools to avoid the need for emergency updates.

6 Limitations and Threats to Validity

In this section, we discuss the limitation and threats to validity of our findings.

[33] https://play.google.com/store/apps/details?id=com.nikolay.arfa

[34] https://play.google.com/store/apps/details?id=flashlight.led.clock

[35] https://play.google.com/store/apps/details?id=com.tapgen.featurepoints

6.1 Construct Validity

We select 1,000 emergency updates based on the emergency ratio that is defined in this paper. The emergency ratio depends on the number of days after the last update and the median update lifetime. As illustrated in Table 1, the lifetime for the updates preceding the emergency updates is less than or equals to 5 % of the median lifetime of the app. Such a low emergency ratio indicates that such updates are most likely emergency updates. However, some mobile apps may have an unstable update cycle. An update with a low emergency ratio may not be an actual emergency update. Including other factors in defining the emergency updates such as considering the standard deviation of the release cycle is another possible alternative definition of emergency updates.

We leverage a heuristic to identify emergency updates. The heuristic may not be 100 % accurate. There is a chance that a developer might release two updates back to back even though the second update is not urgently needed. However we feel the chances of such rapid updating is very low. For example, as illustrated in Table 2 we find one app (for the *"AutoZone"*[36] app) where the lifetime for the update preceding the emergency update is 21 days, while its median life time is 568 days. However, for this particular update, many issues are fixed. The release notes are as follows: *"Version 2.0.1 Multiple bug fixes, including: * Accurate product fitment notes. * AutoZone Rewards login issue. * Free Repair Guide images issue. * Improved accuracy with barcode scanning"*. Therefore, such an update is not included in our manual investigation of patterns of emergency updates.

We manually identify the patterns of emergency updates. Although we examine the decompiled apk file artifacts of the updates, the release notes and the user reviews, we are not experts in the development of these mobile apps. Our observation can be biased by our knowledge. To minimize the bias, the first two authors work together during the manual investigation. However, to further address this threat, interviews and users studies of such mobile apps are required to better understand the rationale for these emergency updates.

6.2 Internal Validity

Although we find eight patterns of emergency updates, not all studied emergency updates follow one of our eight patterns. In short, we do not consider our patterns as a comprehensive set of emergency updates patterns. Instead, our work is a first step in creating a richer and more complete set of such patterns. As more updates are examined, we expect that more patterns will emerge.

The key contribution of our study is to raise awareness about the fact that many of these emergency updates share common reasons and by documenting such patterns we hope to assist in improving the quality assurance processes for app updates.

6.3 External Validity

Our study is based on the 10,747 top free-to-download apps from the Google Play Store. Distimo lists the top apps from each category of apps for the US market. There might be other top apps in other areas of the world. Including other mobile app stores, such as iOS store, and including the top popular apps in different countries (not only the US) would

[36]https://play.google.com/store/apps/details?id=com.autozone.mobile

complement our study. Our results are based on running a Google Play Store crawler for 12 months. Our empirical study may be improved by including the mobile apps' data from a longer run period of our crawler. As any work that identifies patterns, our work is a first step towards creating a catalogue of such patterns. We expect that more patterns will emerge over the years. Although we find other issues that are fixed in emergency releases (e.g., we find one update that fixes the displayed text of a field), we did not formally document them as patterns since we find too few instances of them to claim a recurring pattern. Future studies should explore the generality of these patterns.

The Google Play Store crawler acts as a Samsung S3 device in order to download apk files. Some apps may have multiple apk files for different mobile devices. Therefore, other mobile devices may download different apk files from such apps. Crawling Google Play store with another device may complement our study.

The Google Play Store limits the number of retrieved user reviews for an app. Such a limitation is also noted by recent research by Martin et al. (2015). In our study we try to overcome this issue by running the Google Play Store crawler on a daily basis to ensure that we get as many reviews as possible (since the store will not return more than 500 new reviews since our last crawl).

Our study focuses on the top free-to-download apps since free-to-download apps are the majority of the apps in the Google Play Store. Moreover, free-to-download apps may not be totally free as some free-to-download apps include paid features through in-app purchases or subscriptions. Our study may benefit from including the top non-free apps and comparing the difference in the emergency updates of both free-to-download and non-free apps. However, one would need access to the apk files of these non-free apps by purchasing such apps, requiring a substantial amount of funds.

Our findings are based on a study of the top 1,000 emergency updates. The characteristics and patterns of the emergency updates that are not in the top 1,000 may be different from our findings. Our study may be improved by examining more emergency updates.

7 Related Work

In this section, we present prior research that is related to our work. In particular, we focus on prior research in the area of 1) rapid releases, 2) bugs in mobile apps, and 3) customer reviews of mobile apps.

7.1 Rapid Releases

Many organizations are moving from a traditional slower release cycle to a rapid release cycle (Khomh et al. 2015; Mäntylä et al. 2014; Souza et al. 2015). For example, Mozilla Firefox has shifted from releasing every 12 to 18 months to releasing every six weeks. Souza et al. (2015). Rapid releases enable the delivery of software in a shorter time and enables reacting rapidly to customer feedback (Mäntylä et al. 2014; Khomh et al. 2015). However, there is little knowledge about the impact of a faster release cycle on the quality of software. Thus, researchers have explored the impact of rapid releases on quality (Mäntylä et al. 2014; Khomh et al. 2012, 2015; Souza et al. 2014, 2015).

Khomh et al. (2012, 2015) study the impact of changing from a traditional long release cycle to a rapid release cycle in the Mozilla Firefox by comparing the number of post-release bugs, median uptime, and crash rates. Khomh et al. (2012, 2015) illustrate that the number of reported bugs per day in rapid releases does not change significantly in comparison to traditional long releases, while defects are fixed faster in rapid releases. Khomh et al. also find that users discover bugs faster because the program crashes more quickly than in traditional releases.

Mäntylä et al. (2014) study the testing of rapid releases for the Mozilla Firefox browser. The study collects different metrics that are related to the testing activities; such as the count of tests that are executed per day, the count of testers who are working in the project per day; they compare the testing activities within traditional releases to the testing activities within rapid releases. The study finds that with rapid releases developers test less compared to traditional releases.

Souza et al. (2014) study the impact of rapid releases on the software quality by comparing the bug reopening rate in traditional and rapid releases. They study traditional and rapid releases of Mozilla Firefox and find that rapid releases have a 7 % increase in the bug reopening rate than traditional releases. Hemmati et al. (2015) find that a risk-driven prioritization technique has a higher accuracy in prioritizing the test cases in rapid releases than other prioritization techniques.

All prior work on rapid releases is done on Mozilla Firefox, while we study the mobile apps in the Google Play Store. Moreover, we look at another type of release, which is very rapid by definition, i.e., emergency updates.

7.2 Bugs in Mobile apps

In order to minimize the bugs in mobile apps, prior research studies crashes and bugs in mobile apps. Guana et al. (2012) analyze the bugs data for 20,169 bugs in the Android platform repository. They study which layer in the Android framework contains more reported bugs. They find that the framework layer contains a higher number of bugs than the kernel layer. Han et al. (2012) study the Android platform bugs that are reported for the HTC and Motorola devices. They manually labeled the bugs and applied Latent Dirichlet Allocation (LDA) and Labeled LDA techniques in order to generate the topics from the bug reports. Han et al. identify 57 labels and 72 labels for the Motorola and HTC devices respectively. Han et al. find 14 common topics between the Motorola and HTC devices.

Syer et al. (2013) study the differences between mobile apps, desktop/server applications. Syer et al. study 15 open-source mobile apps (from the Google Play Store and F-Droid apps repositories) and five desktop/server applications. Syer et al. find that mobile apps have smaller code base than desktop/server applications. Syer et al. find that the reported bugs are fixed faster in mobile apps than in the desktop/server applications. Syer et al. (2015) study the relation between the use of Android APIs and the probability of having bug reports in the mobile apps. Syer et al. find that source code files that have a higher dependence on the Android APIs are more bug-prone than other files. Thus, Syer et al. recommended to prioritize code review efforts on source code files that heavily depend on the Android APIs.

Ravindranath et al. (2012) propose an *AppInsight* tool that analyzes the mobile app and identifies performance bottlenecks. Their study includes the analysis of the usage of 30 users of 30 apps over a four months period and leverages the AppInsight tool to identify the performance bottlenecks in these studied apps. Linares-Vásquez et al. (2013) study the impact of the change to mobile OS APIs and the fault-proneness of the used APIs on the app's rating, Linares-Vásquez et al. predict the app success based on the users rating and apply their study on 7,097 free-to-download Android apps. Their study finds a high correlation between the app's rating and the change and fault-proneness of the APIs that are used by the app, which means that the apps with high rating are more likely to use the APIs with lower change and fault-proneness.

The limited battery power of mobile devices may impact the user experience if some app features have a high battery consumption (e.g., the mobile camera) (Wan et al. 2015). Researchers study energy bugs and energy hotspots in mobile apps (Banerjee et al. 2014; Pathak et al. 2011, 2012; Wan et al. 2015). According to Banerjee et al. (2014) the energy bug is defined as the cases when the mobile device resources are still used although the app is no longer active, while energy hotspots are defined as the cases when the app causes high energy consumption although the resource utilization is small. Pathak et al. (2011) propose a taxonomy of smartphone energy bugs. Pathak et al. find that there is a variation in the types and causes of energy bugs. Pathak et al. provide a roadmap for developing a framework that identifies the root-cause of energy bugs. Banerjee et al. (2014) continue the study of energy bugs and energy hotspots and develop a framework that generates test data in order to detect energy bugs and energy hotspots in mobile apps. Wan et al. (2015) study the display energy hotspots in mobile apps. Their research identifies the screens that have more energy consumption than the optimized screen design. For example, the optimized colors may reduce energy consumption. Wan et al. rank the app screens with respect to the difference in the energy consumption between the original screens and the energy-optimized screens, so developers can focus on top energy consuming screens to reduce the consumed energy of apps.

Hecht et al. (2015) study anti-patterns in mobile apps. Hecht et al. analyze the source code from 3,568 updates of 106 apps in the Google Play Store. Hecht et al. identify seven anti-patterns in mobile apps. The anti-patterns belong to two groups: object oriented anti-patterns and Android framework anti-patterns. The findings from Hecht et al. show that mobile app developers need to allocate more quality assurance efforts.

The difference between our work and the prior work related to bugs in mobile apps is that the our identified patterns are different from traditional bugs in the source code. Our identified issues are typically introduced by simple developer mistakes.

7.3 Customer Reviews of Mobile apps

User reviews have been used as new source of data to capture the perceived quality of an app. For example, McIlroy (2014), McIlroy et al. (2015) collect the customer reviews from over 10,000 of the top free-to-download apps across all the app categories in the Google Play Store, and propose an approach to label user reviews automatically using 14 different labels that capture the user's perceived quality of an app.

Guzman and Maalej (2014) study customer reviews for mobile apps by extracting the app features from the customer reviews. They apply sentiment analysis to quantitative the reviews, and then group the extracted app features into high level topics using LDA techniques. Guzman et al. study the customer reviews for seven apps from both the Google Play Store and the Apple Store. Their study proposes an approach for identifying the app features and general topics from customer reviews. They evaluate the identified topics through a manual analysis process and their approach has a 59 % precision and a 51 % recall.

Khalid et al. (2015, 2014) analyze the customer reviews in the Apple Store and identify 12 types of user complaints. They study the devices that have the most reported complaints in order to prioritize the quality assurance resources for these devices.

Khalid et al. (2015) apply FindBugs on 10,000 Android Apps and study the relationship between the apps' ratings and the warnings identified by the FindBugs tool. They find that some FindBugs warnings (such as ‘Performance’, ‘Bad Practice’ and ‘Internationalization’) occur more significantly in the apps with low rating.

Gui et al. (2015) study the hidden cost in mobile apps by analyzing the impact of using advertisement network on the performance, energy, network, maintenance, and reviews of the mobile app. Gui et al. study 21 different mobile apps of the Google Play Store, and they analyze the customer review during the study period from January 2014 to August 2014. The result of the study shows that more than 50 % of the studied apps contain at least 3.28 % of one and two stars reviews that mention advertisement in the review text. The study shows that the hidden cost of using advertisement networks can affect the apps rating.

Harman et al. (2012) study the relationship between the app rating, the rank for the number of users who downloaded the app, and the price of the app. They collect data for 32,108 apps from the Blackberry App Store. Their study shows that there is a strong correlation between the app rating and the app downloads and there is no correlation between the app price and the app rating or the app downloads.

Maalej and Nabil (2015) proposed an automated approach to classify customer reviews into four categories: bug, feature request, rating and user experience. Maalej et al. leverage data that are extracted from reviews, such as reviews rating, reviews length, sentiment analysis and bag of words, to classify the reviews.

Prior work proposes automated analyses of the reviews without mapping the reviews to particular updates. Instead, we manually investigate those reviews that are associated with each studied emergency update (U_i) and the preceding update (U_{i-1}).

8 Conclusions

Mobile app stores provide an update mechanism that enables app developers to rapidly publish new updates to their users in a cost effective manner. Developers leverage this mechanism to publish emergency updates that are published soon after the previous update.

In this paper, we study emergency updates in the Google Play Store by analyzing more than 44,000 updates based on around a year of monitoring the update activities of over 10,000 of the top free-to-download apps in the store.

By analyzing the top 1,000 emergency updates, we find that:

1. The emergency updates are often updates with a long lifetime (i.e., they are rarely followed by another emergency update). Users should update their apps when there is an emergency update without being concerned about another update showing up soon afterwards.
2. Emergency updates rarely include a description in their release notes about the rationale for such an update.
3. The updates preceding the emergency updates receive a higher ratio of negative reviews than the emergency updates.

We identify eight patterns of emergency updates across two categories, updates due to deployment issues and updates due to source code changes. For each pattern, we document the description, root-causes, example updates, examples of user complaints, speed of repair, and the takeaway from this pattern for users, developers, researchers and app store owners. Our findings can help developers and app store owners avoid emergency updates in order to improve the quality and user satisfaction of their apps.

Our study is a first step in creating a rich catalogue of patterns of emergency updates. Future studies should explore additional emergency updates in order to augment our identified patterns.

References

ABI Research Android Will Account for 58% of Smartphone App Downloads in 2013, with iOS Commanding a Market Share of 75% in Tablet Apps. https://www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down/ (Last accessed March 2016)

Akdeniz (2013) Akdeniz: Google play crawler. https://github.com/Akdeniz/google-play-crawler

Alvarez H (2016) A guide to color, ux, and conversion rates. http://www.usertesting.com/blog/2014/12/02/color-ux-conversion-rates/. Accessed March 2016

Android Studio (2016) Managing projects overview - android developers. https://developer.android.com/tools/projects/index.html. Accessed March 2016

Apktool (2016) Apktool - a tool for reverse engineering android apk files. http://ibotpeaches.github.io/Apktool/. Accessed March 2016

AppBrain (2016) Free versus paid android apps. http://www.appbrain.com/stats/free-and-paid-android-applications. Accessed March 2016

Apple Developer (2016) iOS human interface guidelines: Color and typography. https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ColorImagesText.html. Accessed March 2016

Arakelyan A (2016) Which color is right for your mobile app icon. https://www.iconsmind.com/color-right-mobile-app-icon/. Accessed March 2016

Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. In: 35th international conference on software engineering, ICSE '13, San Francisco, pp 712–721. http://dl.acm.org/citation.cfm?id=2486882

Banerjee A, Chong LK, Chattopadhyay S, Roychoudhury A (2014) Detecting energy bugs and hotspots in mobile apps. In: Proceedings of the 22Nd ACM SIGSOFT international symposium on foundations of software engineering, FSE 2014. ACM, New York, pp 588–598

Bavota G, Vásquez ML, Bernal-Cárdenas CE, Penta MD, Oliveto R, Poshyvanyk D (2015) The impact of API change- and fault-proneness on the user ratings of android apps. IEEE Trans Software Eng 41(4):384–407

CFR (2016) Cfr - another java decompiler. http://www.benf.org/other/cfr/. Accessed March 2016

Colblindor (2016) Coblis color blindness simulator. http://www.color-blindness.com/coblis-color-blindness-simulator/. Accessed March 2016

Color Oracle (2016) Color oracle, design for the color impaired. http://colororacle.org. Accessed March 2016

Dex2jar (2016) Dex2jar download - sourceforge.net. http://sourceforge.net/projects/dex2jar/. Accessed March 2016

Distimo (2013) Google play store, united states, top overall, free, week 35 2013. http://www.distimo.com/leaderboards/google-play-store/united-states/top-overall/free

F-Droid (2016) F-droid. http://f-droid.org/. Accessed March 2016

Felt AP, Chin E, Hanna S, Song D, Wagner D (2011) Android permissions demystified. In: Proceedings of the 18th ACM conference on computer and communications security, CCS '11. ACM, New York, pp 627–638

FindBugs (2016) Findbugs^{TM} - find bugs in java programs. http://findbugs.sourceforge.net. Accessed March 2016

Gehan EA (1965) A generalized wilcoxon test for comparing arbitrarily singly-censored samples. Biometrika 52(1–2):203–223

Google Android (2016a) Uses-sdk android developers. http://developer.android.com/guide/topics/manifest/uses-sdk-element.html. Accessed March 2016

Google Android (2016b) Uses-feature, android developers. http://developer.android.com/guide/topics/manifest/uses-feature-element.html. Accessed March 2016

Google Android (2016c) Permission, android developers. http://developer.android.com/guide/topics/manifest/permission-element.htmlhttp://developer.android.com/guide/topics/manifest/permission-element.html. Accessed March 2016

Google Android (2016d) App manifest, android developers. http://developer.android.com/guide/topics/manifest/manifest-intro.html. Accessed March 2016 mobiThinking (2016) Global mobile statistics 2013 section e: Mobile apps, app stores, pricing and failure rates. http://mobiforge.com/research-analysis/global-mobile-statistics-2013-section-e-mobile-apps-app-stores-pricing-and-failure-rates?mT. Accessed March 2016

Google Support (2016) Update your apps - developer console help. https://support.google.com/googleplay/android-developer/answer/113476?hl=en. Accessed March 2016

Gorla A, Tavecchia I, Gross F, Zeller A (2014) Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. ACM, New York, pp 1025–1035

Guana V, Rocha F, Hindle A, Stroulia E (2012) Do the stars align? Multidimensional analysis of android's layered architecture. In: 9th IEEE working conference of mining software repositories, MSR 2012, Zurich, pp 124–127

Gui J, Mcilroy S, Nagappan M, Halfond WGJ (2015) Truth in advertising: the hidden cost of mobile ads for software developers. In: 37th International Conference on Software Engineering, ICSE '15, Florence

Guzman E, Maalej W (2014) How do users like this feature? A fine grained sentiment analysis of app reviews. In: IEEE 22nd international requirements engineering conference, RE 2014, Karlskrona, pp 153–162

Han D, Zhang C, Fan X, Hindle A, Wong K, Stroulia E (2012) Understanding android fragmentation with topic analysis of vendor-specific bugs. In: 19th working conference on reverse engineering, WCRE 2012, Kingston, pp 83–92

Harman M, Jia Y, Zhang Y (2012) App store mining and analysis: MSR for app stores. In: 9th IEEE working conference of mining software repositories, MSR 2012, Zurich, pp 108–111

Hassan AE, Holt RC (2004) Predicting change propagation in software systems. In: 20th international conference on software maintenance (ICSM 2004), Chicago, pp 284–293. doi:10.1109/ICSM.2004.1357812

Hecht G, Omar B, Rouvoy R, Moha N, Duchien L (2015) Tracking the software quality of android applications along their evolution. In: Grunske L, Whalen M (eds) 30th IEEE/ACM international conference on automated software engineering. Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015). IEEE, Lincoln, p 12. https://hal.inria.fr/hal-01178734

Hemmati H, Fang Z, Mantyla MV (2015) Prioritizing manual test cases in traditional and rapid release environments. In: 8th IEEE international conference on software testing, verification and validation, ICST 2015, Graz, pp 1–10

Houston P (2016) Store and use files in assets. https://xjaphx.wordpress.com/2011/10/02/store-and-use-files-in-assets/. Accessed March 2016

Hyzap (2016). https://www.heyzap.com. Last accessed March 2016

Stuart Dredge (2016) Information commissioner's office releases app privacy guidelines. http://www.theguardian.com/technology/2013/dec/19/information-commissioners-office-app-privacy-guidelines. Accessed March 2016

Joorabchi ME, Mesbah A, Kruchten P (2013) Real challenges in mobile app development. In: 2013 ACM / IEEE international symposium on empirical software engineering and measurement, Baltimore, pp 15–24

Khalid H, Nagappan M, Hassan AE (2015) Examining the relationship between findbugs warnings and end user ratings: a case study on 10,000 android apps. IEEE Soft 99

Khalid H, Nagappan M, Shihab E, Hassan AE (2014) Prioritizing the devices to test your app on: a case study of android game apps. In: Proceedings of the 22Nd ACM SIGSOFT international symposium on foundations of software engineering, FSE 2014. ACM, New York, pp 610–620

Khalid H, Shihab E, Nagappan M, Hassan AE (2015) What do mobile app users complain about? IEEE Soft 32(3):70–77

Khomh F, Adams B, Dhaliwal T, Zou Y (2015) Understanding the impact of rapid releases on software quality - the case of firefox. Empir Softw Eng 20(2):336–373

Khomh F, Dhaliwal T, Zou Y, Adams B (2012) Do faster releases improve software quality?: An empirical case study of mozilla firefox. In: Proceedings of the 9th IEEE working conference on mining software repositories, MSR '12. IEEE Press, Piscataway, pp 179–188

Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, Di Penta M, Oliveto R, Poshyvanyk D (2013) API change and fault proneness: a threat to the success of android apps. In: Proceedings of the 2013 9th joint meeting on foundations of software engineering, ESEC/FSE 2013. ACM, New York, pp 477–487

Maalej W, Nabil H (2015) Bug report, feature request, or simply praise? On automatically classifying app reviews. In: 23rd IEEE international requirements engineering conference, RE 2015, Ottawa, pp 116–125. doi:10.1109/RE.2015.7320414

Mäntylä MV, Khomh F, Adams B, Engström E, Petersen K (2013) On rapid releases and software testing. In: Proceedings of the 2013 IEEE international conference on software maintenance, ICSM '13. IEEE Computer Society, Washington, pp 20–29

Mäntylä MV, Adams B, Khomh F, Engström E, Petersen K (2014) On rapid releases and software testing: a case study and a semi-systematic literature review. Empir Softw Eng:1–42

Martin W, Harman M, Jia Y, Sarro F, Zhang Y (2015) The app sampling problem for app store mining. In: 12th IEEE/ACM working conference on mining software repositories, MSR 2015, Florence, pp 123–133. doi:10.1109/MSR.2015.19

Martin Poschenrieder (2016) 77% will not download a Retail app rated lower than 3 stars. http://blog.testmunk.com/77-will-not-download-a-retail-app-rated-lower-than-3-stars/. Accessed March 2016

McIlroy S (2014) Empirical studies of the distribution and feedback mechanisms of mobile app stores. Master's thesis, Queen's University

McIlroy S, Ali N, Khalid H, Hassan AE (2015) Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. Empir Softw Eng

Pandita R, Xiao X, Yang W, Enck W, Xie T (2013) Whyper: Towards automating risk assessment of mobile applications. In: Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), pp 527–542. USENIX, Washington

Pathak A, Hu YC, Zhang M (2011) Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In: Tenth ACM Workshop on Hot Topics in Networks (HotNets-X), HOTNETS '11, Cambridge, p 5

Pathak A, Jindal A, Hu YC, Midkiff SP (2012) What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In: The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12, Ambleside, pp 267–280

Ravindranath L, Padhye J, Agarwal S, Mahajan R, Obermiller I, Shayandeh S (2012) Appinsight: mobile app performance monitoring in the wild. In: 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, pp 107–120

Robotium (2016). https://code.google.com/p/robotium/. Accessed March 2016

Ruiz IJM, Adams B, Nagappan M, Dienst S, Berger T, Hassan AE (2014) A large-scale empirical study on software reuse in mobile apps. IEEE Software 31(2):78–86. doi:10.1109/MS.2013.142

Ruiz IJM, Nagappan M, Adams B, Hassan AE (2012) Understanding reuse in the android market. In: IEEE 20th international conference on program comprehension, ICPC 2012, Passau, pp 113–122. doi:10.1109/ICPC.2012.6240477

Simon VT (2016) What API level should I target?. http://simonvt.net/2012/02/07/what-api-level-should-i-target/. Accessed March 2016

Smali2java (2016). http://www.hensence.com/en/smali2java/. Last accessed March 2016

Smith J (2016) How to use the psychology of color to increase website conversions. https://blog.kissmetrics.com/psychology-of-color-and-conversions/. Accessed March 2016

Sofia (2016) Chapter 2 structure of an android app. http://sofia.cs.vt.edu/sofia-2114/book/chapter2.html. Accessed March 2016

Souza R, von Flach G, Chavez C, Bittencourt RA (2014) Do rapid releases affect bug reopening? A case study of firefox. In: 2014 Brazilian symposium on software engineering, Maceió, pp 31–40

Souza R, von Flach G, Chavez C, Bittencourt RA (2015) Rapid releases and patch backouts: a software analytics approach. IEEE Software 32(2):89–96

Stackoverflow (2016a) Remove extra unwanted permissions from manifest android, stackoverflow. http://stackoverflow.com/questions/8257412/remove-extra-unwanted-permissions-from-manifest-android. Accessed March 2016

Stackoverflow (2016b) How to check if android permission is actually being used?, stackoverflow. http://stackoverflow.com/questions/24858462/how-to-check-if-android-permission-is-actually-being-used. Accessed March 2016

Stackoverflow (2016c) Storage limit in raw and asset folder in android. http://stackoverflow.com/questions/14995756/storage-limit-in-raw-and-asset-folder-in-android. Accessed March 2016

Stackoverflow (2016d) Difference between /res and /assets directories. http://stackoverflow.com/questions/5583608/difference-between-res-and-assets-directories. Accessed March 2016

Stackoverflow (2016e) Clean up unused android permissions, stackoverflow. http://stackoverflow.com/questions/18362305/clean-up-unused-android-permissions. Accessed March 2016

Statista (2016a) Number of apps available in leading app stores as of July 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Accessed March 2016

Statista (2016b) Google play: number of downloads 2010-2013. http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/. Accessed March 2016

Syer MD, Nagappan M, Hassan AE, Adams B (2013) Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source android apps. In: Center for advanced studies on collaborative research, CASCON '13, Toronto, pp 283–297. http://dl.acm.org/citation.cfm?id=2555553

Syer MD, Nagappan M, Adams B, Hassan AE (2015) Studying the relationship between source code quality and mobile platform dependence. Softw Qual J 23(3):485–508. doi:10.1007/s11219-014-9238-2

Telerik (2016) Extra android permissions always set. http://www.telerik.com/forums/extra-android-permissions-always-set. Accessed March 2016

Wan M, Jin Y, Li D, Halfond WGJ (2015) Detecting display energy hotspots in android apps. In: 8th IEEE international conference on software testing, verification and validation, ICST 2015, Graz, pp 1–10

Wikipedia (2016) Wikipedia Patch Tuesday. http://en.wikipedia.org/wiki/Patch_Tuesday. Accessed March 2016

Xu W, Zhang F, Zhu S (2013) Permlyzer: analyzing permission usage in android applications. In: IEEE 24th international symposium on software reliability engineering, ISSRE 2013, Pasadena, pp 400–410. doi:10.1109/ISSRE.2013.6698893

Zimmermann T, Weißgerber P, Diehl S, Zeller A (2004) Mining version histories to guide software changes. In: 26th international conference on software engineering (ICSE 2004), pp 563–572, Edinburgh. doi:10.1109/ICSE.2004.1317478

Safwat Hassan is a PhD student at School of Computing, Queens University. Safwat worked as a software engineer for ten years in different corporations like Egyptian Space Agency (ESA), HP, EDS, VF Germany (outsourced by HP), and Etisalat. During his ten years work experience, he worked on different large-scale systems (varying from Web-Based systems to embedded systems) and in diverse project types (design service, customer support, and R&D) across various domains (Telecommunication, Supply-chain, and Aerospace). His main interest is bigdata analytics, software engineering, mobile app store dynamics. Contact him at shassan@cs.queensu.ca.

Weiyi Shang is an assistant professor in the Department of Computer Science and Software Engineering at Concordia University, Montreal. His research interests include big-data software engineering, software engineering for ultra-large-scale systems, software log mining, empirical software engineering, and software performance engineering. Shang received a PhD in computing from Queen's University, Canada. Contact him at shang@encs.concordia.ca.

Ahmed E. Hassan is the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. Hassan also serves on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, Springer Journal of Computing, and Peer J Computer Science. Contact him at ahmed@cs.queensu.ca.