Noname manuscript No.
(will be inserted by the editor)

An Empirical Study on Release Notes Patterns of Popular Apps in the Google Play Store

**Aidan Z.H. Yang · Safwat Hassan ·
Ying Zou · Ahmed E. Hassan**

Received: date / Accepted: date

Abstract Release notes of a new mobile release provide valuable information for app users about the updated functionality of an app. Moreover, app developers can use the release notes to inform users about the resolution of a previously reported issue in user reviews. Prior work shows that release notes are an essential artifact for app developers to announce the emergency fixes and the newly adopted features. However, little is known about the common practices adapted by developers in preparing release notes in order to increase the ratings of apps.

In this paper, we are interested in capturing the common practices as release notes patterns. We aim to understand how to leverage the release notes to improve the perceived quality of their apps. We study release notes patterns by analyzing 69,851 releases and 67.7 million user reviews of 2,232 top free-to-download apps in the Google Play Store over three years (from April 2016 until April 2019). We observe that app developers tend to write either long release notes (over 50 words) or short release notes (less than 7 words).

We use the characteristics of release notes, such as the number of words, to identify six patterns of release notes in mobile apps. We find that apps with longer release notes tend to have higher average user ratings. Furthermore, we observe that a shift from rarely updated patterns to frequently updated patterns tend to have higher average user ratings.

Aidan Z.H. Yang · Ying Zou
Software Evolution and Analytics Lab (SEAL)
Queen's University
Kingston, Ontario, Canada
E-mail: {a.yang, ying.zou}@queensu.ca

Safwat Hassan · Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, Ontario, Canada
E-mail: {shassan, ahmed}@cs.queensu.ca

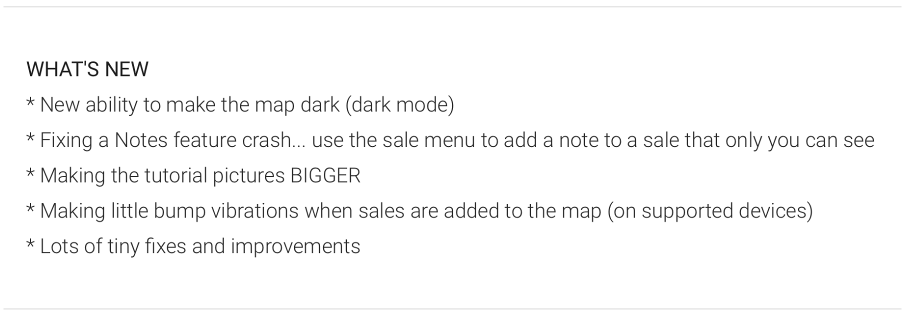

WHAT'S NEW
* New ability to make the map dark (dark mode)
* Fixing a Notes feature crash... use the sale menu to add a note to a sale that only you can see
* Making the tutorial pictures BIGGER
* Making little bump vibrations when sales are added to the map (on supported devices)
* Lots of tiny fixes and improvements

Fig. 1: An example of the release notes of the *"Yard Sale Treasure Map"* app (the release version *"7.6.0"*).

Keywords Android mobile apps · Release notes · Google Play Store · Longitudinal study

1 Introduction

Mobile apps have become an integral part of our daily activities, and are used for different purposes, such as playing games, performing social communications, and running financial and educational services. To fulfill the evolving demand of the user-base of an app, developers need to frequently deploy new releases that introduce new features and fix the previously reported issues in the prior releases. A recent study on mobile app markets reports that there are thousands of daily releases of mobile apps in the Google Play Store in 2018 [35].

To announce the changes in a new release, app stores (such as the Google Play Store and the Apple App Store) enable developers to write the *release notes* of the new releases. In particular, the release notes of a new release should describe the changes in the latest release, the reason for the changes, and the instructions for using the changes [4, 9]. Descriptive release notes are important for app developers as it enables developers to help app users better understand how to use the newly introduced features and whether to install the new release [4]. Hence, app stores require app developers to describe the updates in their release notes [9].

Figure 1 shows an example of the release notes of the *"Yard Sale Treasure Map"* app. As shown in Figure 1, developers of the *"Yard Sale Treasure Map"* app use the release notes to describe the newly added features to app users. Moreover, developers use the release notes to describe the resolved issues in the new release. For example, the release notes of the version *"15.9.1"* of the *"FanFiction.Net"* app mention the resolution of the Application Not Responding (ANR) issue that was reported in the prior release (*"Fixed potential ANR when app returns from idle on Android 9"*).

Table 1: The mean and five-number summary of the median time-to-release (days) of all releases of an app.

Mean	**Min**	**1st Qu.**	**Median**	**3rd Qu.**	**Max**
41.1	1.0	13.0	22.0	41.0	888.0

Prior research shows that release notes are essential artifacts for app developers to announce the resolution of emergency issues and the implementation of the requested features [7, 13, 14]. Existing research also relates information of release notes with the success of an app [21, 29]. For example, Martin *et al.* [21] perform a causal analysis between the release notes and the app rating. Martin *et al.* find that the content of the release notes (e.g., release notes that contain *"bug"*, *"fix"*, or *"feature"* words) has an impact on user ratings.

Despite the integral role of release notes in the mobile app market, to the best of our knowledge, no prior work performs an in-depth analysis of the *release notes patterns* that capture the common release notes practices in mobile apps and how these patterns are associated with the user-perceived quality of an app. We aim to identify the release notes patterns that correlate the most with high user-perceived quality of an app. Our study can help app developers better leverage the release notes to improve the perceived quality of their apps, and recommend the appropriate release patterns to follow. In addition, gaining a solid foundation about release notes patterns can help researchers understand limitations, potential challenges, and future research directions for analyzing release notes of mobile apps.

In this paper, we identify release notes patterns of popular apps in the Google Play Store. In particular, we analyze 69,851 releases and 67.7 million user reviews of 2,232 top free-to-download apps in the Google Play Store over three years (from April 2016 until April 2019). To motivate our study, we analyze whether developers of the studied apps actively update their apps by measuring the *time-to-release* of all the collected 69,851 releases. We measure the time-to-release of a release R_i as the time between current release R_i and immediate prior release R_{i-1}. Finally, we measure the median time-to-release of all releases of an app.

We find that developers actively release new releases. In particular, we observe that the majority (75%) of the studied apps have a median time-to-release of under 41 days (Table 1), which indicates that the studied apps are actively maintained by their frequent new releases. Some apps have very frequent releases, so median time-to-release is one day. For example, the *"slither.io"* app has five patch releases, with each one day apart. On the other hand, apps in the high extreme (above 600 days) have only three or fewer total releases during the studied period, with each over a year apart. For example, the *"Monogram It!"* app has two releases: the first in February 2017, and the next in December 2018. The low frequency of releases may indicate that this app is not actively maintained by its developers.

Our motivational study shows that the selected popular apps are actively updated during the studied period. In this work, our objective is to learn from the developers of these popular apps about how the release notes are used to improve the ratings of apps. In particular, we aim to identify different patterns of release notes and evaluate the impact of the release notes patterns on user-perceived quality by addressing the following three research questions:

RQ1: *What are the release notes update patterns?*
We explore the different release notes patterns to understand how developers write release notes to achieve higher user-perceived quality. We find that app developers leverage release notes differently in terms of the length of release notes and reusing their prior releases. We cluster apps based on their *content richness* and *updatability* to find different patterns of writing release notes. We measure *content richness* by the number of words in each release note. We assess *updatability* by computing the cosine similarity between each release note and all prior release notes of the same app. We identify six patterns of release notes. Some patterns have low content richness (e.g., pattern 1 *short low-updatability steady*), while some patterns have high updatability only during major releases (e.g., pattern 6 *long rising-updatability with major releases*).

RQ2: *What are the characteristics of the apps that follow a certain release notes pattern?*
We aim to understand if a particular release notes pattern correlates to high user-perceived quality in order to help developers follow an appropriate pattern. To achieve this objective, we build six logistic regression models (i.e., one model for each identified pattern in RQ1). We use 11 app features (e.g., app size, app category, and app rating increase) as our independent variables. We observe that the apps clustered into each pattern generally have different app features. For example, we find that apps in pattern 5, *long updating steady*, includes apps with long and steadily changed release notes. Apps in pattern 5 generally have fewer releases and higher user-perceived ratings. Furthermore, apps in pattern 6, *long rising-updatability with major releases*, covers apps with long release notes that update mostly during major releases. Such pattern has higher developer response rates. Therefore, our approach can help developers write release notes that are highly correlated to high user ratings (i.e., release notes in pattern 5). Our result also shows that developers can broadcast responses to users using informative release notes (i.e., release notes in pattern 6).

RQ3: *What causes developers to shift their release notes pattern?*
To investigate why some apps change their release notes pattern and if the changes can lead to higher user-perceived app quality, we perform a qualitative analysis on the dataset. We first use stationary analysis [5] on the characteristics of every app (e.g., *content richness* and *updatability*) and identify 53 apps that show a shift from one pattern to another. We then manually examine the release notes of these 53 app. We observe that 34% of the studied apps shift their release notes pattern due to a change

from listing functionality to providing user guidance (e.g., from describing new app functions to teaching users how to use existing features). We also find that most of the apps shifting from short to long and non-updating to updating release notes exhibit an increase in user ratings.

The main contributions of this paper are as follows:

1. Our paper is the first work to perform a large-scale analysis of the release notes practices of top free-to-download apps in the Google Play Store. We observe that app developers have different release patterns concerning *content richness* (i.e., being descriptive) or *updatability* (frequently updating their release notes). However, the majority of app developers (59% of the studied apps) tend to reuse their prior release notes.
2. We identify six patterns of release notes practices in mobile apps, and we identify the patterns that most strongly correlate to high user-perceived app quality.
3. We analyze the shifts between patterns and how these shifts are correlated to app ratings. Our work can help developers make informed decisions on which release notes patterns to follow.

Paper Organization: The rest of the paper is organized as follows. In Section 2, we outline the experiment setup. Section 3 presents the results with respect to our three research questions. Section 4 presents the implications we draw from the results. Section 5 discusses threats to the validity of our results. In Section 6, we examine related work. Finally, Section 7 draws conclusions to the paper.

2 Experimental Setup

In this section, we outline our data collection and data processing steps. Figure 2 gives an overview of our approach for collecting the release notes and user reviews of the studied apps and analyzing the collected data.

2.1 Data Collection

Step 1: Select top Android Apps. In our study, we focus on popular apps as developers of these apps need to actively maintain their apps by regularly updating their apps to satisfy the needs of their user-base [28]. We collect the list of top popular apps using the App Annie report [1]. We select the top 100 apps in each of the available 28 app categories (e.g., games and tools categories) in the Google Play store. We find that 214 apps are repeated across app categories, and 354 apps are already removed from the store at the start of our studied period. In total, we select 2,232 apps for our study.
Step 2: Crawl app data. We ran a crawler that is customized based on the Akdeniz [3] Google Play crawler to collect data of the selected 2,232 apps. In particular, we collect the following data for each studied app:

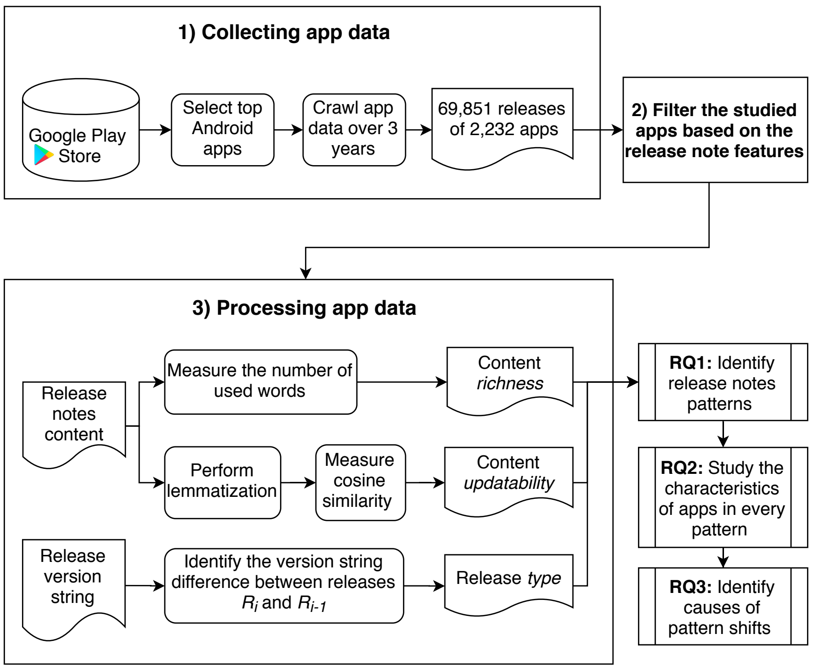

1) Collecting app data
Google Play Store
Select top Android apps
Crawl app data over 3 years
69,851 releases of 2,232 apps
2) Filter the studied apps based on the release note features
3) Processing app data
Release notes content
Measure the number of used words
Content *richness*
Perform lemmatization
Measure cosine similarity
Content *updatability*
Release version string
Identify the version string difference between releases R_i and R_{i-1}
Release *type*
RQ1: Identify release notes patterns
RQ2: Study the characteristics of apps in every pattern
RQ3: Identify causes of pattern shifts

Fig. 2: An overview of our approach for collecting, filtering, and processing release notes data.

1. App metadata: General data that gives insights on the characteristics of each app (e.g., app title, app description, the number of downloads, and app ratings).
2. App releases: For each release, we collect the release date, release notes content, and the release version string (e.g., 4.2.3).
3. User-developer dialogues: We collect all reviews and developers' responses for each app.

We collected data of the studied 2,232 apps over the period of three years from April 2016 to April 2019. In particular, we collect 69,851 releases with 67,697,919 user reviews and 2,902,944 developer responses.

2.2 Data Filtration

We filter the initial 2,232 apps to ensure that all studied apps have sufficient data for further analysis. First, we remove apps with fewer than three release notes in the three-year studied period as apps with only one or two releases do not contain sufficient data to be accurately clustered into a meaningful release notes pattern (RQ1). We also remove apps with nontraditional version strings, such as four numbers separated by dots (e.g., *"2.24.3.4"*). Since we cannot use semantic versioning (i.e., version strings represented by three numbers separated by dots), we cannot determine the release type of nontraditional version strings. As the result of the data filtration steps, our dataset contains 58,069 releases of 1,712 apps.

2.3 Data Processing

In this section, we describe the processing steps to identify the release type (i.e., major, minor, or patch release) and deriving app features that capture the releasing practice of an app.

2.3.1 Identifying the Release Type for the Studied Releases

To identify the release type for the studied releases, we use semantic versioning [32] to classify the studied releases into three types: major, minor, and patch. According to release practice guidelines, major releases occur when developers make incompatible API changes, minor releases occur when developers add functionality in a backward-compatible manner, and patch releases occur when developers make backward-compatible bug fixes [32]. We determine the release type for each release by comparing the version string of the current release R_i and the version string of the prior release R_{i-1}. A version string consists of three numbers separated by dots/periods ($x_i.y_j.z_k$). A major, minor, and patch release occur when the first, second, and third number changes, respectively. ($x_{i+1}.y_j.z_k$ is a major release, $x_i.y_{j+1}.z_k$ is a minor release, and $x_i.y_j.z_{k+1}$ is a patch release). In the case of multiple numbers changing, then major dominates minor/patch, and minor dominates patch ($x_{i+1}.y_{j+1}.z_{k+1}$ is a major release) [32].

2.3.2 Identifying the Release Notes Features

To understand the usage of release notes in practice, we extract the following release notes features that are used in RQ1.

1. **Content richness:** The content richness of the release notes represents the descriptiveness of the release notes text. In particular, we measure the content richness of the release notes by counting the number of words in the release notes text.
2. **Content updatability:** The content updatability of the release notes represents how frequently the release notes content is updated. To quantify the content updatability of the release notes for a release R_i, we measure the *cosine similarity* of the unique keywords between R_i and all prior releases. We choose to include all prior releases by considering the cases in which developers repeat release notes before the prior release R_{i-1}.
   To measure the cosine similarity, we remove punctuation and perform lemmatization on each word of every release note. Lemmatization is the process that converts all words of a sentence into its base form. For example,

the lemmatization of "fixing" is "fix" [31]. We use Stanford CoreNLP [19] for the lemmatization process. Cosine similarity is a value between 0 and 1, where 0 signifies no similarity and 1 signifies identical release notes between the consecutive releases R_i and all prior releases. We exclude the first release of each app (R_1) as it does not have a prior release to calculate the cosine similarity for R_1.
Finally, we calculate the *content updatability* of the release notes as the inverse of the cosine similarity (i.e., 1 - cosine similarity for all previous release notes). Content updatability is a measure of how *varied* the release notes are in comparison to all prior release notes of an app. A content updatability close to 1 signifies a high degree of release notes change, and a content updatability close to 0 represents a low degree of release notes change.

3. **Content updatability for major releases:** We observe that major releases tend to have the most updated release notes compared to the minor and patch releases (Figure 3a). For minor and patch releases, the release notes are frequently updated after major releases (Figure 3b). From the two observations regarding major releases, we use content updatability for only major releases as a feature to identify release note patterns.

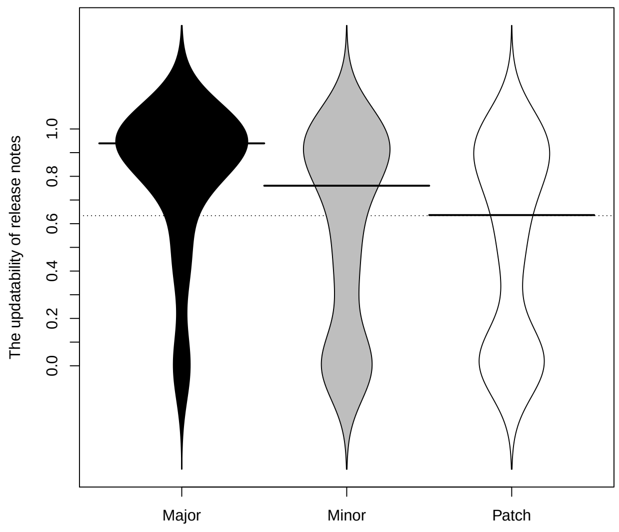
To ensure that the content richness, updatability, and updatability measured only for major releases provide different insights into release notes styles, we measure the correlation between the extracted features using the Spearman rank correlation coefficient [23]. We find that content richness and updatability have a correlation of 0.079 and a p-value of 2.1×10^{-73}. We find that updatability for all releases and updatability measured only for major relases have a correlation of 0.23 and a p-value of 3.4×10^{-4}. Finally, we find updatability measured only for major releases and content richness have a correlation of 0.094 and a p-value of 7.4×10^{-54}. Therefore, the three measured features are not correlated to each other, and we used the three measured features to identify the release notes patterns in mobile apps.

3 Results

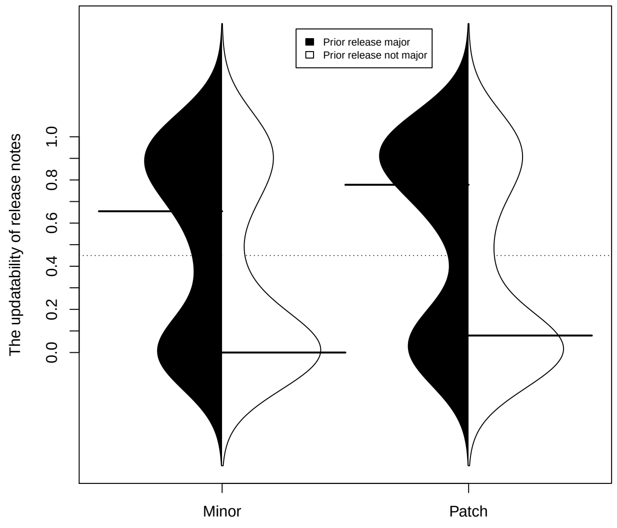
In this section, we present our study in terms of four research questions. For each research question, we discuss the motivation, approach, and the obtained results.

RQ1: What are the release notes update patterns?

Motivation: The main objective of this work is to identify release notes patterns and to understand the patterns that correlate to high user-perceived quality. Hence, we can help developers better understand how to write the release notes in order to improve the perceived quality of their apps. To achieve



(a) The distribution of updatability by the release type



(b) The distribution of updatability for minor and patch releases

Fig. 3: The updatability of the studied release notes (grouped by the release types).

this goal, in this RQ, we aim to identify the most common release notes patterns using the characteristics (e.g., the release notes length) of the studied release notes.

Approach: Our approach for investigating release notes patterns involves the following three steps.

Table 2: A summary of the six identified patterns.

Pattern ID	Pattern Name	Number of Apps	Length	Updatability	Updatability for Major Releases [1]
1	Short non-updating steady	439	11.5%	19.8%	16%
2	Short updating steady	111	16.6%	81.1%	NA
3	Short rising-updatability with major releases	157	31.0%	39.7%	97.1%
4	Long non-updating steady	572	81.0%	27.5%	18.4%
5	Long updating steady	145	92.9%	74.9%	NA
6	Long rising-updatability with major releases	288	85.2%	42.8%	96.5%

Step 1: Deriving app features that capture the releasing practice of an app. We use three features (i.e., richness, updatability, and updatability for only major releases) to capture the characteristics of release notes practice of an app (as described in Section 2).
Step 2: Summarizing the characteristics of app release notes along with the derived features. We calculate the percentage of releases of an app with features above the third quantile. We use the percentage of high value (i.e., above the third quantile) in comparison to the median and the average values as we observed that the percentage of high value provides an unbiased representation for each app. In particular, the median metric value does not take into account extreme values, and the average metric value could be skewed by extreme values.
Step 3: Identifying app releasing patterns. We use K-means clustering [11] on the summarized release notes features to find different release notes patterns from the clusters. We find the optimum number of clusters from both the Gap Statistic method [36] and the manual investigation. For the manual investigation, we start with two clusters and gradually increase the number of clusters until we observe that there are two or more clusters that describe highly similar release notes styles (i.e., the new sub-clusters describe both apps with long and high-updatability release notes content).
Step 4: Evaluating app releasing patterns. We use the silhouette validation technique to evaluate our clustering algorithm. The silhouette validation [33] measures the separation distance between each cluster. We compare the dissimilarity of release notes with other release notes of the same cluster to find the silhouette value $S(i)$, as defined in Equation 1.

$$S(i) = \frac{b(i) - a(i)}{max(b(i), a(i))} \tag{1}$$

[1] Apps in pattern 2 and pattern 5 do not have major releases during our studied period, so the centroid for major releases updatability in patterns 2 and 5 are NA

Where $a(i)$ is the *average* dissimilarity of the characteristics of the release notes of an app in a cluster to the characteristics of the release notes of the other apps in the same cluster and $b(i)$ is the *minimum* of the average dissimilarity of the release notes of an app to the release notes of the apps in the other clusters. The silhouette value has a range between -1 and 1, in which a value close to -1 implies poor clustering performance and a value close to 1 implies good clustering performance.
Results: We find six clusters as the optimum number of clusters. The average silhouette value for each release notes pattern is 0.58. A value over 0.5 (i.e, 0.58) suggests a reasonably good clustering of release notes [33]. Table 2 shows the summary of the six identified patterns. Using the clustering centroid for each of the summarized three features, we tag each pattern with the appropriate pattern name. For example, we name pattern 1 *"Short non-updating steady"* because apps that belong to pattern 1 have short and repetitive release notes content.

The first two authors perform a further manual investigation on a statistically representative random sample with a confidence level of 90% and a confidence interval of 10% from each of the identified six patterns to gain further insights on the content of the release notes of each pattern. The investigation involves reading through all release notes of an app. During our manual analysis, if there was a disagreement in the meaning of release notes, the authors carefully reread the release notes and user reviews of this release and further discussed until consensus was reached. Since both authors analyzed together all release notes and agreements (on all analysis of the examined release notes) were reached in the end, the authors did not compute the inter-rater agreement. In total, we manually investigate the release notes of 315 apps that belong to the identified six patterns. Tables 3 and 4 show the release notes of example apps for each identified pattern. The six patterns can be explained as follows:

Pattern 1 (Short Non-updating Steady Pattern):

Frequency: As shown in Table 2, pattern 1 is the *second most common* pattern in the studied apps (i.e., 439 apps).
Description: Apps in pattern 1 generally have short and rarely updated release notes. More specifically, minor changes are made to the content of release notes during major releases. After the first two authors read through a statistically representative random sample of 60 apps that belong to pattern 1, we observe that apps in this pattern rarely use major releases, and tend to write words "*enhancements*" and "*bug fixes*" in their release notes. We also observe that apps in pattern 1 generally reuse the same text in all their release notes. For example, the *"Old Navy"* app, listed in Table 3, uses identical release notes throughout the studied period.

Pattern 2 (Short Updating Steady Pattern):

Frequency: As shown in Table 2, pattern 2 is the *least common* pattern in the studied apps (111 apps).

Table 3: Examples of release notes from short release notes patterns (pattern 1-3).

Version String	Release Type	Release Notes
Pattern 1: Short non-updating steady - Example app: *"Old Navy"*		
2.3.4	minor	*"GUI enhancements and minor fixes"*
2.3.5	patch	*"GUI enhancements and minor fixes"*
2.3.6	patch	*"GUI enhancements and minor fixes"*
2.3.7	patch	*"GUI enhancements and minor fixes"*
2.3.8	patch	*"GUI enhancements and minor fixes"*
Pattern 2: Short updating steady - Example app: *"NPR News"*		
2.7.0	minor	*"Fix a bug of the grouping separator. Improved calculation precision"*
2.7.1	patch	*"Notifications will open the corresponding story correctly again. Thanks for your feedback."*
2.7.2	patch	*"This release properly respects opting out of news alerts. Thank you for your feedback which enabled us to quickly address this issue."*
2.7.3	patch	*"This release fixes a minor issue impacting some users subscribing to news alerts."*
2.7.4	patch	*"Fixed audio playback issue impacting 8.0+ Android devices. Thank you for your feedback!"*
Pattern 3: Short rising-updatability with major releases - Example app: *"UpToDate"*		
6.4.1	patch	*"Bug fixes"*
6.4.2	patch	*"Bug fixes"*
7.0.2	major	*"New design easy navigation and a stunning new display! Customize your topics and locations to get news just for you. Live video breaking news traffic and weather alerts in real-time."*
7.0.3	patch	*"We made a number of bug fixes and updates in response to user feedback and we're still working on others. Here are some of the updates in this release: Fixed bug that allowed device to go into sleep mode while viewing video. Fixed bug with persistent location services permission requests. Updated weather maps and web cams to support landscape orientation and sharing. Other bug fixes and enhancements."*

Description: Apps in pattern 2 use short release notes that succinctly describe changes. We observe that app developers in pattern 2 sometimes mention user reviews in their release notes and describe what they did to address user concerns. For example, as shown in Table 3, developers of the "*NPR News*" app mention in the patch release version *"2.7.2"* that the newly implemented features address the feedback from the user-base of the app: "*This release properly respects opting out of news alerts. Thank you for your feedback which enabled us to quickly address this issue*".

After the first two authors read through the release notes of a statistically representative random sample of 55 apps that belong to pattern 2, we observe that apps in this pattern frequently mention the overall changes in the new releases, instead of listing the details of the added/modified features. The updated release notes describe the overall enhancements of an app such as (1) performance and stability improvements, (2) UI enhancements, (3) providing support for the latest version of the Android platform, (4) offering app controlling features through the settings screens, (5) reducing advertisements, and (6) reducing the app size.

Pattern 3 (Short Rising-updatability with Major Releases Pattern):

Frequency: As shown in Table 2, pattern 3 contains 157 apps.
Description: Apps in this pattern frequently use "Bug fixes" keywords with minor details that changes with each release notes (e.g., *"Bug fix - language menu"*). For most major releases, developers add more detail. For example, developers of the *"UpToDate"* app (Table 3) write identical release notes until a major release, in which multiple new functionalities are added. After the major release (version *"7.0.2"*), subsequent patches are more varied as well.

Pattern 4 (Long Non-updating Steady Pattern):

Frequency: As shown in Table 2, pattern 4 is the *most common* pattern in the studied apps (572 apps).
Description: We find that apps in pattern 4 have release notes with long content, but the release notes are rarely updated. As observed from the manual analysis, the release notes in pattern 4 mostly contain two types of information:

(1) Promotional information describes new offers or products in an app, such as *"With Staples Weekly Ad sneak peak in this update we've made Staples more convenient than ever !"*.

(2) *Instructional information* describes how to use the new features, such as *"In Android Nougat 7.1 long-press our app icon for App Shortcuts"* messages to app users.

As show in Table 4, the releases of the *"NYC Subway Map"* app contain long release notes. However, the content of the release notes is not updated.

Pattern 5 (Long Updating Steady Pattern):

Frequency: As shown in Table 2, pattern 5 is the *least common* pattern in the group of the studied apps with *long release notes* (145 out of 1,005 apps).
Description: Apps in pattern 5 have detailed and descriptive release notes that are frequently updated. In particular, release notes frequently describe the added or updated functionalities in detail. For example, developers of the *"Yard Sale Treasure Map"* app (Table 4) use release notes to describe all added functionalities. We also observe that the developers in the *"Yard Sale Treasure Map"* app leverage the release notes to communicate *technical* modifications and problems within the app (e.g., "*!!! Emergency fix for server problem!!!*" in version *"7.0.2"*)

We find that the words "*crash*" and "*emergency*" are often included in release notes. We also observe that app developers in pattern 5 change their release notes due to negative reviews and app crashes. For example, the *"My*

Table 4: Examples of release notes from long release notes patterns (pattern 4-6).

Version String	Release Type	Release Note
Pattern 4: long non-updating steady - Example app: *"NYC Subway Map"*		
4.0.1	major	*"All new! The sharpest map out there save your favorites tap on stations interactively get directions..."*
4.0.3	patch	*"All new! The sharpest map out there save your favorites tap on stations interactively get directions..."*
4.0.4	patch	*"All new! The sharpest map out there save your favorites tap on stations interactively get directions..."*
Pattern 5: long updating steady - Example app: *"Yard Sale Treasure Map"*		
6.3.1	patch	*"Improving map efficiency (busy yard sale areas will see smoother map updates) - Bug fixes with Change Location feature - New app icon 6.3.0 ..."*
6.4.0	minor	*"Minor release fixing bugs that could cause crashes. What's New - New feature to viewed sales - New feature to check off visited sales in your route - New option to use Waze for navigation to sales What's Different ..."*
6.4.1	patch	*"!!! Emergency fix for server problem !!! Adding tutorial showcases for some features - Fixing route line problem to draw lines from current location instead of last searched location ..."*
Pattern 6: long rising-updatability with major releases - Example app: *"Shopular"*		
6.8.0	minor	*"We've made the highest-rated coupons - cash back app even better by adding more stores and deals. Happy shopping!"*
6.9.2	minor	*"We've made the highest-rated coupons - cash back app even better by adding more stores and deals. Happy shopping!"*
7.0.1	major	*"Black Friday 2017 is right around the corner! Our team works around the clock to bring you ad scans and the best Black Friday deals as soon as they are released."*
7.0.4	minor	*"We've made the highest-rated coupons - cash back app even better by adding more stores and deals. Happy shopping!"*

Pregnancy and Baby Today" app introduced a new functionality in version *"3.4.0"* after receiving negative reviews ("*Message received! You told us you wanted the ability to share articles and videos with your family and friends. So thank you for all the feedback and we're happy to introduce Sharing*").

Pattern 6 (Long Rising-updatability with Major Releases Pattern):

Frequency: As shown in Table 2, pattern 6 contains 288 apps.

Description: Apps in pattern 6 have repetitive release notes during minor and patch releases that are used for instructional or promotional purposes for users. Release notes are updated as the major releases and the updated release notes are generally written for large functionality updates. For example, the *"Shopular"* app (Table 4) contains mostly repetitive release notes except for

Table 5: The user-developer dialogue and the release notes of the *"Staples"* app.

Type	**Time**	**Content**
User review	December 24st 2016	*"Your app has fallen back into a nearly terrible category. Rewards section rarely works can never fetch order statuses and browsing in the app is poor. Best Buy's app is leaps and bounds better"*
Release Notes	December 31st 2016	*"We made some tweaks for this release that address the following issues: Redesigned account page so you can check your rewards and order history more easily..."*
Developer Response	December 31st 2016	*"Hi we have made some recent updates to the app. Thank you for reaching out. If you update your app version the performance should be improved"*

the major release *"7.0.1"*, in which the release notes are updated due to a holiday event (e.g., Black Friday).

We also observe that developers in pattern 6 use the release notes as a way to broadcast a response to many user reviews. For example, we observe from Table 5 that the *"Staples"* app frequently addresses specific user concerns. We find a two-star user review on December 24st 2016 with regards to the rewards section. A week after the user review (December 31st 2016), the *"Staples"* app released version 5.4.1 and responded to the user. Moreover, the developers on the *"Staples"* app addressed the negative user reviews in their release notes, and incorporated both the review and the release notes message in the response message.

Summary of the identified patterns

Through the manual investigation of the six identified patterns, we find that long release notes do not necessarily provide information about the latest updates in a new release. For example, the provided instructions or advertisement for an app are often repetitive and do not indicate developer activities. However, updating release notes generally leads to descriptive and leveraged release notes. We observe that developers who update release notes frequently write release notes in *technical* wording to describe functionality modifications in detail. Moreover, developers who rarely update release notes frequently insert promotional information in the release notes.

Of the six patterns, we observe that the main reason that some apps are grouped into short and non-updating patterns is that the release notes are written only for the customers to understand the app as a whole. For example, short release notes may advertise the best functionalities, instruct functionalities for customers, or thank customers for using the app. However, apps in the long and updating patterns leverage release notes to inform customers on a specific release. Long release notes discuss new functionalities in detail and explain why/how they were added. Furthermore, apps in pattern 6 (i.e., *long rising-updatability with major releases*) tend to use release notes to address user concerns in the most updated release notes (i.e., release notes of major releases).

Summary

We identify six patterns of release notes with different characteristics. App developers dominantly apply the short non-updating steady pattern (26% of apps) and the long non-updating steady pattern (33% of apps) to write release notes. Non-updating release notes (apps in patterns 1 and 4) tend to repeat words, such as"*bug*" and "*improvements*", while updating release notes tend to explicitly mention the addressed issues and demonstrate the emergency nature of the new release.

RQ2: What are the characteristics of the apps that follow a certain release notes pattern?

Motivation: In the previous RQ, we identify six patterns based on the characteristics, such as richness and updatability, of the release notes. In this RQ, we are interested in discovering the common features across the apps that belong to the same release notes pattern. For example, we want to see if a descriptive and non-updating pattern mainly occurs in certain app categories or certain app download ranges. Furthermore, understanding the correlation between release note patterns and the releasing practices of an app can give advice to developers on how to prepare their release notes. In particular, finding the correlation between release note patterns and the perceived quality of an app (e.g., average release rating) can provide guidance for developers to follow some of the six patterns.

Approach: To further understand release notes patterns, we build six different models to identify the correlation between app-related features and each of the six identified patterns. Our approach consists of three steps: (1) collecting the app-related features, (2) constructing the models, and (3) analyzing the constructed models.

Step 1: Collecting the app-related features. We use app specific features as independent variables in predicting whether an app belongs to pattern i, where $i \in \{1, 2, .., 6\}$. As listed in Table 6, we measure 11 app-related features that can potentially correlate to release notes styles. We choose these features based on prior research regarding the Google Play Store apps and release notes [12, 14, 21, 29, 30].

For the keyword features (i.e., *bug*-related releases, *improvement*-related releases, and *emergency*-related releases), we use *word2vec* [8] on all release notes words to expand our initial keyword sets (i.e., *"bug"*, *"issue"*, *"enhance"*, *"fix"*, *"improve"*, *"emergency"*, and *"urgent"*). For example, the word *"bug"* appears frequently with the word *"fix"* (e.g., *"minor bug fixes"*) on all release notes, so word2vec groups the words "*bug*" and "*fix*" together as related

Table 6: The collected features for each app.

Feature	**Values**	**Explanation**
App metadata features		
App category	Categorical	Category of the studied app (i.e., either *Game* or *non-Game* app).
App size	Numerical	Size of the APK file in MB.
Size of the user-base	Numerical	The number of downloads.
Release practice features		
Release count	Numerical	The total number of releases for every app during the studied period.
Release frequency	Numerical	The median time-to-release for every app.
Bug keywords	Numerical	The percentage of releases with *bug-related* keywords in the release notes text.
Improvement keywords	Numerical	The percentage of releases with *improvement-related* keywords in the release notes text.
Emergency keywords	Numerical	The percentage of releases with *emergency-related* keywords in the release notes text.
Same-day releases	Numerical	The percentage of releases that have time-to-release of a single day.
Response rate	Numerical	The ratio of developer responses to the posted reviews of an app.
Perceived quality features		
Average release rating	Numerical	The average star ratings of the deployed releases of an app.
Percentage of low ratings	Numerical	The percentage of one and two star ratings for all the deployed releases of an app.
Rating increase	Categorical	The difference between the app ratings at the end of the studied period and the app ratings at the beginning of the studied period (converted to Boolean value to reflect the increase/decrease in the ratings of an app).

words. When searching for *bug*-related releases, any release notes that include the words "*fix*" or"*bug*" are flagged as a *bug*-related release. The Appendix describes the list of keywords that are used to identify bug-related releases, improvement-related releases, and emergency-related releases.

Finally, we remove the highly correlated features. We measure the correlation between the collected features using the Spearman rank correlation test. We use a cut-off value for ρ of 0.7 [29]. After the correlation process, the "*percentage of low ratings*" feature and the "*average release rating*" feature are highly correlated ($\rho = 0.78$). The "*release count*" feature and the "*release frequency*" feature are highly correlated ($\rho = 0.71$). Hence, the "*percentage of low ratings*" feature and the "*release frequency*" feature are removed from our features.

Step 2: Constructing the models. We construct six logistic regression models using the **glm** package [20]. For model i, we tag all apps with pattern P_i as 1 and apps with other patterns as 0. For each of the six models, the presence of pattern P_i is the dependent variable, or *response class.* Since our dependent variable can only take on the values 0 and 1, we treat the dependent variable as a binary variable. We use logistic regression models, as logistic regression is frequently used for models involving *binary dependent variables* [6, 10, 17, 27].

Step 3: Analyzing the constructed models. After building the logistic regression models, we measure the discriminatory power of the constructed models using the area under the receiver operator characteristic curve (AUC). AUC is the area under the plot of the true positive rate against the false-positive rate. An AUC value close to 1 means that our independent variables have high discriminatory power. More specifically, our logistic regression model M_i can better determine whether an app is clustered to pattern P_i [15].

To understand the most important features in our logistic regression models and identify the features that are most contributing to each release notes pattern, we use Wald statistics (χ^2) to estimate the relative contribution of each app feature in the constructed model. The larger the χ^2 value, the larger the impact that a particular feature has on the performance of the model (i.e., the AUC value) [24,37]. We then calculate the *percentage* of χ^2 for each feature to the total χ^2 for all app features [12]. We also analyze the *coefficient sign* for each app feature in the generated models. A positive coefficient sign of an app feature indicates that the app feature *positively* correlates to our independent variable (e.g., if the *app size* has a positive coefficient sign in model 1, then an app with a large app size is more likely to belong to pattern 1). Similarly, a negative coefficient sign for an app feature signifies that the app feature negatively correlates to our independent variable.

Results: The constructed models for the apps with short updating release notes have the highest explanatory power. Table 7 shows a summary of our constructed models. The highest AUC values from the logistic regression models for pattern 3 (*short rising-updatability with major releases*) and pattern 2 (*short updating steady*) are 0.82 and 0.79 respectively. Each model has a different feature with the highest explanatory power. We describe our analysis of the constructed models in the following discussion.

Model 1 (short non-updating steady). Apps in pattern 1 frequently mention *bug* keywords (e.g., *"Bug fixes"*) in the release notes. However, apps in pattern 1 infrequently use *emergency* keywords (e.g., *"Emergency update due to crash"*) or *improve* keywords (e.g., *"UI improvements"*) in the release notes. Developers of the apps in pattern 1 infrequently respond to user reviews. Apps in pattern 1 negatively correlate to average release ratings, meaning that apps

Table 7: A summary of the analysis of the constructed six models.

Discriminatory power of the constructed models

Model statistics	**Model 1** [2]	**Model 2**	**Model 3**	**Model 4**	**Model 5**	**Model 6**
AUC	0.67	0.79	0.82	0.75	0.77	0.69

Impact of all app features on the six models

App features	**Percentage of χ^2 (Coeff. Sign)**					
	Model 1	**Model 2**	**Model 3**	**Model 4**	**Model 5**	**Model 6**
App category	0.02 (+)	0.05 (-)	0.01 (+)	0.03 (+)	0.01 (+)	0.10 (-)
App size	0.13 (-)	0.14 (-)	0.12 (+)	0.03 (+)	0.02 (+)	**0.26 (+)**
Size of the user-base	0 (-)	0.03 (-)	**0.30 (-)**	0 (+)	0.02 (+)	0.01 (+)
Release count	0.00 (-)	**0.22 (-)**	0.06 (-)	**0.44 (+)**	**0.37 (-)**	0.00 (-)
Bug keywords	**0.27 (+)**[3]	0.10 (+)	0.10 (+)	0.05 (-)	0.05 (-)	0.01 (+)
Improve keywords	0.17 (-)	**0.22 (+)**	0.01 (+)	0.13 (+)	0.01 (-)	0.01 (+)
Emergency keywords	**0.21 (-)**	0.07 (+)	0.05 (-)	0.04 (+)	0.01 (+)	0.08 (+)
Same-day releases	0.00 (+)	0.10 (+)	0.03 (+)	**0.19 (+)**	0.02 (-)	0.08 (-)
Response frequency	0.16 (-)	0.03 (-)	0.04 (+)	0.01 (+)	**0.20 (+)**	**0.24 (+)**
Average release rating	0.03 (-)	0.01 (-)	0.08 (-)	0.17 (+)	0.11 (+)	0.04 (+)
Rating increase	0.01 (-)	0.03 (+)	**0.20 (-)**	0.01 (+)	0.18 (+)	0.17 (+)

that use short and non-updating release notes tend to have lower average release ratings than other apps.

In summary, the developers of apps in pattern 1 do not respond to user reviews and do not clearly mention the addressed issues in the release notes.

Model 2 (short updating steady). Developers of apps in pattern 2 frequently use *improve* keywords (e.g., "improvement" and "enhancement") as developers of these apps describe briefly the newly added features (e.g., *"Improved the stability of playing video"* and *"Performance enhancements"*). Apps in pattern 2 generally have a lower release count than the other apps. Similar to pattern 1, apps in pattern 2 generally have a lower average release rating than other apps.

Model 3 (short rising-updatability with major releases). Apps in pattern 3 have a low number of downloads and a decrease in release ratings during the period of the study. The result shows that apps in pattern 3 are negatively

[2] Models 1-6 are the models for the patterns 1-6 as follows: (1) *short non-updating steady*, (2) *short updating steady*, (3) *short rising-updatability with major releases*, (4) *long non-updating steady*, (5) *long updating steady*, and (6) *long rising-updatability with major releases.*

[3] The bold text highlights the app features with the highest impact on the response variable.

correlated to the rating increase. Based on our manual investigation, we observe that the frequently updated major release notes mainly describe the resolved issues in the release. It may indicate the reason for the low average release rating of the apps in pattern 3 as such apps have frequent issues that need to be fixed.

Model 4 (long non-updating steady). Apps in pattern 4 generally have rapid release cycles and frequently deploy same-day releases, unlike apps in patterns 1, 2, and 3. For such apps, developers write short release notes. Apps in pattern 4 generally have a high average release rating and an increase in release ratings over the three-year studied period.

Model 5 (long updating steady). Apps in pattern 5 infrequently deploy but frequently broadcast the new releases to users. Similar to apps in pattern 4, apps in pattern 5 have high release ratings overall and an increase in release rating during our studied period.

Model 6 (long rising-updatability with major releases). Apps in pattern 6 generally have a larger APK size and a higher response rate than the other studied apps. Apps in pattern 6 are less likely to be in the Game category than apps in the other identified patterns.

Summary of the six patterns

The constructed six models show that apps in patterns 4, 5, and 6 with longer release notes tend to have a higher average release rating than the other studied apps. Apps in pattern 3 that have short release notes and major varied releases have a decrease in user ratings during our studied period.

The apps in pattern 5 with long and updating release notes have both high average release rating and a rating increase during our studied period. Hence, store owners can notify users with apps in pattern 5 as such apps have high perceived quality releases and high-quality release notes that contain more comprehensive content.

Furthermore, apps in pattern 6 tend to have the highest response rate. As responding to use reviews results in a positive change in review ratings [12], we suggest that developers with apps in pattern 6 can leverage release notes content to automatically respond to user reviews by notifying users who are impacted by the changes in the new release.

Summary

Apps in patterns 2 and 5 with updating release notes tend to have a lower number of releases. Apps in pattern 6 that have long, rising-updatability release notes with major releases tend to have a higher user response rate than other apps. Apps in pattern 5 that have long and updating release notes have higher perceived quality releases compared to other apps.

RQ3: What causes developers to shift their release notes pattern?

Motivation: During the evolution of mobile apps, we observe that the developers might change their practices in writing their release notes. We are interested in finding whether the release note shifts are driven by the attempt to improve star ratings and if the shifts have any positive effects on the perceived app quality. In this RQ, we identify pattern shifts that are associated with an increase in the app ratings. Identifying such shifts can help developers understand how to adapt their release notes patterns to improve their app ratings.
Approach: For all the studied apps, we treat each of the clustering metrics (e.g., content richness and updatability) as a time series. In particular, we treat every new release as a change in time. We then use the mean and variance value of each time series to determine whether the time series of the clustering metrics is *stationary.* For a stationary time series, the mean value of time series is constant over time implying that the trend component is nullified, and the variance does not increase over time implying that the seasonality effect is minimal [5]. We use a *stationary p-value* threshold of 0.05, where a time series above the threshold is *non-stationary* [34]. When any of the clustering metrics change significantly for the time series of an app, we define the changing release as a *metric shift* and identify the time series as *non-stationary.*

To analyze the impact of the shift in release notes patterns on app ratings, we measure the average rating of each non-stationary app (i.e., pattern shifting app) *after* the shift and compare it to the average rating of the app *before* the shift. Finally, the first two authors of this paper manually analyze all non-stationary apps to gain further insights on what developers mention in the release notes after changing their release patterns. During our manual analysis, if there was a disagreement in the meaning of release notes, the authors carefully reread the release notes and user reviews of this release and further discussed until consensus was reached. Since both authors analyzed together all release notes and agreements (on all analysis of the examined release notes) were reached in the end, the authors did not compute the inter-rater agreement.

Results: We find 69 apps that are non-stationary on at least one of the clustering metrics. After manually examining each of the 69 non-stationary apps, we observe that 53 consistently shift patterns. The 16 excluded apps either have an insignificant metric shift, or have a metric shift that continues for two to three releases, and then return to the previous metric patterns. To determine the reasons behind each shift, we then read the release notes of the 53 non-stationary apps and identify their release notes style before and after the shift. Table 8 shows the summary of all identified pattern shifts.

92% of the studied apps shift either from short to long release notes or from rarely updated to frequently updated release notes. Of the 53 apps that display a shift in release patterns, 49 (92%) shift from short to long (31 apps) and from rarely updated to frequently updated (18 apps). The results show that when apps shift patterns, developers tend to add more

Table 8: A summary of pattern shifts.

Event type	Event	Occurrences
Short to long	Pattern 1 to pattern 5	18
	Pattern 1 to pattern 4	11
	Pattern 3 to pattern 5	2
Rarely updated to frequently updated	Pattern 4 to pattern 5	9
	Pattern 1 to pattern 2	6
	Pattern 4 to pattern 2	3
Others	Pattern 5 to pattern 1	3
	Pattern 6 to pattern 4	1

Table 9: The identified reasons for pattern shifts in the studied release notes.

Number	Event	Occurrences
1	**Listing functional updates to providing user guidance** (e.g., "*To access the newly updated profile settings, please go to ...*")	18
2	**Negative user feedback** (e.g., "*Thank you for your feedback which enabled us to address the issue of: ...*")	10
3	**Crash fixes** (e.g., "*Crash emergency update!*")	7
4	**UI-related changes** (e.g., "*Changed color for the app border*")	6
5	**Standard/professional release notes to joke release notes** (e.g., "*We squashed some cute bugs!*" or "*Swat swat, go away pesky bugs!*")	5
6	**Changes in the Google Play Store service libraries** (e.g., "*Updated Google Play Store service libraries due to ...*")	4
7	**Holiday events** (e.g., "*Christmas Update: ...*")	4

details to non-updating release notes. We observe infrequent shifts involving patterns 3 (*short rising-updatability with major releases*) and 6 (*long rising-updatability with major releases*). It indicates that developers implementing apps in *rising-updatability* patterns do not frequently shift their release notes behavior.

Developers shift their release notes patterns to provide detailed guidance to their users and spot the importance of the new release. Through manual analysis of the apps with shifts of release note patterns, we identify seven main reasons for developers to change their release notes patterns. As outlined in Table 9, 33% of the identified pattern shifts are because of a shift from functional updates to provide detailed user guides on how to use the new features. An example of a functional update is *"Sign in with Fingerprint - Added PDF support for invoices - Bug Fixes- UI/UX improvements"*.

Table 10: Examples of release notes shifts.

Pattern shift	**Version string**	**Release type**	**Release notes**
Example release notes of the *“Airbnb”* app. The app ratings increased from 4.30 to 4.33 after the shift.			
P1 to P5 (short non-updating to long updating)	16.44.1	minor	*“Bug fixes and performance improvements.”*
	16.45.2	minor	*“Bug fixes and performance improvements.”*
	16.46.0	minor	*“Bug fixes and performance improvements.”*
	16.47.3	minor	*“Introducing the new Airbnb. Now you’ve got homes experiences and places all in one app...”*
	16.48.1	minor	*“This holiday give the gift of travel. Now you can send Airbnb gift cards right from our app. Introducing the new Airbnb ...”*
Example release notes of the *“AVG Anti-Virus”* app. The app ratings increased from 4.48 to 4.52 after the shift.			
P4 to P5 (long non-updating to long updating)	16.44.1	minor	*“Enhancements and bug fixes on ...”*
	16.45.0	minor	*“Enhancements and bug fixes ...”*
	16.46.3	minor	*“Delivering our new GDPR-focused data privacy and security benefits to all of our users worldwide ...”*
	16.47.1	minor	*“Improved efficiency of Web Shield. Faster checking of URLs and fixed the occasional error...”*
Example release notes of the *“NYC Subway Map”* app. The app ratings decreased from 4.58 to 4.35 after the shift.			
P5 to P1 (long updating to short non-updating)	3.1.0	minor	*“You can now view the scheduled train arrival times! Stability improvements ...”*
	3.1.1	patch	*“You can zoom in farther! You can now view the scheduled train arrival times! ...”*
	3.1.3	patch	*“Remove unnecessary permissions”*
	4.0.1	major	*“All new! The sharpest map out there ...”*
	4.0.3	patch	*“All new! The sharpest map out there ...”*

Moreover, developers change their release notes patterns due to negative user feedback (19%) and crashes (13%). The obtained results indicate that developers use the release notes to spot the importance of the new release to the user-base (e.g., “*Crash emergency update!*”).

Apps receive higher ratings after shifting from rarely updated to frequently updated release notes. We study the changes in average ratings for the most frequent pattern shifts. Of the apps that shift from pattern 1 to pattern 5, 94% show a higher average rating after the shift. 78% of the apps that shift from pattern 4 to pattern 5 show a higher average rating after the shift. All three apps that shift from pattern 5 to pattern 1, namely the *“Fifth*

Third Mobile Banking" app, the *"News 12"* app, and the *"NYC Subway Map"* app, show a *decrease* in the average release rating. The obtained results show that an increase in user ratings correlates to a shift from either short to long release notes or from rarely updated to frequently updated release notes.

Table 10 shows an example of release notes from three apps to demonstrate a release notes pattern shift. As listed in Table 10, apps shifting from short to long release notes and from rarely updated to frequently updated release notes receive higher release ratings. Examples of this shift are the *"Airbnb"* app and the *"AVG Anti-Virus"* app. Furthermore, apps shifting from long to short release notes and from frequently updated to rarely updated release notes tend to receive lower ratings.

Summary

Pattern shifts mainly occur when developers shift from short to long release notes, and rarely updated to frequently updated release notes. We identify seven reasons for the shifts, such as a shift from listing functional updates to providing user guidance, addressing negative user reviews, and resolving crash issues. We observe that an increase in user ratings correlates to a shift from either short to long release notes or from rarely updated to frequently updated release notes.

4 Implications

In this section, we discuss the implication of our results for app developers, store owners, and researchers.

4.1 Implications for app developers

Developers can leverage the release notes to broadcast the responses to app users. In RQ2 and RQ3, we observe that app developers of patterns 5 and 6 respond to the posted user reviews. App developers of pattern 6 use release notes to broadcast responses to the user-base of their apps. For example, as shown in Table 5, developers of the *"Staples"* app addressed the negative user review in their release notes. In addition, developers incorporated both the user reviews and the release notes in the response messages.

4.2 Implication for store owners

Store owners can recommend users the apps that have descriptive release notes, high perceived quality in their deployed releases, and frequently respond to user reviews. As discussed in RQ2, apps in pattern 5 generally perceive a higher average release rating than other apps. Apps in patterns 5 also show an increase in rating during our studied period and frequently respond to user reviews. Prior work finds that a high developer response rate to user concerns correlates to higher review ratings [12]. Hence, store owners can recommend users the apps that have a higher user response rate than other apps and deploy high perceived quality releases with descriptive release notes.

4.3 **Implications for researchers**

Release notes are not always an insightful indicator of the change log of a new release. As discussed in RQ2, the most common patterns for our studied apps are the non-updating patterns. Most of the studied apps do not regularly update their release notes. Hence, researchers should include other resources in tracking the changes in a new release. For example, researchers can investigate the code commits for every release. As not all apps are open source apps, it opens a challenge for researchers to provide tools and approaches that can analyze the deployed releases (i.e., APKs) without depending on the release notes.

5 Threats to validity

5.1 Construct validity

Construct threats to validity are concerned with the degree to which our analyses measure what we claim to analyze. Throughout our study, we filter out the first release note of each app as we cannot measure their cosine similarity to the previous release notes and time-to-release. The excluded release notes could contain useful information that we discard. To mitigate this effect, we try to study a long period for each app, which is from April 2016 to April 2019. Our study includes a median of 30.8 releases per app.

In RQ3, we use Stationary analysis on time-series data. The automatic identification of the pattern shifts using the Stationary time-series analysis can lead to false-positive cases. Hence, we performed a manual analysis for the identified 69 apps and read their to understand why their shift occurred.

5.2 Internal validity

Internal threats to validity are concerned with the relationships between our dependent variables and independent variables. The dependent variables are if an app is clustered into pattern P_i, for $i \in \{1, 2, ..., 6\}$, and our independent variables are the app features. We select app features that cover a wide range of characteristics and would likely be linked to the writing styles of release notes. However, our selection may not be exhaustive. For example, we were unable to extract data from individual developers of each app, such as cultural background and coding experience. Developer information could have a high impact on the style of release notes they write. Hence, further studies can analyze the other factors that may be correlated to the characteristics of release notes by conducting user surveys on app developers.

5.3 External validity

External threats to validity are concerned with our ability to generalize our results. Our study focuses on analyzing the 2,232 top apps in the Google Play Store over a period of three years. To eliminate the bias in our results, we select top apps across all the app categories in the Google Play Store. Further studies should investigate more apps in a more extended period to understand how our findings apply to other types of apps, such as iOS apps and non-free apps.

6 Related work

In this section, we discuss the related work concerning the empirical studies on release notes, automatic generation of release notes, and emergency releases.

6.1 Empirical study of release notes

Abebe et al. [2] manually analyzed 85 releases notes across 15 software systems. Abebe et al. identified six types of information (e.g., addressed issues) in release notes, and observe that most release notes list only a subset of addressed issues. Abebe et al. built machine learning models to automatically suggest the issues that need to be listed in the release notes. Abebe et al.'s approach achieves an average precision of 84% and an average recall of 90%.

Martin et al. [21] performed a causal analysis between the release notes and the app ratings. Martin et al. found that the content of the release notes has an impact on user ratings.

Noei et al. [30] studied 4,193,549 user-reviews of 623 apps in the Google Play store. Noei et al. identified the key topics in every app category. For example, Noei et al. identified the most frequently mentioned topics in the reviews of every app category, and then measured the similarity between the release notes and the identified key topics. Noei et al. found that the release notes of the highly rated releases have significant correlation with the key topics of the category of this app. McIlroy et al. [22] studied the frequency of releases of 10,713 mobile apps in the Google Play Store for two months. McIlroy et al. found that 45% of the frequently-updated apps do not provide any information on the rationale of new releases.

Our paper empirically studies how developers leverage release notes in terms of how descriptive and updating release notes are. We group different release notes styles into six patterns and identify the patterns that tend to have higher ratings. Our work can help developers leverage the release notes to improve the perceived quality of their apps.

6.2 Automatic generation of release notes

Moreno et al. [25] manually analyzed 1,000 release notes from 58 software projects and classified the content of release notes into 17 types, such as fixed bugs, new features, or changes to documentation. Moreno et al. then introduced an approach that can automatically generate release notes. Based on their previous work on the generation of release notes [25], Moreno et al. [26] performed more qualitative studies on the perceived quality of their release notes generation technique and find an accuracy of 86% for their approach.

Yuan et al. [39] performed a case study on the Android Open Source projects to predict and detect features that need to be included in release notes. Yuan et al. used word2vec to represent commit messages and release notes as lower-dimensional vectors. Yuan et al. used K Nearest Neighbors to cluster these vectors into groups, each of which represents a feature (e.g., network connection, build system). The work then proposes to include the features with big change difference compared to the previous version in the release notes, and achieves 13.83% to 17.69% improvements compared to the state-of-the-art prediction techniques.

Khalfallah et al. [16] proposed an approach to generate release notes for satellite projects. Khalfallah et al. first summarized the patterns in the release notes written by architects, and then used the identified patterns to generate release notes based on the source code changes in the newly released versions. Klepper et al. [18] proposed a semi-automated approach for collecting information from the issue tracker, build server, and version control systems and generating release notes based on the collected information. The information contained in the release notes can be customized for a different audience, such as user, customer, and team member.

Our study shows that app developers in some cases write long and descriptive release notes in patterns 5 and 6. Therefore, developers of such apps can leverage the existing release notes to write long and descriptive responses to users.

6.3 Emergency releases

Wang et al. [38] performed an empirical study on app changelogs from the developers' perspective. Wang et al. analyzed changelogs in terms of functional and non-functional requirements, and found that the majority of the app changes refer to non-functional requirements. Gao et al. [7] proposed an approach to identify emerging issues from app reviews. Gao et al. found that the release notes from six popular apps on the Google Play Store and Apple App Store address the identified emerging topics.

Hassan et al. [14] analyzed the emergency releases of the top 10,000 mobile apps in the Google Play Store. Hassan et al. found that 70% of the release notes of the emergency releases are repetitive and generic releases notes. Hassan et al. [12] analyzed the release notes in 250 bad releases of 2,526 top free-to-download apps in the Google Play Store. Hassan et al. found that 63.4% of the cases developers fix the reported issues without updating the release notes or mentioning details about the fixed issues.

Our work differs from prior studies as we are the first work that performs an in-depth analysis of release notes. In particular, we analyze 58,069 releases and over 67 million reviews of 1,712 apps. Our work shows that app developers write release notes to show the emergency nature of the deployed releases. For example, we find that apps in pattern 6 frequently mention the "*emergency*" keyword when they describe the newly emerging features in their releases. In addition, we observe a high correlation (44% χ^2) between the *release count* feature and apps belonging to pattern 4. The obtained results show that apps with long and non-updating release notes tend to release more frequently than apps in other release note patterns.

7 Conclusions

App release notes are important artifacts for both app users and app developers. In particular, developers use release notes to provide details on the updated functionalities of the newly deployed releases. However, little is known about the release notes practice in mobile apps and how app developers leverage the release notes of their apps. In this paper, we perform an in-depth analysis of release notes practice in the Google Play Store. In particular, we study 69,851 releases and 67.7 million user reviews for 2,232 top free popular apps in the Google Play Store. The most important findings of our study are:

1. App developers have different release notes practice concerning the richness and updatability of the release notes content. However, the majority of app developers tend to reuse their prior release notes when they update the release notes of their apps.
2. We identify six patterns of release notes. App developers dominantly use rarely updated release notes practice to write release notes. We observe that rarely updated release notes of apps in patterns 1 and 4 tend to repeat words *bug* and *improvements* while frequently updated release notes tend to address user reviews and use the keyword emergency.
3. We observe that apps in pattern 5 have high perceived quality releases, an increase in user ratings, and informative release notes. We also find that apps in pattern 6 have high perceived quality releases, informative release notes, and high response rates in their major releases.
4. Pattern shifts mainly occur when developers shift from short to long, and rarely updated to frequently updated release notes. We observe that an increase in user ratings correlate to a shift from short to long and rarely updated to frequently updated release notes.

Our work is the first step towards an in-depth analysis of the release notes practice in the Google Play Store. Our approach can help store owners notify users with apps that have descriptive release notes and high perceived quality in their deployed releases, such as apps in patterns 5 and 6. In addition, our work shows potential directions for developers to improve the release notes mechanisms in app stores. Future work can perform a qualitative user study on app developers to gain deeper insights on why they choose a particular release notes pattern.

References

1. App annie - the app analytics and app data industry standard. https://www.appannie.com/. Accessed: 2019-08-30.
2. S. L. Abebe, N. Ali, and A. E. Hassan. An empirical study of software release notes. *Empirical Software Engineering*, 21(3):1107–1142, 2016.
3. Akdeniz. Google Play Crawler. https://github.com/Akdeniz/google-play-crawler, Feb 2014. Accessed: 2019-09-30.
4. Appcues. 5 excellent product release note examples and how to write your own. https://www.appcues.com/blog/release-notes-examples. Accessed: 2020-01-23.
5. P. J. Brockwell, R. A. Davis, and M. V. Calder. *Introduction to time series and forecasting*, volume 2. Springer, 2002.
6. A. E. Camargo Cruz and K. Ochimizu. Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 460–463. IEEE Computer Society, 2009.
7. C. Gao, J. Zeng, M. R. Lyu, and I. King. Online app review analysis for identifying emerging issues. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 48–58. IEEE, 2018.
8. Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.
9. Google. Prepare & roll out releases - play console help. https://support.google.com/googleplay/android-developer/answer/7159011?hl=en. Accessed: 2020-01-23.
10. T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, 31(10):897–910, 2005.
11. J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.
12. Hassan, C. Tantithamthavorn, C.-P. Bezemer, and A. E. Hassan. Studying the dialogue between users and developers of free apps in the Google Play Store. *Empirical Software Engineering*, 23(3):1275–1312, 2018.
13. S. Hassan, C.-P. Bezemer, and A. E. Hassan. Studying bad updates of top free-to-download apps in the google play store. *IEEE Transactions on Software Engineering*, 2018.
14. S. Hassan, W. Shang, and A. E. Hassan. An empirical study of emergency updates for top Android mobile apps. *Empirical Software Engineering*, 22(1):505–546, 2017.
15. J. Huang and C. X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.
16. M. Khalfallah. Generation and visualization of release notes for systems engineering software. In *Proceedings of the International Conference on Complex Systems Design & Management*, pages 133–144. Springer, 2018.
17. T. M. Khoshgoftaar and E. B. Allen. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(04):303–317, 1999.

18. S. Klepper, S. Krusche, and B. Bruegge. Semi-automatic generation of audience-specific release notes. In *Proceedings of the 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*, pages 19–22. IEEE, 2016.
19. C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.
20. I. Marschner, M. W. Donoghoe, and M. M. W. Donoghoe. Package 'glm2'. *Journal, Vol*, 3(2):12–15, 2018.
21. W. Martin, F. Sarro, and M. Harman. Causal impact analysis for app releases in Google Play. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 435–446. ACM, 2016.
22. S. McIlroy, N. Ali, and A. E. Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, 21(3):1346–1370, 2016.
23. X.-L. Meng, R. Rosenthal, and D. B. Rubin. Comparing correlated correlation coefficients. *Psychological bulletin*, 111(1):172, 1992.
24. D. S. Moore. Generalized inverses, wald's method, and the construction of chi-squared tests of fit. *Journal of the American Statistical Association*, 72(357):131–137, 1977.
25. L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 484–495. ACM, 2014.
26. L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Arena: an approach for the automated generation of release notes. *IEEE Transactions on Software Engineering*, 43(2):106–127, 2016.
27. N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality. In *Proceedings of the 2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 521–530. IEEE, 2008.
28. E. Noei, D. A. Da Costa, and Y. Zou. Winning the app production rally. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 283–294. ACM, 2018.
29. E. Noei, M. D. Syer, Y. Zou, A. E. Hassan, and I. Keivanloo. A study of the relation of mobile device attributes with the user-perceived quality of Android apps. *Empirical Software Engineering*, 22(6):3088–3116, 2017.
30. E. Noei, F. Zhang, and Y. Zou. Too many user-reviews, what should app developers look at first? *IEEE Transactions on Software Engineering*, 2019.
31. J. Plisson, N. Lavrac, D. Mladenic, et al. A rule based approach to word lemmatization. *Proceedings of IS-2004*, pages 83–86, 2004.
32. T. Preston-Werner. Semantic versioning 2.0.0. `https://semver.org/`. Accessed: 2019-08-30.
33. P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
34. F. Sowell. Maximum likelihood estimation of stationary univariate fractionally integrated time series models. *Journal of econometrics*, 53(1-3):165–188, 1992.
35. Statista. Average number of new Android app releases per day from 3rd quarter 2016 to 1st quarter 2018. `https://www.statista.com/statistics/276703/android-app-releases-worldwide/`. Accessed: 2020-01-23.
36. R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.
37. H. Y. Toda and T. Yamamoto. Statistical inference in vector autoregressions with possibly integrated processes. *Journal of econometrics*, 66(1-2):225–250, 1995.
38. C. Wang, J. Li, P. Liang, M. Daneva, and M. van Sinderen. Developers' eyes on the changes of apps: An exploratory study on app changelogs.
39. W. Yuan, Z. Feng, S. Chen, K. Huang, and J. Yao. What biscuits to put in the basket? features prediction in release management for Android system. In *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pages 73–80. IEEE,

2017.

Appendices

Refined Bug, Improvement, and Emergency Related Keywords

Table 11: List of refined *bug-related*, *improvement-related*, and *emergency-related* keywords for identifying release notes that resolve bugs, introduce new features, and provide emergency updates to the app.

Keyword type	**Keywords**
Improvement-related keywords	improve, improved, improvements, enhance, enhances, update, updates, add, optimize
Bug-related keywords	bug, bugs, fix, fixes, workaround, solve, resolved, problem, error, defect, incorrect, incorrectly, issue, issues, crash
Bug-related keywords	emergency, emergencies, urgent