Empirical Software Engineering (2022) 27: 55
https://doi.org/10.1007/s10664-021-10086-2

An empirical study on release notes patterns of popular apps in the Google Play Store

Aidan Z. H. Yang[1] · Safwat Hassan[2] · Ying Zou[3] · Ahmed E. Hassan[4]

Accepted: 17 November 2021 / Published online: 4 March 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

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 app developers in preparing their release notes. In this paper, we are interested in capturing the common practices as release notes patterns. First, we conduct an online survey with 102 respondents to investigate their views on mobile release notes. Our results show that most developers find release notes to be useful for notifying their user-base. Then, 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 manually investigate the release notes from each of the six patterns, and find 17 release drivers for the release notes. We also 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. Our work shows potential directions for developers to improve the release note mechanisms in app stores.

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

Communicated by: Mark Harman

✉ Safwat Hassan
shassan@tru.ca

Extended author information available on the last page of the article.

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 (Statista 2020).

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 (Google 2020; Appcues 2020). 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 (Appcues 2020). Hence, app stores require app developers to describe the updates in their release notes (Google 2020).

Figure 1 shows an example of the release notes of the *"Yard Sale Treasure Map"* app. As shown in Fig. 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"*).

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 (Gao et al. 2018; Hassan et al. 2018b; Hassan et al. 2017). Existing research also relates information of release notes with the success of an app (Martin et al. 2016; Noei et al. 2017). For example, Martin et al. (2016) 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 inform their user-base about the changes in the new releases, 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.

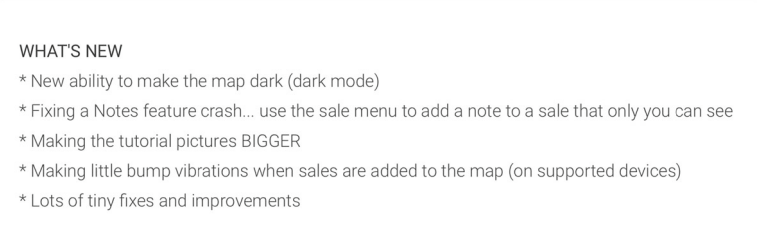


Fig. 1 An example of the release notes of the *"Yard Sale Treasure Map"* app (the release version *"7.6.0"*)

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). In our motivational study (Section 2), we observe that the user of the 2,232 top apps complain about the content of the release notes by leaving negative reviews. Furthermore, we conduct a survey of 102 developers and observe that 90% of the collected responses find it useful to notify the user-base with release notes.

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 communicate app updates to their users. In particular, we aim to identify different patterns of release notes 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 *length* and *updatability* to find different patterns of writing release notes. We measure *length* 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 length (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 attributes (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 attributes. 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 (Brockwell et al. 2002) on the characteristics of every app (e.g., *length* 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 *length* (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 perform a manual analysis to find 17 release drivers described by the release notes of each pattern. Furthermore, 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 provide a motivational study for our work. In Section 3, we outline the experiment setup. Section 4 presents the results with respect to our three research questions. Section 5 presents the implications we draw from the results. Section 6 discusses threats to the validity of our results. In Section 7, we examine related work. Finally, Section 8 draws conclusions to the paper.

2 Motivational Study

We perform a motivational study to determine the importance of release notes for users and developers. In particular, we investigate (1) the user reviewers from our collected app data, (2) how developers discuss release notes on question and answer platforms, (3) the frequency of releases in our collected app data, and (4) surveyed developer responses.

2.1 Analyzing User Reviews

We first read through our collected user reviews and found that users do read release notes. Users also leave negative reviews based on the quality of the release note (e.g., a release note describes a release inaccurately). The following five app reviews all correspond to release notes.

1. "*When you publish release notes instead of saying the blanket bug fixes maybe you should try to make a useful statement instead.* "
2. "*Initial issue below was stated to be resolved in release notes, but it wasn't shown in the app*"
3. "*Setting up fingerprint login should have been announced in the release notes not hidden in the options!* "
4. "*The release notes are not helpful and don't actually explain what changed between minor updates so it's anyone's guess if any particular issue has been addressed or if the update will just make things worse*."
5. "*I rarely care about the feels-like temperature. So thanks but uninstalling. Might check your release notes to see if fix this*."

Review 1 suggests general improvements to the release notes of an app. Reviews 2, 3 and 4 are concerned with the inaccuracies or the lack of information in the release notes. Review 5 suggests that the user is not happy with the status of the app, but will read the release notes in the future to determine if the app is worth using again.

2.2 Analyzing Question and Answer Platforms

We observe from two Stack Overflow questions that developers care about writing their release notes. In the first Stack Overflow question,[1] a user asks about the content developers should usually include in release notes. Other Stack Overflow users recommend that release notes should "summarize the new developments, features, and bug fixes in a broad, sweeping way if possible". In the second Stack Overflow question, a user asks how to generate Azure DevOps release notes.[2]

We find that users also tend to read mobile release notes from a survey conducted on Reddit.[3] The Reddit survey finds that, of the 372 responses from Reddit users, 311 (83.6%) read release notes. 320 users responded that they read release notes to find out "what's new or changed" in an app.[4]

A recent study surveyed mobile developers of different apps to learn that writing a good release note is challenging for developers.[5] For example, the Transit app tries to use release notes to show the personality of the company while keeping release notes informative. Furthermore, experts in software design urge designers to make more engaging release notes[6].

2.3 Analyzing Frequency of Releases

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.

[1] https://stackoverflow.com/questions/638423/how-should-release-notes-be-written

[2] https://stackoverflow.com/questions/56774669/is-there-any-way-to-order-group-work-item-type-in-azure-devops-release-notes

[3] https://medium.com/@scottydocs/do-people-actually-read-release-notes-449d099d73ee

[4] https://medium.com/@scottydocs/do-people-actually-read-release-notes-449d099d73ee

[5] https://uxdesign.cc/design-better-release-notes-3e8c8c785231

[6] https://spectrum.ieee.org/tech-talk/telecom/internet/the-art-of-writing-app-release-notes

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

2.4 Surveying Developers

In addition to release frequency, we survey mobile developers on their perception of writing release notes. We design a survey that consists of questions relating to the background of developers (e.g., the years of experience participants have in mobile development, or the category of the app the participants are currently working on), and their view on the function and importance of release notes (e.g., do the participants use the release notes to respond to user reviews). Table 14 shows the questions we used for the survey. To ensure that the participants of a specific organization or app type do not bias our survey, we send our survey to developers through multiple communications as follows.

We retrieve a list of 922 open-source Android apps from F-Droid.[7] For each app, we send our survey to the contact email addresses of developers provided by the Google Play Store or GitHub. We also retrieve the contact emails from the 2,232 top free-to-download apps used in RQ1-3, and post our survey to Reddit mobile development groups.[8,9] Of the 922 contacted developers, we receive 102 responses (i.e., a response rate of 11%).

Responses to Background Questions Figures 2 and 3 show the developers responses to background questions. 21 % of the participants were test engineers, 19% of them were developers, 17% of them were product support engineers, 15% of them were release management engineers, 15% of them were configuration management engineers, 11% of them were program managers, and 2% were in other roles. We observe that 58% of the participants have 2-5 years of mobile development experience. 30% of the participants are currently working on "Browser" related apps, 20% of them are currently working on "News" related apps, and 19% of them are currently working on "Video Editor" related apps.

Responses to Release Note Practice Questions As shown in Figs. 4 and 5, 51% of the participants responded that they frequently update their release notes, and 4% of the participants responded that they never update their release notes. 90% of the participants agree that it is useful to notify their user base about app updates, and 83% of the participants agree that release notes can be used to respond to user reviews. When asked about methods to help developers write release notes, 78% of the participants agree that they would appreciate a tool for generating release notes.

Summary

Our motivational study shows that users leave negative reviews based on the content of the release notes. Developers from various blogs and question and answer platforms understand the importance of writing release notes for their user-base. From our survey results, we observe that developers frequently update their release notes, and understand the importance in using release notes to update and respond to their users.

[7] https://f-droid.org/en/
[8] https://www.reddit.com/r/androiddev
[9] https://www.reddit.com/r/appdev/

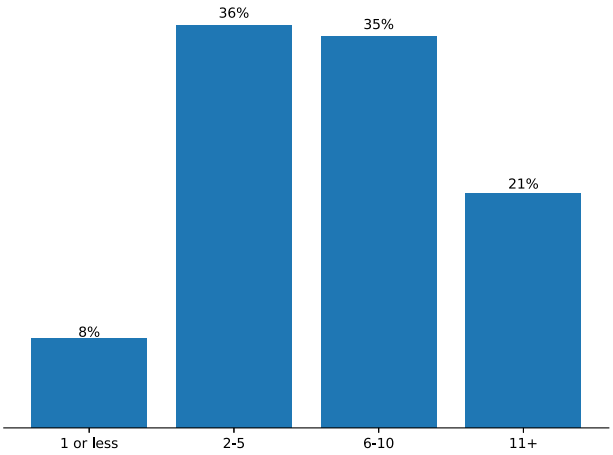


Fig. 2 The distribution of the years of experience of the surveyed developers

3 Experimental Setup

In this section, we outline our data collection and data processing steps. Figure 6 gives an overview of our approach for collecting the release notes and user reviews of the studied apps and analyzing the collected data.

3.1 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 (Noei et al. 2018). Furthermore, we select the most popular apps because they have a large user-base. With larger user-base apps, we can measure the user perception of the apps, the dialogue between developers and users, and how the release notes are involved in the user-developer dialogues.

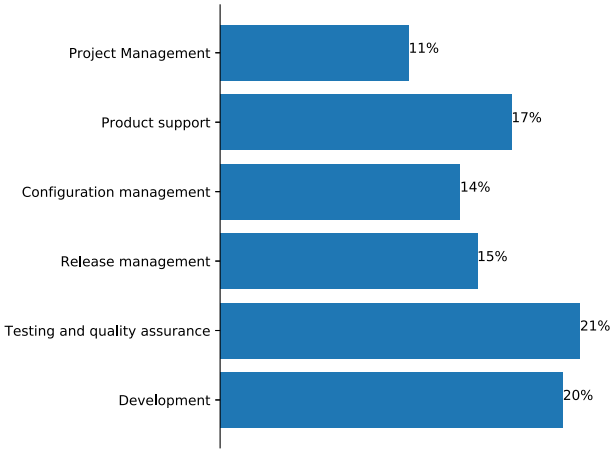


Fig. 3 The distribution of the job roles of the surveyed developers

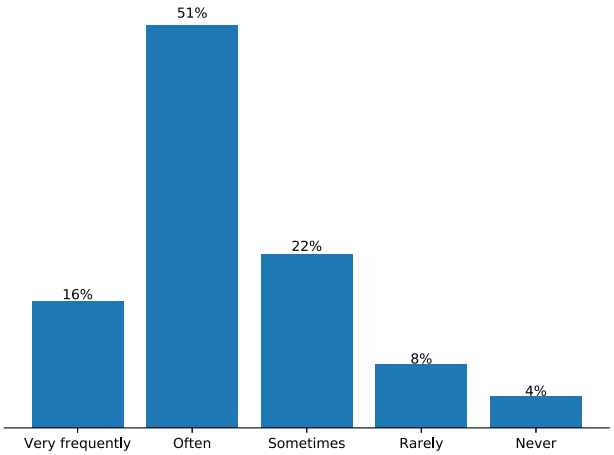


Fig. 4 Developers' answers for frequency of release note update

We collect the list of top popular apps using the App Annie report (App annie 2019). 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 (2019) Google Play crawler to collect data of the selected 2,232 apps. In particular, we collect the following data for each studied app:

(a) **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).
(b) **App releases:** For each release, we collect the release date, release notes content, and the release version string (e.g., 4.2.3).
(c) **User-developer dialogues:** We collect all reviews and developers' responses for each app.

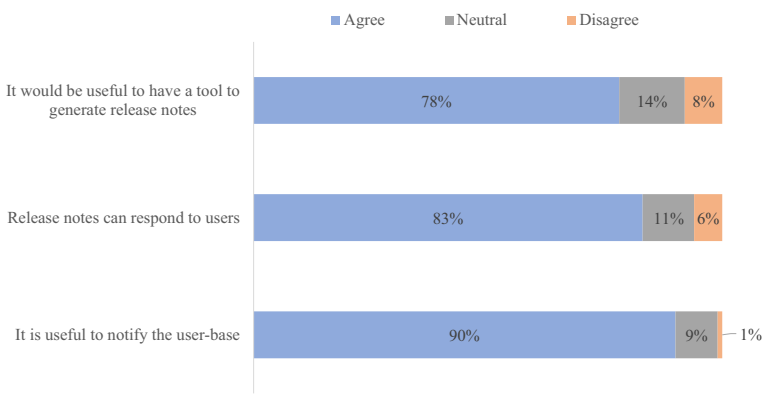


Fig. 5 Developers' answers for release note uses

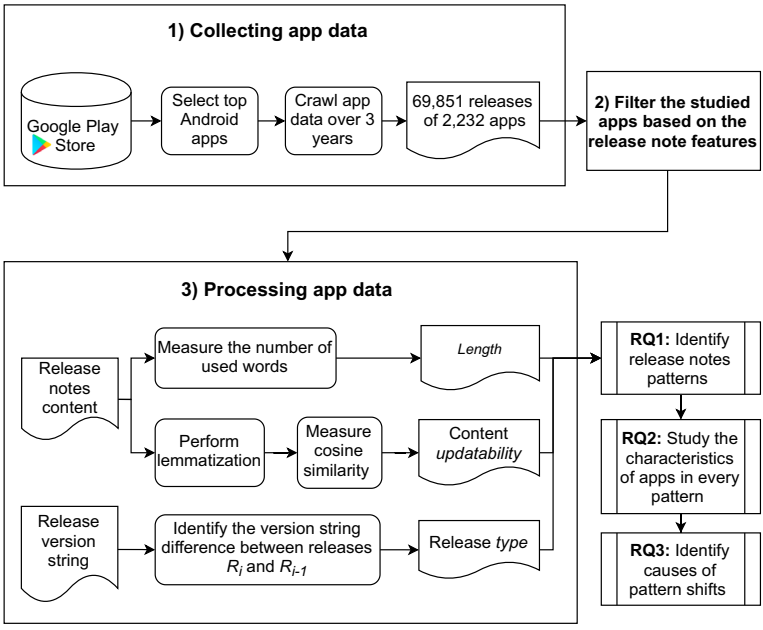


Fig. 6 An overview of our approach for collecting, filtering, and processing release notes data

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.

3.2 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.

3.3 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 attributes that capture the releasing practice of an app.

3.3.1 Identifying the Release Type for the Studied Releases

To identify the release type for the studied releases, we use semantic versioning (Preston-Werner 2019) 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 (Preston-Werner 2019). 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) (Preston-Werner 2019).

3.3.2 Identifying the Release Notes Features

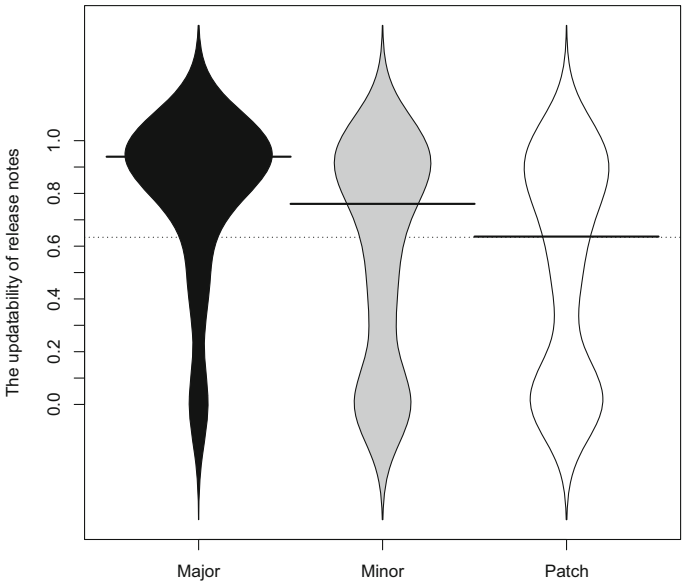
To understand the usage of release notes in practice, we extract three release notes features that are used in RQ1. To derive the release note features, we perform a manual investigation on a randomly selected sample of the 58,069 release notes. Using a confidence level of 95% and a confidence interval of 10%, we manually investigate 96 release notes. We read through all 96 release notes to gain insight on how developers express the new changes in the release notes. We observe that long release notes may include more information (e.g., a list of added features). However, some app developers tend to reuse the prior release notes. Therefore, we design two release notes features based on the length and the cosine similarity of release notes. We also observe that the release type (i.e., major, minor, and patch releases) may impact the cosine similarity of the release notes. Therefore, we design a third release notes feature based on the cosine similarity during a major release. We describe the extracted features as follows.

1. **Length:** We measure the length of the release notes by counting the number of unique 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}.

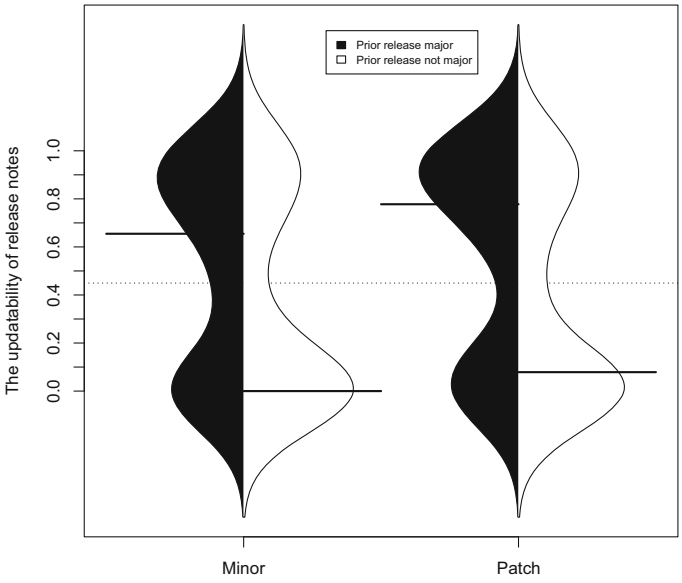
   To measure the cosine similarity, we remove punctuation and perform lemmatization on each word of every release note. Lemmatization is the process that converts all words of a sentence into its base form. For example, the lemmatization of "fixing" is "fix" (Plisson et al. 2004). We use Stanford CoreNLP (Manning et al. 2014) 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 (Fig. 7a). For minor and patch releases, the release notes are frequently updated after major releases (Fig. 7b). From the two observations regarding major releases, we



(a) The distribution of updatability by the release type



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

Fig. 7 The updatability of the studied release notes (grouped by the release types)

use content updatability for only major releases as a feature to identify release note patterns.

To ensure that the length, 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 (Meng et al. 1992). We find that length 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 length 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.

4 Results

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

4.1 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 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.

Step 1: ***Deriving app attributes that capture the releasing practice of an app.*** We use three features (i.e., length, updatability, and updatability for only major releases) to capture the characteristics of release notes practice of an app (as described in Section 3).

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 (Hartigan and Wong 1979) on the summarized release notes features to find different release notes patterns from the clusters. We find the optimum number of clusters from the Gap Statistic method (Tibshirani et al. 2001).

Step 4: ***Evaluating app releasing patterns.*** We use the silhouette validation technique to evaluate our clustering algorithm. The silhouette validation (Rousseeuw 1987) measures the separation distance between each cluster. We compare the

Table 2 A summary of the six identified patterns

Pattern ID	Pattern Name	Number of Apps	Length	Updatability	Updatability for Major Releases[10]
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%

dissimilarity of release notes with other release notes of the same cluster to find the silhouette value $S(i)$, as defined in (1).

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

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 (Rousseeuw 1987). 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.

In addition to the clustering process based on syntactic features, we perform a manual analysis to further understand the release notes in every pattern. The manual analysis does not attempt to find further classification. Instead, we need to understand the drivers for releasing a new release. Our manual analysis includes the following steps:

Step 1: ***Select sample of the studied apps***. Manually analyzing the release notes of all apps is a tedious and time consuming process. Hence, we selected a statistically representative random sample of release notes that belong to the identified six patterns (with a confidence level of 95% and a confidence interval of 10%). In total, we randomly selected 559 release notes belonging to 405 apps. Table 3 shows the characteristics of the selected release notes.

[10] 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

Table 3 The sample size for each pattern for manual analysis

	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6
Total number of apps	439	111	157	572	145	288
Total number of release notes	11704	1044	4467	24786	1840	9295
Sample number of apps	71	55	58	86	57	78
Sample number of release notes	95	88	94	95	91	95

Step 2: ***Identify the drivers for the new release.*** The first and the second author of this paper manually read the release notes of every release independently and identified the drivers for releasing this release. For example, a release with the release notes *"Fix for Motorla Android 8"* has a driver *"platform and device support"*. When a new driver is identified during the analysis of the release notes, it is added to the list of drivers, and all release notes were reanalyzed using the new list of identified drivers. During this process, we conducted three revisits of all release notes to identify all drivers. We conclude our analysis after reading all 559 release notes and no further drivers were found.

Step 3: ***Finalize the list of drivers of each release.*** For every studied release note, we compared the two lists of identified drivers that are generated by the two coders . We then calculate the Kohen's Kappa inter-rater agreement (Cohen 1960) between the two authors. Cohen's Kappa value ranges from -1 to +1. A Cohen's Kappa value of +1 means that both coders identified the same drivers for all analyzed release notes. For our analysis, we calculate a Cohen's Kappa value of 0.90, which is an almost perfect agreement according to the interpretation of the Cohen's Kappa value proposed by Landis and Koch (1977).

As a result of our manual analysis, we manually label the 559 release notes with 17 release drivers. As shown in Table 4, we found a total of 17 drivers. Table 3 shows the statistically representative number of release notes and apps selected. Table 5 shows the frequency of the 17 labels for the six patterns. For each driver in Table 5, we highlight the highest frequency release note pattern. In general, we observe that apps with shorter release notes (i.e., apps in patterns 1, 2 and 3) tend to have more release notes describing general bug fixes. Apps with longer and more frequently updated release notes (i.e., apps in patterns 5 and 6) tend to have more release notes describing added features or UI changes (42% of pattern 4 release notes describe new features, 22% of pattern 6 release notes describe UI changes). Apps in patterns 2 and 5 (i.e., apps with updating release notes) tend to describe platform device specific fixes (14% and 17%, respectively), and detailed bug fixes (20% and 25%, respectively).

Tables 6 and 7 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.

Table 4 Types of content for release notes in the 91 manually investigated apps

Release driver	Description	Example
General bug fix	The release notes contain bug fixes without mentioning the details of the fixed bugs.	*"Bug fix", "fix some bugs", and "Bug fixes"*
General improvements	The release notes contain improvements without mentioning the details of such improvements.	*"New improvements."*
Detailed bug fix	The release notes describe the details of the fixed issues.	*"Fixed issues with re-borrowed titles."*
Detailed improvements	The release notes describe the details of the feature improvements.	*"Improved interstitial ads display."*
Performance	The release notes describe an improvement to the performance of an app.	*"Enhanced invoice performance."*
New features	The release introduces new features to the user-base.	*"Added some new emoji resource"*
Coming soon	The release notes announce features coming soon to the app.	*"New game coming soon."*
Permission and security	The release notes contain permission and security-related changes.	*"Added Privacy Policy."*
Platform and device support	The release notes contain changes accommodating software or hardware updates (e.g., Android version changes).	*"Fix for Motorola Android 8."*
UI	The release notes contain user interface (UI) changes, such as changing the layout, the colors, or the UI design of the app.	*"Icons added to all overflow menus."*
Localization	The release notes provide language-related changes.	*"Added Ukrainian language."*
Crash	The release notes contain crash fixes.	*"Fixed a crash on empty comments."*
Resource utilization	Resource improvements (e.g., app size, memory, battery usage).	*"Game improvements for memory."*
Ask for feedback	The release notes ask users for their feedback.	*"Found a bug we missed? Please let us know here: ..."*
Advertisement	The release notes promote new offers or discounts.	*"Limited time pricing for new subscribers now $9.99/month."*
Provide support	The release notes guide users on how to handle issues or to use new features.	*"Tip: save a search and we'll let you know when new homes hit the market that meet your criteria."*
Non-informative	The release notes do not mention any information about the release changes.	*"The long-awaited major update is finally here!"*

Table 5 The frequency of release drivers in the 91 manually investigated apps

Drivers	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6
General bug fix	**56%**	26%	40%	35%	21%	36%
Detailed bug fix	10%	20%	3%	9%	**25%**	14%
General improvement	23%	12%	**26%**	25%	24%	17%
Detailed improvement	5%	7%	6%	10%	**15%**	5%
Performance	7%	4%	4%	**9%**	2%	6%
New feature	15%	21%	25%	42%	40%	40%
Permission and security	0%	3%	2%	4%	**8%**	1%
Platform Device Support	6%	14%	7%	4%	**17%**	4%
UI	9%	9%	12%	21%	19%	**22%**
Localization	1%	3%	1%	5%	**6%**	2%
Crash	1%	5%	4%	4%	**6%**	5%
Resource utilization	4%	0%	2%	1%	**12%**	4%
Ask for feedback	0%	3%	1%	**24%**	9%	12%
Advertisement	0%	3%	1%	**10%**	4%	7%
Non informative	0%	1%	1%	**8%**	2%	4%
Provide support	3%	2%	0%	0%	**8%**	2%

After the first two authors read through a statistically representative random sample of 95 release notes 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 observe from Table 5 that 56% of the release notes in Pattern 1 mention general bug fixes, and 23% of the release notes in Pattern 1 mention general improvements. 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 6, 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).

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 6, 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 a statistically representative random sample of 94 release notes that belong to pattern 2, we observe that release notes in this pattern frequently mention both general and detailed bug fixes (26% and 20%, respectively), as well as new features and support for devices and platforms (21% and 14%, respectively). 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

Table 6 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."*

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 6) 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.

After the first two authors read through a statistically representative random sample of 94 release notes that belong to pattern 3, we observe that release notes in this pattern mostly describe general bug fixes and improvements (40% and 26%, respectively).

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 show in Table 7, the releases of the *"NYC Subway Map"* app contain long release notes. However, the content of the release notes is not updated.

After the first two authors read through a statistically representative random sample of 95 release notes that belong to pattern 4, we observe that release notes in this pattern tend to mention new features and ask for user feedback (42% and 24%, respectively).

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 The first two authors read through a statistically representative random sample of 91 release notes that belong to pattern 5. We find that release notes in pattern 5 have detailed and descriptive release notes that are frequently updated. In particular, release notes frequently describe the added or improved functionalities in detail (40% describe new features, and 15% describe detailed improvements). For example, developers of the *"Yard Sale Treasure Map"* app (Table 7) 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 release notes in Pattern 5 provide more platform specific support (17%), resource utilization support (12%), and user specific support (8%) as compared to release notes of other patterns. We observe from specific examples that app developers in pattern 5 change their release notes due to negative reviews and app crashes. For example, the *"My 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 The first two authors read through a statistically representative random sample of 95 release notes that belong to pattern 6. Apps in pattern 6 have repetitive release notes during minor and patch releases that mention newly added features (40%) and general bug fixes (36%). Release notes are updated as the major releases and the updated release notes are generally written for large functionality update, such as UI changes (22%) and detailed bug fixes (14%) For example, the *"Shopular"* app (Table 7) contains mostly repetitive

Table 7 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!"*

release notes except for 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 way to broadcast a response to many user reviews. For example, we observe from Table 8 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.

Through the manual investigation of the six identified patterns from our representative sample of 559 apps, 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.

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

Table 8 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"*

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 and 17 drivers for the new releases. 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 length and updatability, of the release notes. In this RQ, we are interested in discovering the common attributes 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 attributes and each of the six identified patterns. Our approach consists of three steps: (1) collecting the app-related attributes, (2) constructing the models, and (3) analyzing the constructed models.

Step 1: ***Collecting the app-related attributes.*** We use app specific attributes as independent variables in predicting whether an app belongs to pattern i, where $i \in \{1, 2, .., 6\}$. As listed in Table 9, we measure 11 app-related attributes that can potentially correlate to release notes styles. We choose these attributes based on prior research regarding the Google Play Store apps and release notes (Noei et al. 2019; Martin et al. 2016; Hassan et al. 2017; Hassan et al. 2018a; Noei et al. 2017).

For the keyword attributes (i.e., *bug*-related releases, *improvement*-related releases, and *emergency*-related releases), we use *word2vec* (Goldberg and Levy 2014) 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 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 attributes. We measure the correlation between the collected attributes using the Spearman rank correlation test. We use a cut-off value for ρ of 0.7 (Noei et al. 2017). After the correlation process, the "*percentage of low ratings*" attribute and the "*average release rating*" attribute are highly correlated ($\rho = 0.78$). The "*release count*" attribute and the "*release frequency*" attribute are highly correlated ($\rho = 0.71$). Hence, the "*percentage of low ratings*" attribute and the "*release frequency*" attribute are removed from our attributes.

Step 2: ***Constructing the models.*** We construct six logistic regression models using the **glm** package (Marschner et al. 2018). 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* (Khoshgoftaar and Allen 1999; Camargo Cruz and Ochimizu 2009; Nagappan et al. 2008; Gyimothy et al. 2005).

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 (Huang and Ling 2005).

To understand the most important attributes in our logistic regression models and identify the attributes that are most contributing to each release notes pattern, we use Wald statistics (χ^2) to estimate the relative contribution of each app attribute in the constructed model. The larger the χ^2 value, the larger the impact that a particular attribute has on the performance of the model (i.e., the AUC value) (Toda and Yamamoto 1995; Moore 1977). We then calculate the *percentage* of χ^2 for each attribute to the total χ^2 for all app attributes (Hassan et al. 2018a). We also analyze the *coefficient sign* for each app attribute in the generated models. A positive coefficient sign of an app attribute indicates that the app attribute

Table 9 The collected attributes for each app

Attribute	Values	Description (D) - Rationale (R)
App metadata attributes		
App category	Categorical	**D:** Category of the studied app (i.e., either *Game* or *non-Game* app). **R:** Developers from different app categories may write release notes differently.
App size	Numerical	Size of the APK file in MB. **R:** Developers from apps with more functionalities may write more detailed release notes.
Size of the user-base	Numerical	The number of downloads. **R:** Developers working on apps that have a larger user-base may write more detailed release notes.
Store Experience	Numerical	The total number of apps offered by the owner of the app. **R:** App stores with more experience may write more detailed release notes.
Release practice attributes		
Release count	Numerical	**D:** The total number of releases for every app during the studied period. **R:** Apps with more releases may write less detailed release notes.
Release frequency	Numerical	**D:** The median time-to-release for every app. **R:** Apps that release more frequently may write less detailed release notes.
Bug keywords	Numerical	**D:** The percentage of releases with *bug-related* keywords in the release notes text. **R:** Apps that write release notes containing "bug-fix" may write shorter release notes.
Improvement keywords	Numerical	**D:** The percentage of releases with *improvement-related* keywords in the release notes text. **R:** Apps that write release notes containing *improvement-related* keywords may write shorter release notes.
Emergency keywords	Numerical	**D:** The percentage of releases with *emergency-related* keywords in the release notes text. **R:** Apps that write release notes containing *emergency-related* keywords may write shorter release notes.
Same-day releases	Numerical	**D:** The percentage of releases that have time-to-release of a single day. **R:** Apps that have a large amount of same-day releases may right shorter release notes.
Response rate	Numerical	**D:** The ratio of developer responses to the posted reviews of an app. **R:** Apps that frequently respond to users might have more detailed release notes.
Perceived quality attributes		
Average release rating	Numerical	**D:** The average star ratings of the deployed releases of an app. **R:** Developers may tend to write more detailed release notes after receiving reviews with a low rating to increase the overall rating of their app.
Percentage of low ratings	Numerical	**D:** The percentage of one and two star ratings for all the deployed releases of an app. **R:** Developers may tend to write more detailed release notes after receiving reviews with a low rating to increase the overall rating of their app.
Rating increase	Categorical	**D:** 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). **R:** Apps that have an increase in ratings might have more changes and more detailed release notes.

Table 10 A summary of the analysis of the constructed six models

Discriminatory power of the constructed models						
Model statistics	Model 1[11]	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 attributes on the six models						
App attributes	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	0.11 (-)	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 (+)
Store Experience	0.02 (-)	0.03 (+)	0 (+)	0.01 (+)	0.02 (+)	0.06 (+)
Release count	0.00 (-)	**0.22 (-)**	0.06 (-)	**0.44 (+)**	**0.37 (-)**	0.00 (-)
Bug keywords	**0.27 (+)**[12]	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 (-)**	0.06 (+)	0.05 (-)	0.02 (+)	0.01 (+)	0.06 (+)
Same-day releases	0.00 (+)	0.10 (+)	0.02 (+)	**0.19 (+)**	0.02 (-)	0.08 (-)
Response frequency	0.14 (-)	0.03 (-)	0.03 (+)	0.01 (+)	**0.20 (+)**	**0.22 (+)**
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 (+)	0.17 (+)	0.15 (+)

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 attribute signifies that the app attribute negatively correlates to our independent variable.

Results: The constructed models for the apps with short updating release notes have the highest explanatory power Table 10 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 attribute 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

[11] 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*.

[12] The bold text highlights the app attributes with the highest impact on the response variable.

correlate to average release ratings, meaning that apps 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 attributes (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 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.

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 descriptive release notes that contain more comprehensive content.

Apps in pattern 6 tend to have the highest response rate. As responding to use reviews results in a positive change in review ratings (Hassan et al. 2018a), 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.

Furthermore, we observe that apps under the "Games" category are negatively correlated with release notes in patterns 2 and 6. Therefore, our model suggests that it is less likely for "Game" category apps to contain updating release note patterns.

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 communication with their app users.

Approach For all the studied apps, we treat each of the clustering metrics (e.g., length 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 (Brockwell et al. 2002). We use a *stationary p-value* threshold of 0.05, where a time series above the threshold is *non-stationary* (Sowell 1992). 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 association 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. We calculate a Cohen's Kappa value of 0.89 for the inter-rater agreement (Cohen 1960).

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. Figure 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

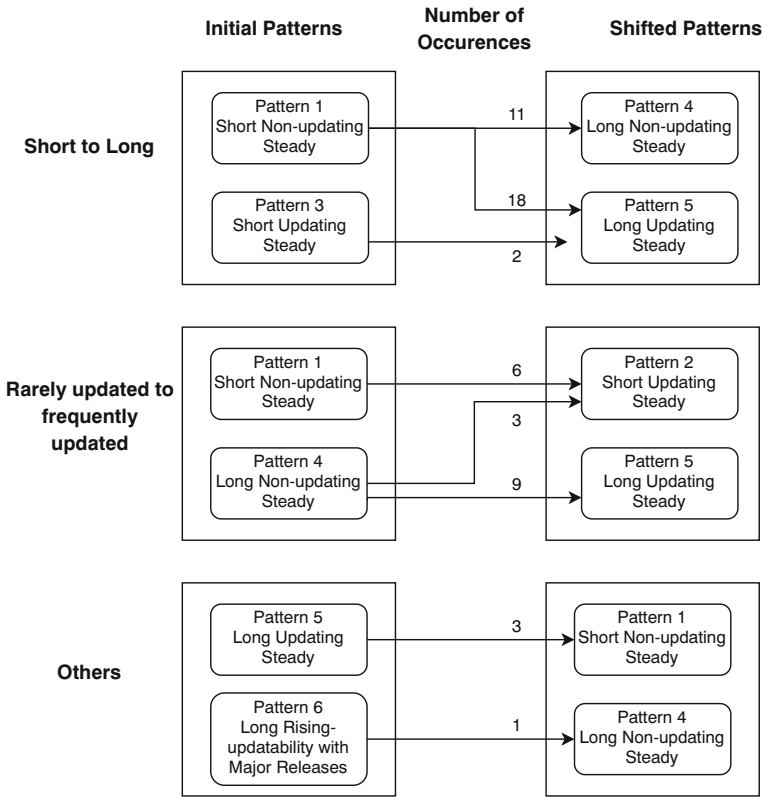


Fig. 8 A summary of pattern shifts

updated (18 apps). The results show that when apps shift patterns, developers tend to add more 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 11, 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"*.

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

Table 11 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

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. We perform the Wilcoxon rank-sum statistical test (Cuzick 1985) between the ratings before shifts and ratings after shifts. As the p-value turns out to be 4.78e-05, which is less than the 0.05 significance level, we reject the null hypothesis and conclude that we observe a significant difference in user ratings from either short to long release notes or from rarely updated to frequently updated release notes.

Table 12 shows an example of release notes from three apps to demonstrate a release notes pattern shift. As listed in Table 12, 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.

5 Implications

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

Table 12 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 ..."*

Implications for app developers In RQ1, we observe that developers in pattern 5 and 6 use release notes to inform their users about the changes in the new release. For example, 8% of analyzed release notes of pattern 5 provide support for app users. Prior studies proposed approaches to automatically generate the release notes (Moreno et al. 2014). The automatically generated release notes can help app users understand the introduced changes in the new releases. Hence, we recommend that developers of pattern 5 can leverage the existing tools to automatically generate informative release notes.

In the conducted survey, we observe that 90% of the 102 survey participants agree that it is useful to notify their user base about app updates. We identify a taxonomy of 17 drivers mentioned in the release notes. Hence, developers in patterns 1 and 4 (i.e., developers who do not update their release notes) can benefit from the provided taxonomy to improve the documented information in their release notes. For example, developers of patterns 1

and 4 can mention details about the improved features or the fixed issues (e.g., fixing the performance and the resources utilization issues) in their release notes.

Finally, we find that app developers notify users individually (i.e., responding to the posted reviews) with the new updates written in the release notes. Prior work proposed approaches for the automatic generation of developer responses (Gao et al. 2019; Gao et al. 2020). Developers of patterns 5 and 6 (i.e., patterns with the highest response rate to app users) can use the existing response generation tools to respond to user reviews and inform users about the changes in the new releases. For example, as shown in Table 8, 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. For specific user issues, the developer response should be tailored for the user. For general issues (i.e., a commonly occurring bug in the latest update), our study suggests that high quality release notes can be used to help developers broadcast their responses to their user base. From our conducted survey, we also observe that 83% of the participants agree that release notes can be used to respond to user reviews.

Implication for store owners 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 (Hassan et al. 2018a). Hence, store owners can recommend users the apps that have: (1) release notes that are more frequently updated, (2) releases with higher perceived quality, and (3) a higher user response rate than other apps. Based on our analysis, we find that apps in pattern 5 are the only apps that follow all three criteria.

Implications for researchers Software engineering research can benefit from our work to be careful when using release notes as an artifact that reflects the changes in the latest release (i.e, release log). As discussed in RQ2, the most common patterns for our studied apps are the non-updating patterns. For example, using the current text in the release notes may not indicate the actual changes in the release, as we find that 49% of the releases use the previously mentioned release notes. We also noted that long repetitive release notes (i.e., pattern 4) mainly contain "advertisement" and "non-informative" release drivers. 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.

6 Threats to Validity

6.1 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 release notes to understand why their shift occurred.

6.2 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 attributes. We select app attributes 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 the coding experience of the developers. Developer information (e.g., developers' experience) 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 analyzing the characteristics of the releases in open source apps (e.g., F-Droid apps).

6.3 External Validity

External threats to validity are concerned with our ability to generalize our results. Our study focuses on analyzing the 2,232 apps in the Google Play Store over a period of three years. Furthermore, our study focuses on the 2,232 apps that are the most popular in the Google Play Store. Our findings may not hold for non-popular apps. To eliminate the bias in our results, we select 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.

We assume throughout this paper that the posted release notes are written by app developers. We were not able to collect information on the structure of the teams that built the studied apps. We observe from Clutch,[13] a company that lists 2,532 mobile app development firms, that 1,732 out of 2,532 (68%) of the app development firms have less than 50 employees. Hence, while we do not have strong evidence that responses were written by actual software developers, we do have evidence that most mobile app development firms are small. Due to the small team size, it is likely that responses to reviews are posted either by app developers themselves or by other team members who are in close contact with the app developers. Future studies should investigate further the team size and structure of mobile app development teams and their impact on the release notes patterns of mobile apps.

7 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.

[13] https://clutch.co/

7.1 Empirical Study of Release Notes

Abebe et al. (2016) 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. (2016) 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. (2019) 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. (2016) 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.

Ahasanuzzaman et al. (2020) manually analyze the release notes of 163 releases of 8 ad libraries. Ahasanuzzaman et al. observed that fixing issues that are related to displaying video ads is the main driver for releasing new versions. From our manual analysis, we observe that Google Play Store app release notes do not only describe general app updates (e.g., "fix bugs"), but also describe app specific updates (e.g., "fixed the audio playback issue"). Recently, Bi et al. (2020) analyzed 32,425 release notes in 1,000 GitHub projects, and found eight categories of information provided in the release notes. Our work is the first to analyze the evolution of release notes and identify the shifts between each release note patterns.

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. Based on manual analysis, we show that the six patterns contain different release note drivers. Our work is useful for researchers to further study how different release notes correlate to high perceived quality of apps.

7.2 Automatic Generation of Release Notes

Moreno et al. (2014) 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 (Moreno et al. 2014), Moreno et al. (2016) 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. (2017) 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 (2018) 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. (2016) 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.

7.3 Emergency Releases

Wang et al. (2019) 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. (2018) 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. (2017) 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. (2018b) 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.

8 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 length 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 further study on iOS apps or paid apps on the Google Play Store.

Appendix

Table 13 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

Table 14 List of defined questions in the conducted survey

Questions	Possible answers
Q1.1 What is your age?	"Less than 25 years old", "26-35 years old", "36-45 years old", "Above 46 years old", and "Prefer not to answer"
Q1.2 How many years of work experience do you have?	"1 or less", "2-5 years", "6-10 years", "11-20 years", and "More than 21 years"
Q1.3 How many years of work experience in the field of mobile development do you have?	"1 or less", "2-5 years", "6-10 years", and "11+ years"
Q1.4 What are your roles in the development of mobile apps? Please select all that apply.	"Development", "Testing and quality assurance", "Release management", "Configuration management", "Product support", "Project/product management", "Other"
Q1.5 How many mobile apps have you developed (including the current one)	"None", "1", "2-5", "6-10", and "11+"
Q1.6 Which of the following industries best describe the category of your app? Please select all that apply.	"Weather", "Bible", "Browser", "Navigation", "Free Call", "SMS", "Music player", "News", "Security", "Wallpaper", "Taxi and rideshare", "Dating", "Recipe cooking", "Coloring", "Pregnancy", "Sports news", "Video editor", "Notes", "Mobile banking apps", "Accommodation booking", "Games", "Other"
Q2.1 How frequently do you update your app release notes?	"Very frequently", "Often", "Sometimes", "Rarely", and "Never"
Q2.2 Do you agree or disagree with the following statement: it is useful to notify the user-base about new updates in your app.	"Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree"
Q2.3 Do you agree or disagree with the following statement: release notes can be used to respond to user reviews.	"Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree"
Q2.4 Do you agree or disagree with the following statement: it would be useful to have a tool that automatically generates release notes for you.	"Strongly agree", "Agree", "Neither agree nor disagree", "Disagree", and "Strongly disagree"

References

App annie (2019) The app analytics and app data industry standard. https://www.appannie.com/. Accessed: 2019-08-30

Abebe SL, Ali N, Hassan AE (2016) An empirical study of software release notes. Empir Softw Eng 21(3):1107–1142

Ahasanuzzaman M, Hassan S, Bezemer C-P, Hassan AE (2020) A longitudinal study of popular ad libraries in the Google Play Store. Empir Softw Eng 25(1):824–858

Akdeniz (2019) Google Play Crawler. https://github.com/Akdeniz/google-play-crawler, Feb 2014. Accessed: 2019-09-30

Appcues (2020) 5 excellent product release note examples and how to write your own. https://www.appcues.com/blog/release-notes-examples. Accessed: 2020-01-23
Bi T, Xia X, Lo D, Grundy J, Zimmermann T (2020) An empirical study of release note production and usage in practice. IEEE Trans Softw Eng
Brockwell PJ, Davis RA, Calder MV (2002) Introduction to time series and forecasting, vol 2. Springer
Camargo Cruz AE, Ochimizu K (2009) 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. IEEE Computer Society, pp 460–463
Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46
Cuzick J (1985) A wilcoxon-type test for trend. Stat Med 4(1):87–90
Gao C, Zeng J, Lyu MR, King I (2018) Online app review analysis for identifying emerging issues. In: 2018 IEEE/ACM 40Th international conference on software engineering (ICSE). IEEE, pp 48–58
Gao C, Zeng J, Xia X, Lo D, Lyu MR, King I (2019) Automating app review response generation. In: 2019 34Th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 163–175
Gao C, Zhou W, Xia X, Lo D, Xie Q, Lyu MR (2020) Automating app review response generation based on contextual knowledge. arXiv:2010.06301
Goldberg Y, Levy O (2014) word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv:1402.3722
Google (2020) Prepare & roll out releases - play console help. https://support.google.com/googleplay/android-developer/answer/7159011?hl=en. Accessed: 2020-01-23
Gyimothy T, Ferenc R, Siket I (2005) Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng 31(10):897–910
Hartigan JA, Wong MA (1979) Algorithm as 136: a k-means clustering algorithm. J Royal Stat Soc Ser C (Applied Statistics) 28(1):100–108
Hassan C, Bezemer C-P, Hassan AE (2018a) Tantithamthavorn Studying the dialogue between users and developers of free apps in the Google Play Store. Empir Softw Eng 23(3):1275–1312
Hassan S, Bezemer C-P, Hassan AE (2018b) Studying bad updates of top free-to-download apps in the Google Play Store. IEEE Trans Softw Eng
Hassan S, Shang W, Hassan AE (2017) An empirical study of emergency updates for top Android mobile apps. Empir Softw Eng 22(1):505–546
Huang J, Ling CX (2005) Using auc and accuracy in evaluating learning algorithms. IEEE Trans Knowl Data Eng 17(3):299–310
Khalfallah M (2018) Generation and visualization of release notes for systems engineering software. In: Proceedings of the international conference on complex systems design & management. Springer, pp 133–144
Khoshgoftaar TM, Allen EB (1999) Logistic regression modeling of software quality. Int J Reliab Qual Safety Eng 6(04):303–317
Klepper S, Krusche S, Bruegge B (2016) Semi-automatic generation of audience-specific release notes. In: Proceedings of the 2016 IEEE/ACM international workshop on continuous software evolution and delivery (CSED). IEEE, pp 19–22
Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics: 159–174
Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S, McClosky D (2014) The stanford corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pp 55–60
Marschner I, Donoghoe MW, Donoghoe MMW (2018) Package 'glm2'. J Vol 3(2):12–15
Martin W, Sarro F, Harman M (2016) Causal impact analysis for app releases in Google Play. In: Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. ACM, pp 435–446
McIlroy S, Ali N, Hassan AE (2016) Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. Empir Softw Eng 21(3):1346–1370
Meng X-L, Rosenthal R, Rubin DB (1992) Comparing correlated correlation coefficients. Psychol Bull 111(1):172
Moore DS (1977) Generalized inverses, wald's method, and the construction of chi-squared tests of fit. J Am Stat Assoc 72(357):131–137
Moreno L, Bavota G, Di Penta M, Oliveto R, Marcus A, Canfora G (2014) Automatic generation of release notes. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering. ACM, pp 484–495
Moreno L, Bavota G, Di Penta M, Oliveto R, Marcus A, Canfora G (2016) Arena: an approach for the automated generation of release notes. IEEE Trans Softw Eng 43(2):106–127

Nagappan N, Murphy B, Basili V (2008) The influence of organizational structure on software quality. In: Proceedings of the 2008 ACM/IEEE 30th international conference on software engineering. IEEE, pp 521–530

Noei E, Da Costa DA, Zou Y (2018) 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. ACM, pp 283–294

Noei E, Syer MD, Zou Y, Hassan AE, Keivanloo I (2017) A study of the relation of mobile device attributes with the user-perceived quality of Android apps. Empir Softw Eng 22(6):3088–3116

Noei E, Zhang F, Zou Y (2019) Too many user-reviews, what should app developers look at first?IEEE Trans Softw Eng

Plisson J, Lavrac N, Mladenic D et al (2004) A rule based approach to word lemmatization. Proceedings of IS-2004:83–86

Preston-Werner T (2019) Semantic versioning 2.0.0. https://spectrum.ieee.org/tech-talk/telecom/internet/the-art-of-writing-app-release-notes. Accessed: 2019-08-30

Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65

Sowell F (1992) Maximum likelihood estimation of stationary univariate fractionally integrated time series models. J Econ 53(1-3):165–188

Statista (2020) 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.

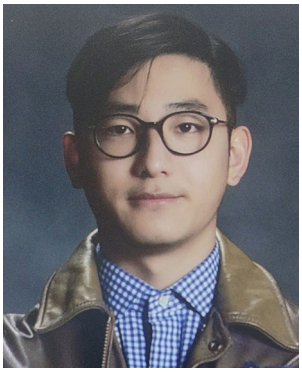
Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a data set via the gap statistic. J Royal Stat Soc Ser B (Statistical Methodology) 63(2):411–423

Toda HY, Yamamoto T (1995) Statistical inference in vector autoregressions with possibly integrated processes. J Econ 66(1-2):225–250

Wang C, Li J, Liang P, Daneva M, Sinderen M (2019) Developers' eyes on the changes of apps: an exploratory study on app changelogs. In: 2019 IEEE 27Th international requirements engineering conference workshops (REW). IEEE, pp 207–212

Yuan W, Feng Z, Chen S, Huang K, Yao J (2017) 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). IEEE, pp 73–80

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Aidan Z. H. Yang is a PhD candidate in the Institute of Software Research (ISR) at Carnegie Mellon University. Before that, he was an undergraduate research assistant at the Software Evolution and Analytics Lab (SEAL) at Queen's University in Kingston, Canada. His research interests include program analysis, program synthesis, and machine learning. His work has been published at premier software engineering venues such as the MSR and ICSE conferences. He obtained his BEng from Queen's University (Department of Mathematics and Engineering). Contact him at aidan@cmu.edu.

Safwat Hassan is an assistant professor in the Department of Engineering at Thompson Rivers University, Canada. Hassan completed his Ph.D. degree at Queen's University in the Software Analysis and Intelligence Lab (SAIL). Hassan worked as a software engineer for ten years in different corporations, including the Egyptian Space Agency (ESA), HP, EDS, VF Germany (outsourced by HP), and Etisalat. During his ten years in the software industry, he worked on multiple software systems, such as web-based systems and embedded systems. He also participated in diverse project roles (e.g., design service, customer support, and R&D) across multiple business domains (e.g., telecommunication, supply-chain, and aerospace). His research interests include data mining for software engineering, mobile app store analytics, software architecture, system anomaly prediction, continuous integration, and software performance analytics. More about Safwat Hassan can be read on his website: https://safwathassan.com

Ying Zou is the Canada Research Chair in Software Evolution. She is a professor in the Department of Electrical and Computer Engineering, and cross-appointed to the School of Computing at Queen's University in Canada. She is a visiting scientist of IBM Centers for Advanced Studies, IBM Canada. Her research interests include software engineering, software evolution, software analytics, and service-oriented architecture. More about Ying and her work is available online at https://www.ece.queensu.ca/people/Y-Zou/

Ahmed E. Hassan is an IEEE Fellow, an ACM SIGSOFT Influential Educator, an IEEE TCSE Distinguished Educator, an NSERC Steacie Fellow, 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. He 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. He also serves/d on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. Contact ahmedcs.queensu.ca. More information at: http://sail.cs.queensu.ca/

Affiliations

Aidan Z. H. Yang[1] · Safwat Hassan[2] · Ying Zou[3] · Ahmed E. Hassan[4]

Aidan Z. H. Yang
aidan@cmu.edu

Ying Zou
yingzou@queensu.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

1 School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

2 Department of Engineering, Thompson Rivers University, Kamloops, British Columbia, Canada

3 Software Evolution and Analytics Lab (SEAL), Queen's University, Kingston, Ontario, Canada

4 Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, Ontario, Canada