Empir Software Eng
DOI 10.1007/s10664-015-9388-2

Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store

Stuart McIlroy[1] · **Nasir Ali**[2] · **Ahmed E. Hassan**[1]

Abstract Mobile app stores provide a unique platform for developers to rapidly deploy new updates of their apps. We studied the frequency of updates of 10,713 mobile apps (the top free 400 apps at the start of 2014 in each of the 30 categories in the Google Play store). We find that a small subset of these apps (98 apps representing ˜1 % of the studied apps) are updated at a very frequent rate — more than one update per week and 14 % of the studied apps are updated on a bi-weekly basis (or more frequently). We observed that 45 % of the frequently-updated apps do not provide the users with any information about the rationale for the new updates and updates exhibit a median growth in size of 6 %. This paper provides information regarding the update strategies employed by the top mobile apps. The results of our study show that 1) developers should not shy away from updating their apps very frequently, however the frequency varies across store categories. 2) Developers do not need to be too concerned about detailing the content of new updates. It appears that users are not too concerned about such information. 3) Users highly rank frequently-updated apps instead of being annoyed about the high update frequency.

Communicated by: Andreas Zeller

✉ Stuart McIlroy
mcilroy@cs.queensu.ca

Nasir Ali
nasir.ali@uwaterloo.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

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

[2] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

Published online: 07 July 2015

Keywords Mobile apps · Release patterns · Reverse engineering · Source code analysis · Software releases · Release frequency

1 Introduction

Mobile app stores provide a unique platform that enables developers to rapidly update and deploy new updates of their apps. Popular mobile app stores include the Apple App Store, the Blackberry World Store, the Google Play Store, and Microsoft Phone Apps Store. Once a developer uploads a new update of an app, the update is available to the app's user base immediately with no cost to the developer. Such a rapid and low cost deployment platform is very unique compared to traditional software applications.

The frequency of releasing new updates remains an open challenge in industry today (Ruhe and Greer 2003; Ruhe and Saliu 2005). Some applications, e.g., Mozilla, have moved to a rapid update cycle. However, such update cycles have led to many new software engineering challenges, e.g., the limited time available for software quality efforts for rapid updates. Other corporations had to reduce the frequency and inconsistency of their update schedule, e.g., Microsoft (2003) instead opting for consistent update frequencies, due to customer backlash about such rapid and ad-hoc update frequencies. Mobile development is a relatively new community compared to traditional software development and very little is known about the update patterns of mobile apps, e.g., as a mobile software developer is it normal to push an update daily or would your app be singled out? What do your peer developers often do? Recent studies show that frequently-updated apps could benefit, in terms of ranking, since some mobile stores factor in the freshness of an app (Lim and Bentley 2013; Lynch 2012). Maintaining a high ranking is essential for developers of mobile apps given the large number of apps in app stores. However, to the best of our knowledge, there exists no empirical studies regarding how often mobile apps are being updated. Such a study is of a great interest since the mobile stores provide a very convenient and automated process to facilitate frequent updates.

In this paper, we sought to shed some light on the update frequency of mobile apps and the frequency at which developers make use of such a convenient delivery platform for updates, e.g., how often do developers update their apps. We examined the update frequency of 10,713 mobile apps (the top 400 free[1] apps at the start of 2013 in each of the 30 categories in the Google Play store). We crawled the Google Play store over a 47 day time period at the beginning of 2014. We chose apps that had been popular for at least one year to ensure that the observed frequency of updates is more representative of a stable app, not one that is in its early lifetime (where frequent updates are more likely to be expected).

We structure our study around the following research questions:

RQ1: How frequently and consistently are apps updated?
~14 % of the studied apps are updated on a bi-weekly basis (or more frequently). With ~1 % of the apps updated at least once per week. Apps in the "Social" category in the Play store are the most frequently-updated. We also note that there is no

[1] We define a free app as an app that is free-to-download.

consistent update frequency – no consistency on which day of the week or update period, e.g., length of time between updates.

RQ2: What is the rationale for frequent updates?
Almost half (45 %) of the updates in the top 100 most frequently-updated apps do not contain any information about what has changed (e.g., silent updates). When the rationale for the update is communicated to users, the update is often due to bug-fixing (63 % of the time) – followed by new features (35 %) and improvements (30 %).

RQ3: What is actually changing for frequent updates?
The binaries (APK files) of silent updates exhibit a similar amount of churn as that of non-silent updates. The source code is the most modified part of the binaries. Overall, the binary files exhibit moderate growth – a median growth of 6 % across the most frequently-updated apps.

This paper is organized as follows: Section 2 presents background about the update process of apps in mobile stores. Section 3 presents the design of our study. Section 4 provides the results of our empirical study. Section 5 discusses our findings. Section 6 discusses the threats to the validity of our study. Section 7 presents the related work. Section 8 concludes our work.

2 Background

This section introduces the app stores, details the process by which developers and users interact with the app store and motivates developer interest in understanding the update frequency of mobile apps.

2.1 App Stores

The mobile app store continues to grow at a very rapid pace with thousands of developers, thousands of apps, and millions of dollars in revenue as of September 2013 (Gartner 2014). Mobile apps are available through app stores, as previously mentioned, the largest app stores are the Apple App Store, the Blackberry World Store, the Google Play Store and the Microsoft Phone Apps Store. However, there are many more specialized or regional app stores with thousands of apps. Such stores provide a convenient and efficient medium for users to download or purchase apps and to provide feedback to the developer on their experiences.

We briefly describe the two most popular app stores (the Google Play Store and the Apple App Store). The Google Play Store is a digital distribution outlet run by Google. Apart from apps, it sells other digital media e.g., e-books, movies and music. There are paid and free apps available in the stores. Paid apps must be purchased before use, while free apps are available to download free of charge. Apps can be downloaded and updated manually or automatically from the app store. The Apple App Store is Apple's digital distribution outlet for third-party apps and media. The apps range from games, productivity apps, social networking and business apps.

Each app in the app stores has its own homepage. The page contains meta-data about the app such as the title, developer, average rating, description, similar apps, the upload date, recent user reviews, screenshots, video previews and a link to download the app.

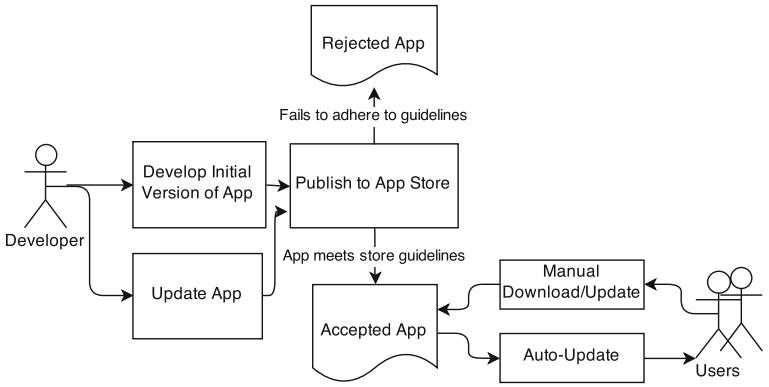


Fig. 1 The process that a developer follows to publish or update an app in the Google Play Store

2.2 Publishing an App

In order to publish an app on the Google Play Store, a developer must follow the guidelines and instructions of the app store as Fig. 1 demonstrates. The developer must ensure that their app meets the usability and reliability standards of the app store. Once the developer is confident that the app meets the quality standards and respects the content guidelines of the app store, the developer may submit it to the store. Apple requires a $99 developer fee per year, whereas Google requires a one time $25 developer fee (Viswanathan 2014).

Once an app is submitted, each store will review the app submitted to it. The Apple review process involves a manual review of the app, whereas the Google review process is automated. The Apple process can take several days or even weeks and it is not uncommon for apps to be rejected. In the case of the Google Play Store, the app is verified with automated tests for malware and within a few hours the app is available on the store.

In the Google Play Store, an app is stored as an APK file. In the case of the Apple App Store, an app is stored as an IPA file. The binary file, e.g., APK, contains the app's entire content including code, art assets, and configuration files.

Once an app is published, a developer can update it as much as he or she wishes. In Google's case, a new update is available immediately and users are notified that there is a new update to download. For Apple, the update undergoes a review by Apple (Apple 2014). Users also have the option to enable automated updates for an app (App updates are automatically installed whenever available). There is an optional *update note*-like section on an app's store homepage called "What's New" where the developer can provide a brief message about what has changed in the new update. Figure 2 shows an example of a typical text for the "What's New" section.

2.3 Mobile App Analytics

There are many app analytics companies that specialize in providing tools or reports to developers to understand how users interact with their apps, how developers generate revenue (in-app purchases, e-commerce, direct buy) and the demographics of their app users (Flurry 2014; App Annie 2014; Distimo 2013; Adobe 2014). These app analytics companies also provide developers with overviews of user feedback and logging crash reports.

```
What's New
Version 3.2.1
-Immersive mode is now supported on KitKat devices, so
you can use your whole screen to read
-New, nicer-looking thumbnails
-Removed un-needed permissions
-Other bug fixes and improvements
Version 3.2.0
-Added more font size options so you can get just the
 right size for you
-You can now like a book on Scribd - just tap the share
 icon while reading and pick "Like on Scribd"
-Applied another coat of polish to the app
```

Fig. 2 Example of the "What's New" section for the app 'Scribd'

Google has their own extensive analytics tools for Android developers. The tools provide various services, such as measuring how users are using a developer's app, locating where the users originate from and how they reached the app, tracking how the developer makes money through in-app purchases and calculating the impact of promotions on the sales of the app (Google 2014b). App analytics tools are prevalent in the industry and show demand for the tools. Vision Mobile performed a study based on a survey of 7,000 developers and found that over 40 % of developers make use of user analytics tools and 18 % use crashing reporting and bug tracking analytics tools (Vision mobile 2014). Previous studies also highlight that developers make use of app analytics tools. Pagano & Bruegge conducted a study on how feedback occurs after the initial release of a product (Pagano and Bruegge 2013). The authors concluded that there is a need to structure and analyze feedback, particularly when it occurs in large quantity.

3 Empirical Study Design

In this section, we present the design of our study, the data collection and the processing methods that are used in our study.

3.1 Study Goals

The *goals* of our empirical study are to analyze frequently-updated mobile apps, to better understand the rationale of frequent updates.

The *focus* of our empirical study is the frequency of updates of mobile apps, maintenance tasks and APK change. Our *perspective* is that of practitioners (e.g., stakeholders) and researchers interested in apps that are frequently being updated. The *object* of our case study is the different updates of the top 12,000 apps in the Google Play Store. In this empirical study, we address the three RQs that were introduced in Section 1.

3.1.1 Data Selection

We select the Google Play Store as our app store of interest. Our criteria for an app store is based on the popularity (the Google Play Store is one of the most popular app stores) and that tools exist to automatically collect information from the store. Both the Apple App Store and the Google Play Store are comparable so we focus on Google play and leave the Apple app store for future studies. We select a total of 12,000 free-to-download apps from the Google Play Store. We select the top 400 most popular apps in the USA

for each of the thirty different store categories e.g., Photography, Sports and Education. The popularity ranking is based on Distimo's ranking of apps. Distimo is an app analytics company which has been acquired by Distimo (2013). The list was selected from the most popular apps according to Distimo in the Spring of 2013. As we previously stated, we chose apps that were popular one year ago to avoid the expected frequent updates of a brand new app.

3.1.2 Data Collection

We use an open source Google Play Store crawler (Akdeniz 2013) to extract the apps' information and APK files. The crawler simulates a mobile device and interfaces with the Google Play API as a regular mobile device. We select the Samsung Galaxy S3 phone as our simulated device since it is one of the most popular Android devices (Chen 2014). We modified the crawler for our purposes. We only extract the relevant information for our study. We instituted a timer to pause the crawler to avoid sending too many requests to Google's servers and we scaled the crawler over multiple machines to distribute the load. The machines pool the results into a central database.

We ran the crawler on a daily basis over a period of 47 days beginning on January 1^{st} 2014 to February 16^{th} 2014. The crawler cycles over every one of the 12,000 apps in the span of 24 hours. In the first crawler run, the crawler downloads the static information, such as the app title, author, and number of downloads. All the downloaded data is stored locally in a database. For each app, the crawler downloads the latest user reviews and app permissions (e.g., access to contacts, and ability to send notifications). By latest, we mean recent reviews that are accessible since the last crawl.

Then, the crawler checks if there is a new update, if not, it moves to the next app. If there is a new update (e.g., the current update date is greater than the update date stored in the database) then the APK file of the new update is downloaded and the relevant information of the new update is downloaded and stored in the database. The information for a new update includes the size of the app, version number, update date, description of the app, installation size, description of recent changes to the app, and the current star ratings (one through five) given to the app. 11 % (1,287) of the 12,000 apps could not be accessed by the crawler and so the crawler failed to extract the information on these apps. These apps no longer exist in the store and return a "Not Found" page result if one attempts to view the app's homepage. These apps ceased to exist between the time that we selected the app list and when we began to crawl the app store. In total, 10,713 apps were crawled regularly as Table 1 shows.

Table 1 Number of versions per apps as of Feb 16, 2014

Total crawled apps	12,000
Inaccessible	1,287
Total accessible apps	10,713
Not updated during the study period	7,044 (66 %)
Updated at least once	3,669 (34 %)
Updated bi-weekly or more frequently	1448 (14 %)
Updated weekly or more frequently	98 (0.9 %)

3.1.3 Data Processing

We observe in our data that the update date will sometimes change but the version number and APK file remains the same. We therefore, use the version number to track unique updates for each app in our analysis.

The reason for this irregularity is due to a feature in the Google Play Store which allows developers to associate multiple APK files for different models of mobile devices. Developers are able to manage their apps' configuration and distribution to target specific users and phones. A developer can specify the features, e.g., OpenGL support, screen sizes that must be present on a mobile device etc., for a mobile device to be compatible. The Google Play Store will then intelligently select an APK file that best suits the user's mobile device. Therefore, the update date is somewhat misleading as it coincides with any new update by the developer for any mobile device's model.

In the following sub-sections, we present the results of our empirical study.

4 Empirical Study Results

This section presents and discusses the results of our three research questions. For each research question, we present our motivation to study the question, the approach that we used to answer the question, and the results of our analysis.

RQ1: How frequently and consistently are apps updated?

4.1 Motivation

The frequency of releasing new updates remains an open challenge in industry today (Ruhe and Greer 2003; Ruhe and Saliu 2005). Mobile development is a relatively new community compared to traditional software development and very little is known about mobile apps update planning, e.g., as a mobile software developer is it normal to push an update daily or would your app be singled out? What do your peer developers often do? Recent studies show that frequently-updated apps could benefit, in terms of ranking, since some mobile stores factor in the freshness of an app (Lim and Bentley 2013; Lynch 2012).

4.2 Approach

We calculate the frequency of new updates by summing the number of crawled new versions of each app. We then separate the app into three groups based on their update frequency: less than bi-weekly, bi-weekly to weekly and weekly or more frequently. To further investigate the update frequency, we examine the frequency of updates across the various categories, e.g., Arcade, or Social, of the store.

To investigate the consistency of updates, we observe the variation of update periods with respect to the mean update periods of an app. Specifically, we plot the relationship of the logged standard deviation of update periods of an app with the mean update period of the app. We use the logged standard deviation because of the high skew present in the data.

We continue our investigation of the update consistency by examining the consistency of days of the week. We also calculate the mode day of the week for each app.

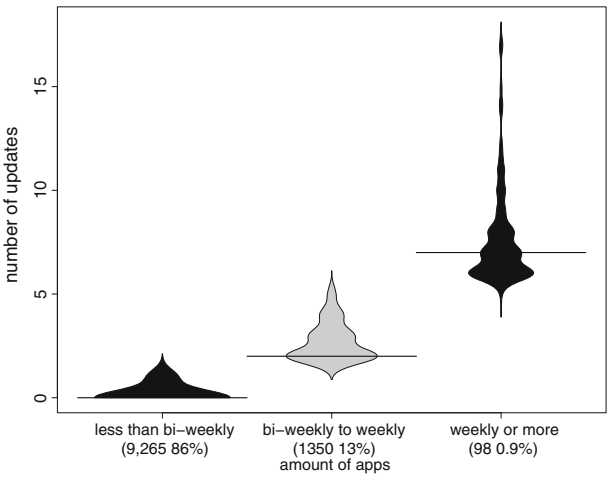


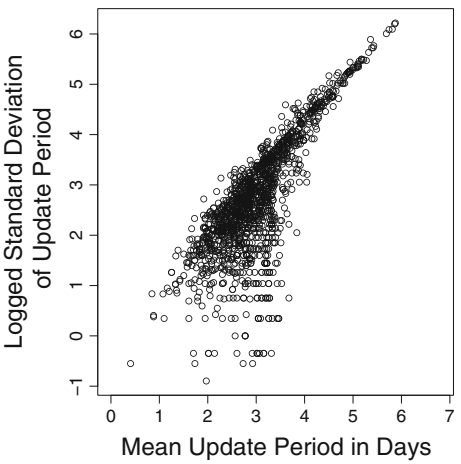
Fig. 3 Apps divided into three groups: updated less than bi-weekly, updated bi-weekly to weekly and updated weekly or more frequently on average. We crawled for over six weeks

4.3 Results

˜1 % of the apps are updated at least once a week Approximately 1 % of the studied apps are updated weekly or more frequently as Table 1 shows. The frequency of updates are quite high considering that these apps are at least one year old. A sizable subset (˜14 % of the studied apps) are updated bi-weekly or more frequently as Figs. 3 and 4 shows. 34 % of the apps are updated at least once during our study period, while the remaining apps (66 %) were not updated. The aforementioned percentages are calculated based on the apps that we successfully crawled. The update frequency of the frequently-updating subset is markedly different from studies performed on traditional software (Otte et al. 2008).

We further investigate the update frequency across the different categories. We calculate the number of times each category occurs in the bi-weekly or more frequently-updated apps.

Fig. 4 The relationship of the logged standard deviation of an app over the mean update period (in days) of the app



We divide the amount by the total number of successfully crawled apps in that category. The most frequently-updated Google play category is 'social' followed by 'productivity' as Fig. 5 shows. Some categories, e.g.,medical apps, have a much slower update frequency.

Frequently-updated apps are updated on a consistent period whereas infrequently-updated apps do not follow a consistent update period We investigate this subset of apps further by examining if the update frequency plays a role in the consistency of an update period. We find a linear relationship where frequently-updated apps (e.g., lower mean update period) have a lower standard deviation for their update period, whereas, infrequently-updated apps (e.g., apps with higher mean update periods) have a higher standard deviation for their update period (e.g., inconsistent period) as Fig. 4 demonstrates.

Updates do not occur on consistent days of the week By observing the mode day of the week for update dates per app, we find that most apps do not often update on weekends as Fig. 6 shows. The updates occur evenly throughout the week. The results demonstrates that there is no favored day of the week for app developers to update their app. Most likely, developers prefer releasing an update when it is ready as opposed to a scheduled release day (in contrast to e.g., Microsoft's patch Tuesday (Microsoft 2003)).

~1% of apps update at least once per week. We also note that there is no consistent frequency of updates for a particular day of the week or for apps with long update periods.

RQ2: What is the rationale for frequent updates?

4.4 Motivation

Developers use the "What's New" section of a mobile app homepage to communicate the rationale for a new update to the users. However, it is unknown how developers make use of the "What's New" section (e.g., what do developers communicate to the users for frequent-updates). With this knowledge, developers can better understand the tasks that are

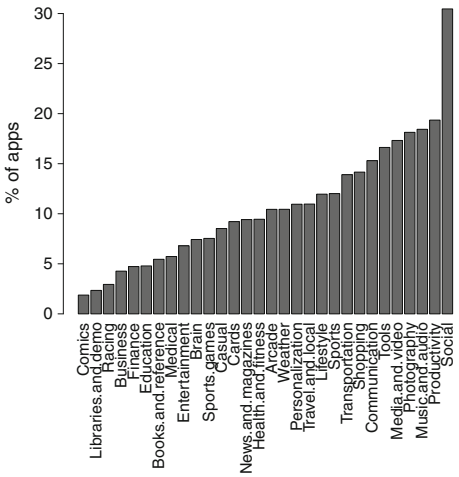


Fig. 5 Percentage of the studied apps which are updated bi-weekly or more frequently in each category of the store

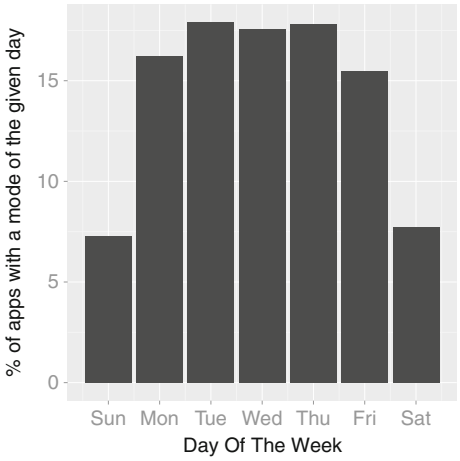


Fig. 6 Mode day of the week for each app's update date

often performed for new updates, just as developers access knowledge on the revenue and popularity of apps in the store (App Annie 2014).

4.5 Approach

We analyze the text in the publicly available "What's New" section (see Fig. 2 for an example). This optional section is authored by the developer and denotes what has changed in the new update. We choose to analyze the "What's New" sections for each new update of the top 100 most frequently-updated apps. Our choice is motivated by our focus on frequently-updated apps.

We manually label each of the "What's New" sections. In total, we manually labelled 852 different updates and their corresponding "What's New" sections for the top 100 most frequently-updated apps. We discover five common tasks that are performed in the updates (see Table 2). The five tasks are (i) adding new content, (ii) fixing bugs, (iii) new features,

Table 2 Types of tasks performed in the top 100 most frequently-updated apps

Task	Description
New content	New content is the addition of information separate from the core features of an app. For example, a new video game character, or chat icons for users, or a new map for a game.
New feature	New features encompass any new addition to the core functionality of an app.
Improvement	Improvements are tweaks, performance, memory improvements or small improvements to existing features.
Bug fix	Reported fixes of bugs, defects, crashing or unexpected events.
Permission changes	A change in permissions reported by the developer.

(iv) implementing improvements and (v) adjusting permissions. We record the amount of the same task that occurs in the text, whenever possible. For example if the "What's New" section says "added a new like button in the menu and added the ability to scroll", then we record two new features for that update. We do not count how many improvements or bug fixes occur in the same "What's New" section because developers often state an unknown amount of bug-fixes or improvements e.g., "bug fixes and some improvements".

From our manual labelling, we calculate the co-occurrence of the tasks with one another. For example, we calculate how many times new feature and bug fixing tasks co-occurred together for an update. This information gives us an idea about the amount of coupling between the different tasks.

To complement our manual analysis, we perform an automated analysis of the "What's New" section. We calculate the size of the text and the change in size between successive updates. The automated analysis is done to study the size of the developer communication, and whether developers simply append to or update the "What's New" text or replace it in its entirety.

4.6 Results:

Almost half (45 %) of the updates are silent e.g., do not contain any information about what has changed Silent updates are where the "What's New" section is either left empty or is identical to the "What's New" section from the previous update. We designate the apps that only release silent updates as "silent apps". In fact, the most frequently-updated app, "3D Cruz Azul Fondo Animado" is completely silent. Further, there are semi-silent apps that contain some silent updates and some non-silent updates.

Figure 7 shows the distribution of the percentage of silent updates in the 100 most frequently-updated apps. The x-axis denotes the percentage of silent updates for an app. The y-axis denotes the percentage of apps with a specific percentage of silent updates.

Silent updates leave users completely in the dark as to what has changed and users may not download the new update especially if they are bandwidth-limited. Further, two developers actually apologized in the"What's New" section for frequent updates. Evidently the frequent updates upset their users because they were asked to update their app so frequently. A weather app "InstaWeather" in version 64 had added several small features in each update

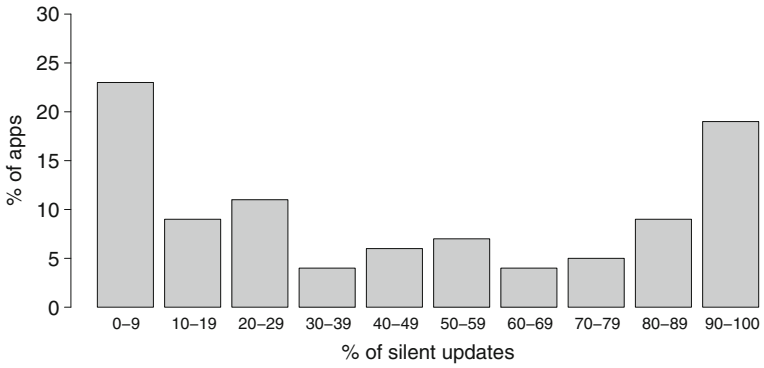


Fig. 7 Distribution of the percentage of silent updates in the updates of the 100 most frequently-updated apps

along with bug fixes and apologized for the many updates that week. We next discuss the tasks mentioned in the "What's New" section of the non-silent app updates.

The most frequently communicated task is bug fixing which occurs in 63 % of the non-silent updates As Table 3 shows, the rest of the non-silent updates contain 35 % new features, 30 % improvements, 25 % new content and very few permission changes (1 %).

The style that developers use to communicate tasks varies. For some bugs, developers do not explicitly state the amount of bugs fixed, instead they simply state "fixed bugs" or "fixed crashes". However, in other cases, developers might communicate details of a specific bug fix. For example, version 44 of the fitness app "Sports Tracker", the developer states explicitly "fix losing edited workout details". We found that a bug fix's details are provided in the "What's New" section in 51 % of the cases when an update indicates that a bug fix has occured.

However, developers explicitly mention, in the "What's New" section of an app, if they add new features, new contents and new permissions to their app(s). New features and content are tasks that developers want to advertise to the users so it is understandable that they would feature them prominently in the "What's New" section.

New permissions were mentioned in only three apps. The first app is a tool to manage invoices and manage your expenses called "Zoho Invoice and Time Tracking". In version 33, "Zoho Invoice and Time Tracking" required the ability to read the user's phone state to know when a customer of the user calls. The second app was a communication app called "Open Garden" that wanted permission in version 259 for sending text messages and reading users' contact list. In each case "Open Garden" mentioned explicitly the reason for adding a new permission. The third app "Ultimate custom widget" had accidentally removed a permission and added the permission back in the subsequent update.

We further investigate if the frequently-updated apps are adding permissions silently. We collected the permission data of each new version as part of our crawling. For each of the frequent updated apps we determine if a new version has added or removed permissions.

Permissions are changed silently We observe that 31 % of the apps are silently changing permissions at least once. This contrasts with only 3 apps which reported permission changes in their "What's New" section. The changes to permissions are split evenly between additional permissions and removal of permissions with 46 % added and 54 % removed in total. An example of an app which added permissions silently is the app "Customizable Countdown Widget" which added the ability to read and write to external storage. Their "What's New" section said: "3.1.2 Fixed a few issues with uploading images. It should be fixed. Email me if it is not". The absence of notice about a change in permissions shows a clear lack of communication. Users still must agree to permission changes when a new permission is requested so the permission changes are not hidden from the users.

Table 3 The frequency of non-silent updates for a given task

Task	Frequency
New content	114 (25 %)
New feature	158 (35 %)
Improvement	136 (30 %)
Bug fix	284 (63 %)
Permission	5 (1 %)

Yet, the developers choose not to present a reason behind the permission changes. The lack of explanation may trouble some users as a new permission may allow access to sensitive information or alter a phone's settings. Users may wish to understand exactly how the new permission will be used before they grant the permission to a specific app. For example, many apps who request access to a user's GPS location do not require such permission to function correctly, yet such a permission is needed by the ad libraries that are integrated into these apps.

We now examine the co-occurrence of tasks in an update.

Tasks do not occur in isolation Tasks often co-occur together and new features and new content occur more than once in the same "What's New" section. 47 % of the studied "What's New" sections contained more than one task. Figure 8 shows that bug fixes, new features and improvements co-occur in about 10 % of tasks. A developer may perform multiple tasks, e.g., adding a new feature or fixing a bug, in a update. Thus, there is an overlap among different tasks. Consequently, the sum of percentages of nodes in Fig. 8 is over 100 %. The mix of tasks suggests that developers are in a continuous flux of development and maintenance. If we consider how often a task occurs in the "What's New" section, we find that new features have the highest occurrence in the same "What's New" section. The highest occurrence of new features was 7 in the "Solo Launcher for Your Theme" social app where the developers implemented a redesign of the app. Finally, we end the discussion of the results of RQ2 by presenting the results of our automated analysis.

Developers follow two styles in the "What's New" section We observe that developers tend to organize their "What's New" section along two styles. The first style is to replace the text in the "What's New" section completely across updates. The replaced text can either be larger or shorter depending on the content of the new update. The second style maintains

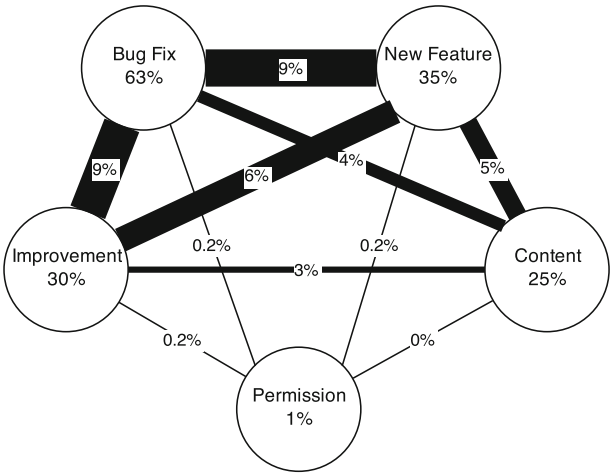


Fig. 8 Amount of task coupling per update. An edge between two tasks means that they co-occur in the "What's New" section. The number depicted on an edge is the percentage of updates where both of these tasks co-occur. A thicker edge denotes a larger percentage. The number in a task circle indicates the percentage of updates where such a task occurred in the "What's New" section

the content from previous "What's New" sections then appends new information relevant to the current update. The median change is quite small because in both cases the text is not increasing or decreasing sizeably. In total, the text size is 300-400 characters long, relatively short and easily digestible.

45% of frequently-updated apps do not inform the users what has changed including 28% of apps that modified their permissions silently. When apps do inform the user, the update is a mix of development and maintenance: 63% bug-fixes, 35% new features, 30% improvements and 25% new content.

In the following section we focus on what actually changes in the code as opposed to what is reported by developers.

RQ3: What is actually changing for frequent updates?

4.6.1 Motivation

RQ2 shows that non-silent frequently-updated apps inform the users that they are continually developing and maintaining their apps. However, we wish to complement our analysis of the developer's communications by analyzing the changes in the APK files. We perform this analysis for two reasons. The first reason is that we do not know what is actually changing inside the APK files or by how much regardless of what the developer communicates. Secondly, as we have seen, there are apps that do not communicate any information regarding a new update. We investigate the amount of change of the top 100 frequently-updated silent and non-silent apps.

4.6.2 Approach

To investigate how apps are changing over multiple updates, what is occurring inside the apps and the differences between silent and non-silent apps, we analyze the binaries of the apps (e.g., the APK files). We select the contents of the APK files for each of the top 100 most frequently-updated apps whose "What's New" sections we had manually labelled in RQ2. For each app and its updates, we extract the associated APK files. In total, we analyze 852 updates. The total number of silent apps are 17 and the total number of non-silent apps is 83.

We use a tool called APK Tool (Brut.alll and Connor Tumbleson 2014) to extract the content of the APK files. The output of the tool are the resources, assets and files used by the app. The java code is not available as the tool does not decompile the source code. The java code remains in files ending with the extension .smali which is the Dalvik bytecode language (Google 2014a). Dalvik is the Android virtual machine that is used to run the apps.

We analyze the differences of each version through time starting from version 1 to version n. We are interested in observing the changes between APK size and the frequency of churn (i.e. code change) of the various file types e.g., .smali, .xml. To simplify the presentation of our analysis, we group similar file names together e.g., .smali, .lua, .js are grouped as "code" and .png, .jpeg .gif are grouped as "images". Table 4 presents the file types and the groups we selected.

For each file type group in every update of every studied app, we calculate the size of the update, the percentage of code churn and the total change in size of the app's APK file over our data collection period which we define as growth rate as seen in (1). Once we calculate the amount of change for each file type in each update of an app, we calculate the

Table 4 The file type groupings, their descriptions and their extensions

File type group	Extensions
Code	smali,js,css,lua php
Markup langauges	xml, html, json, yml, grxml
Native code	.so
other	none, ccz, en, machinex, 30287,clientx 35536, 35536, cafe, g2g, svntmp, pkm, bks, LICENSE, pkcs12, atlas, acv, coords, armeabi, bc, 0, 3, 7, 8
images	png, jpg, gif, svg, tbn, xcf, raw, jpeg, ico, psd, PNG
font	ttf, otf, fnt, ttc, TTF
config	properties, plist, prop, config, ini
archive	zip, jar, apk
audio	wav, aac, smf, mp3, m4a, mp4
binary	ccbi, bin
multimedia	ogg, swf
graphics	vsh, fsh
data	dat, assets, csv
db	db, sql

median change across every update of that app to determine how each file type changes over time.

$$\text{growth rate} = \left(\frac{\text{current size} - \text{initial size}}{\text{initial size}}\right) \times 100 \tag{1}$$

4.6.3 Results

Silent apps grow as much as non-silent apps During the study period, the median growth rate of non-silent versus silent apps is very close at 6.6 % versus 4 % as Fig. 9 shows. This similar growth rate demonstrates that changes of similar magnitude are occurring in

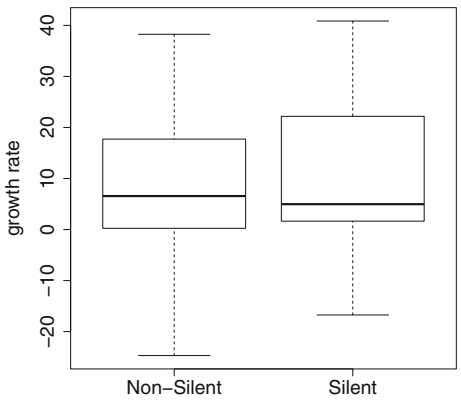


Fig. 9 The total APK file size growth of frequently-updated silent and non-silent apps over the study period

frequently-updated silent apps as they are in frequently-updated non-silent apps. The differences are not statistically significant with a Mann-Whitney test p-value > 0.05. The similar growths suggest that developers are likely performing similar maintenance tasks without communicating to the user what they are.

Negative changes do occur The APK file changes for non-silent apps is above zero, however 25 % of changes were negative as Fig. 9 demonstrates. We see the same percentage of negative changes among the non-silent apps as well. However, if we only consider the APK file changes, we do not gain insight into which types of files are changing. Therefore we explore below which file types are changing the most.

Source code changes the most Figure 10 presents the percentage of change of the different file types. It is clear that the source code is where most of the changes occur. The only other file group with a median percentage change that is greater than 0 is markup languages such as XML. One use of XML in Android apps is to represent the UI layout of apps. It is possible that changes to source code are mostly done because upgrading a version of a library (no changes to app-specific code), or because including a third party library inflates the number of classes in the APK (Linares-Vásquez et al. 2014). Thus, changes in the source code due to updating a third party library could impact our results. In addition, it is possible that a developer may add 200 lines of code and remove 200 lines of code. In this case, the difference between the source code file sizes might be 0. This may impact our results. All the apps are closed source code and we do not have access to actual source code. To generalize the results, a study is required on open source code apps to better understand source code changes in frequently-updated mobile apps. Additionally, it would be valuable to examine the tasks that are performed when changing the code such as modifying ad libraries, fixing a resource leak or changing a deprecated API.

Frequently-updated apps are growing moderately even though they are mature apps We find that the apps undergo a moderate increase of 6 % in size over our entire study period. Figure 9 shows the growth factor of apps over the period of our study. The

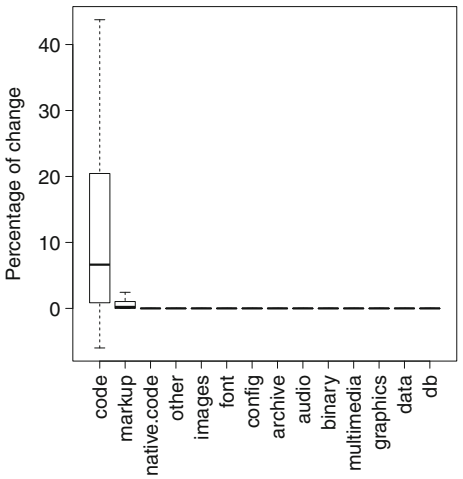


Fig. 10 The median change in file type groups of the frequently-updated apps

growth factor is the relative amount of growth from the start of our study till the end. The apps continue to grow even after at least one year of existence on the store.

Frequently-updated apps are continuing to undergo code churn and have moderate growth of their APK file. Silent apps undergo similar changes.

5 Discussion

We now discuss some of additional observations that we made during our experiments.

5.1 The Impact of Update Frequency on "App Ratings"

We divide our entire dataset of apps into two categories using the median update frequency: frequently-updated and infrequently-updated. We compare the perceived user quality of infrequently vs. frequently-updated apps. Our measure of perceived user quality is the percentage of "negative ratings". We chose the percentage of negative ratings because most apps in our dataset have an overall four star rating. The percentage of negative ratings allows us to see the amount of negative sentiment associated with the apps (Maas et al. 2011; Pang and Lee 2004). We calculate the percentage of "negative ratings" (1 or 2 star ratings) for each app. The percentage of negative ratings is simply the ratio of one-star and two-star ratings over the total amount of ratings.

We only use the negative ratings given to apps during the time of our study. The Google Play Store only updates the amount of 1, 2, 3, 4 and 5 stars of an app over its entire lifespan (this includes ratings given many years ago). However, we collect the amount of 1 and 2 stars after each new update which allows us to calculate the negative ratings of an update (Mojica Ruiz et al. 2014). To calculate the amount of 1 and 2 stars an update n receives, we subtract the amount of 1 and 2 stars when update $n + 1$ is released from when update n was released. We calculate the weighted average. We calculate the weighted average for every update and the median weighted average per app in which we are interested.

Frequently-updated apps have a lower percentage of negative ratings As Fig. 11 demonstrates, there are more negative ratings for infrequently-updated apps. The results are statistically significant with a Mann-Whitney test p-value < 0.05 and an effect size of 0.22 (small effect) (Kampenes et al. 2007). Additionally, we control for the size of apps since larger apps may take longer to update. We divide each frequent and infrequent app groups into two by the median size of the app's APK file. Figure 12 demonstrates that our observation still holds – frequently-updated large/small apps have lower percentages of negative ratings than their respective infrequently-updated large/small apps. The difference in ratings between frequent small and infrequent small apps as well as the difference in ratings between frequent large and infrequent large apps is statistically significant with a Mann-Whitney test p-value < 0.05. The comparison is conducted on popular apps drawn from Distimo's most popular apps. Therefore, our results may not generalize over less popular apps. Additionally, even a small difference in rating can effect the visibility of an app in the app store. As previous research shows, the ranking of apps in app stores are influenced by the rating of apps (Lim and Bentley 2013). A rank of an app is the position of an app in Google play store search results. Whereas, a rating is the star voting given to an app by users.

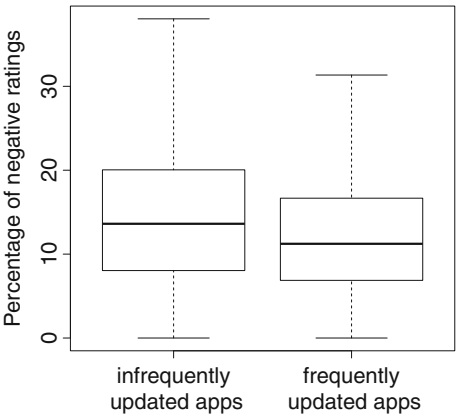


Fig. 11 Percentage of negative star ratings for infrequent and frequent updated apps.

The statistically significant results of our comparison between frequently-updated apps and infrequently-updated apps suggests at the very least that developers who choose to update frequently should not fear an increase in negative ratings.

5.2 The Impact of Update Frequency on App Popularity

We further investigate whether frequently-releasing new versions of an app leads to an app becoming not as highly popular in according to Distimo's ranking. The popularity is a judgment of success that includes overall downloads and ratings according to Distimo. We compare the 100 most frequently-updated apps against all other apps in our dataset. We verify if an app is still in the most popular apps of its category in Distimo for January 2014 (a total of approximately 10,000 apps). In the case of the 100 most frequently-updated apps, we manually verify that each app still appears in Distimo's popularity list. For the rest of the apps, we perform automated verification. We crawl Distimo's rankings and compare the

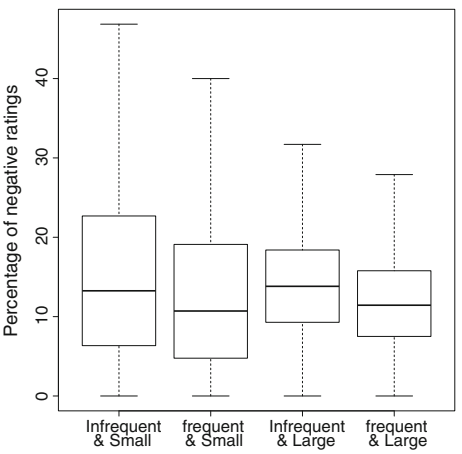


Fig. 12 Percentage of negative star ratings for infrequent large, infrequent small, frequent large and frequent small apps

list of apps that we collected in the Spring of 2013 with all the popularity lists of Distimo. As we crawl, there may be some false negatives because apps may have changed their names since our original crawl in Spring 2013. However, manual verification of approximately 10,000 apps would be prohibitively time-consuming therefore we use the popularity of infrequently-updated apps as an estimate only.

Frequently-updated apps remain popular We find that 81 % of apps in the frequently-updated apps are still found in the list of most popular apps of their category, whereas only 45 % of the rest of the apps (e.g., not frequently-updated apps) are in the list of the most popular apps of their category.

6 Threats to Validity

Some threats could potentially limit the validity of our results. We now discuss such threats and how we control or mitigate them.

6.1 Threats to Construct Validity

The first author manually labelled the maintenance tasks mentioned in "What's New" section. The labelling may have produced some false labels. In order to mitigate this threat, the second author independently verified the manually-labelled tasks. If there were disagreements between labels, both raters came to a consensus. The raters could not come to a consensus 2 % of the time – in such cases the third author was consulted to break ties.

Our study investigates various characteristics and frequently-updated apps. However, to truly understand the rationale behind an update and update frequency, we need to conduct developer interviews.

6.2 Threats to Internal Validity

The selection of the top 100 most frequently-updated apps may bias any conclusion that we may draw on apps in general. We therefore only draw conclusions on frequently-updated apps.

The APK tool that we used may not be perfect in disassembling all the APK files. We mitigate this threat by only focusing on the file extensions and not the reverse engineered code in the files.

Distimo may not be an accurate measure of popularity. Distimo uses a combination of ratings, downloads and their own proprietary method for determining popularity. However, there is no other comprehensive tool to determine popularity as Google does not publish these statistics. Thus, we use the top apps lists that is provided by Distimo for the 30 store categories, in order to to ensure that our selected apps are popular amongst users.

The relationship between update frequency and the ratings and popularity of an app may be influenced by other factors. We mitigate this threat by not implying causation but merely highlighting the existence of a correlation. Further indepth controlled studies are needed to explore whether such a causation might exist.

The crawler needs a device ID to crawl Google Play Store. Google Play Store uses this ID to show compatible apps to the Device. Thus, selection of Samsung Galaxy S3 device ID may have an impact on the results. However, we successfully crawled all the selected apps. In other words, all the selected apps were compatible to Samsung Galaxy S3.

We crawled the Google Play Store since early January of 2014. Early January is a holiday season for many in the Western world. Thus, some apps updates could be delayed. This may have an impact on the results. However, the majority of the crawl period occurs outside the holiday season.

6.3 Threats to External Validity

The selection of apps may not generalize to all mobile apps. To mitigate this threat, we selected apps from all the different categories of the store to be as representative as possible. There are a total of 30 categories from which we draw our data from.

The selection of the most popular apps could bias our results. The popular apps could have a different frequency of updates than less popular apps. The Google Play Store does not have a directory from which all app names are crawlable therefore the list of popular apps had to suffice. Our results may only generalize to popular mobile apps. We only study apps that are free. Paid apps may exhibit different frequency of updates in comparison to free apps.

In our study, we consider all developers as the same, whether they are part-time individuals or large corporates. The size of a team could also impact the frequency and consistency of updates. Thus, we cannot generalize the results for all of the companies. A more detailed study is required to explore the impact of the size of an organization on the frequency of app updates.

6.4 Threats to Conclusion Validity

The selection of statistical test could bias our results. We mitigate this threat by using non-parametric tests because our data has a non-normal distribution.

7 Related Work

To the best of our knowledge, our study is the first attempt to empirically study frequently-updated mobile apps. This section looks at the literature related to update cycles of traditional software development, followed by prior work related to mobile apps.

7.1 Traditional Software Updates

Due to the migration of open source projects to agile development, many projects moved to frequent updates (Khomh et al. 2012). Otte et al. (2008) performed a survey to analyze the frequency of software releases. The results show that 49.3 % updated at least once per a quarter, 32.7 % releases every 6 month and roughly 18 % have even longer intervals. Developers and researchers have developed new tools and techniques to enable continuous delivery (Humble and Farley 2010; Porter et al. 2006). Der Storm (2005) and Dolstra et al. (2004) developed a framework to automatically build and deploy code changes. Amazon, for example, deploys on average every 11.6 seconds (Jenkins 2011), achieving more than 1,079 deployments per hour. These studies do not cover mobile software updates. Our work is able to analyze the update frequency of thousands of developers on the same deployment platform (mobile app store) whereas the other studies feature unrelated deployment environments.

Some researchers have explored the impact of software releases on the development effort and code quality. Kuppuswami et al. showed that small incremental releases reduce the required development effort by 2.67 % (Kuppuswami et al. 2003). Marschal observed that short cycle of releases involves a steady flow of releases in order to control the number of reported bugs (Marschall 2007). Escrow reduced 70 % of the defects by reducing its update cycle to bi-weekly (Hodgetts and Phillips 2002). Khomh et al. showed that shorter update cycles in open source software did not lead to more bugs and bugs were fixed faster (Khomh et al. 2012). Our study differs from the previously mentioned studies by highlighting the effects of the user experience both in what developers communicate to users and how the users respond with feedback.

Some applications, e.g., Mozilla, have moved to a rapid update cycle. However, such an update cycle has led to many new software engineering, e.g., software quality, challenges. Other corporations had to reduce the frequency and inconsistency of their update schedule, e.g., Microsoft (2003), instead opting for consistent update frequencies, due to customer backlash about such rapid and ad-hoc update frequencies. The right frequency is a topic of debate amongst software practitioners. Our aim is to expand on existing work on traditional software releases in order to study update strategies of developers in the mobile development community.

7.2 Mobile Apps

Rapidly growing mobile app stores with thousands of apps have not only attracted the attention of both researchers and software engineers but also posed new challenges for them as well (Harman et al. 2012; Kim et al. 2011; Linares-Vásquez et al. 2013; Mudambi and Schuff 2010). Some of the recent studies have shown that traditional software repositories and software are different from mobile apps (Harman et al. 2012; Syer et al. 2013). Thus, there is a need for specific studies related to mobile apps (Syer et al. 2013). Some of the studies have been conducted by researchers to better understand the problems facing mobile app developers. Linares-Vásquez et al. explored the issues that mobile developers had on Stack Overflow (Linares-Vásquez et al. 2013). Syer et al. explored the development of micro-apps in Android and Blackberry apps and found Android apps relied heavily on the Android platform and had fewer files than the Blackberry apps (Syer et al. 2011). Ruiz et al. studied software reuse in the Google Play Store. They found a great deal of reuse which allowed for the rapid growth of the store (Mojica Ruiz et al. 2012). Linares-Vásquez et al. empirically analyzed how the stability and fault-proneness of APIs used by some free Android apps could impact the success of apps (Linares-Vásquez et al. 2013). Our study differs from the above studies as we aim to understand how the mobile deployment platform (i.e, app stores) affects the update frequency of apps.

Users of mobile apps can leave feedback, e.g., complaints, feature requests, and rate an app. The feedback helps developers to improve the quality of their apps. However, users' feedback is full of textual noise that make it harder for a developer to analyze important feedback. Iacob and Harrison (2013) built a rule-based automated tool to extract feature requests from user reviews of mobile apps – their approach identifies whether a user review contains a feature request or not. Galvis Carreño and Winbladh (2013) used opinion mining techniques and topic modelling to extract requirements from user reviews. Harman et al. mined the Blackberry app store to analyze downloads and ratings of apps and found correlation between customer rating and number of downloads. They did not find any correlation between price and rating (Harman et al. 2012). Khalid et al. manually analyzed and

categorized 1 and 2 star mobile app reviews (Khalid et al. 2014). They manually identified the different issues users made in mobile apps. Pagano and Maalej analyzed the content of users' feedback of both free and paid apps and its impact on the user community (Pagano and Maalej 2013). The authors observed that most of the reviews are provided after an update, with a quickly decreasing frequency of reviews over time.

To the best of our knowledge, no study has ever empirically analyzed mobile apps updates. Lim & Bentley analyzed different ranking algorithms for mobile stores and found that the frequency of new updates impacted the ranking of an app (Lim and Bentley 2013). Our work is different from the aforementioned studies due to our focus on the top frequently-updated mobile apps. The combination of our analysis of developer communications and the analysis of the APK files provides a detailed picture of the frequently-updated mobile apps and separates us from prior work.

8 Conclusion

As the first study of the update strategies of mobile apps, our paper provides valuable empirical information to support future research in this important and growing area. This paper provides actionable information regarding the update strategies employed by the top mobile apps, namely that:

Developers should not shy away from updating their apps very frequently. We find that frequently-updated apps do not experience an increase in negative ratings. Therefore, developers should not be too concerned about releasing frequently for fear of the negative impact on their ratings. Future studies are needed to better understand the willingness of users to continue to accept new updates without an explanation of the purpose or value to the user. However the frequency varies across store categories. Future research should explore the rationale for such variations, e.g., is it due to user expectations, domain evolution, or technical complexity that might hinder or enforce rapid updates. Moreover, this observation does not match prior observation for non-mobile apps (Adobe 2014; Khomh et al. 2012). Other studies are needed to explore this mismatch.

It appears that users highly rank frequently-updated apps instead of being annoyed about the high update frequency. Future research should carefully examine such correlation to determine if one can imply a casual relation or not, e.g., are frequently-updated apps of high quality? Are users pleased with frequently and continuously receiving new updates? Even if such updates appear to be mostly bug fixes or of unknown purpose.

References

Adobe (2014) Mobile analytics. http://goo.gl/Y4c1Pl
Akdeniz (2013) Google play crawler. http://goo.gl/UAKiap
App Annie (2014) Analytics. http://goo.gl/jDLVIg
Apple (2014) Viewing and changing your app?s status and availability. http://goo.gl/YL0Az7
Brut.alll, Connor Tumbleson (2014) Apk tool. http://goo.gl/d77er4
Chen BX (2014) Samsung galaxy phone is no. 1 for now. http://goo.gl/iqTkcf
Distimo (2013) Google play store, united states, top overall, free, week 35 2013
Dolstra, E, De Jonge M, Visser E (2004) Nix: A safe and policy-free system for software deployment. In: LISA, vol 4, pp 79–92
Flurry (2014) Flurry analytics. http://goo.gl/rUFUeR

Galvis Carreño LV, Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pp 582–591. IEEE Press, Piscataway

Gartner (2014) Gartner says mobile app stores will see annual downloads reach 102 billion in 2013. http://goo.gl/IYdFtQ

Google (2014) Bytecode for the dalvik vm. http://goo.gl/5TKib9

Google (2014) Google analytics. http://goo.gl/3sm2lR

Harman M, Jia Y, Zhang Y (2012) App store mining and analysis: MSR for app stores. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp 108–111. IEEE

Hodgetts P, Phillips D (2002) Extreme adoption experiences of a B2B start-up. *Extreme Programming Perspectives*, pp 355–362

Humble J, Farley D (2010) Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education

Iacob C, Harrison R (2013) Retrieving and analyzing mobile apps feature requests from online reviews. In: Proceedings of the Tenth International Workshop on Mining Software Repositories, pp 41–44. IEEE Press

Jenkins J (2011) Velocity culture (the unmet challenge in ops). In: Presentation at O'Reilly Velocity Conference

Kampenes VB, Dybå T, Hannay JE, Sjøberg DIK (2007) A systematic review of effect size in software engineering experiments. Inf Softw Technol 49(11):1073–1086

Khalid H, Shihab E, Nagappan M, Hassan AE (2014) What do mobile app users complain about? A study on free iOS apps. In: IEEE Software. IEEE Press

Khomh F, Dhaliwal T, Zou Y, Adams B (2012) Do faster releases improve software quality? an empirical case study of mozilla firefox. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp 179–188

Kim H-W, Lee HL, Son JE (2011) An exploratory study on the determinants of smartphone app purchase. In: The 11th International DSI and the 16th APDSI Joint Meeting, Taipei, Taiwan

Kuppuswami S, Vivekanandan K, Ramaswamy P, Rodrigues P (2003) The effects of individual XP practices on software development effort. ACM SIGSOFT Software Engineering Notes 28(6):6–6

Lim SL, Bentley PJ (2013) Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp 2672–2679

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

Linares-Vásquez M, Dit B, Poshyvanyk D (2013) An exploratory analysis of mobile development issues using stack overflow. In: Proceedings of the Tenth International Workshop on Mining Software Repositories, pp 93–96. IEEE Press

Linares-Vásquez M, Holtzhauer A, Bernal-Cárdenas C, Poshyvanyk D (2014) Revisiting android reuse studies in the context of code obfuscation and library usages. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp 242–251. ACM

Lynch J (2012) App store optimization: 8 tips for higher rankings. http://goo.gl/htvSNL

Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C (2011) Learning word vectors for sentiment analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pp 142–150. Association for Computational Linguistics

Marschall M (2007) Transforming a six month release cycle to continuous flow. In: Agile Conference (AGILE), 2007, pp 395–400. IEEE

Microsoft (2003) Understanding patch and update management: Microsoft?s software update strategy. http://goo.gl/geZXp5

Vision mobile (2014) Developer Economics Q1 2014: State of the Developer Nation. Technical report, 05

Mojica Ruiz I, Nagappan M, Adams B, Berger T, Dienst S, Hassan A (2014) On the relationship between the number of ad libraries in an android app and its rating

Mudambi SM, Schuff D (2010) What makes a helpful online review? a study of customer reviews on amazon.com. MIS Q 34(1):185–200

Otte T, Moreton R, Knoell HD (2008) Applied quality assurance methods under the open source development model. In: 32nd Annual IEEE International Computer Software and Applications, p, 1247–1252. IEEE

Pagano D, Bruegge B (2013) User involvement in software evolution practice: a case study. In: Proceedings of the 2013 International Conference on Software Engineering, pp 953–962. IEEE Press

Pagano D, Maalej W (2013). In: Proceedings of the 21st. IEEE International Requirements Engineering Conference. IEEE

Pang B, Lee L (2004) A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In: Proceedings of the 42nd annual meeting on Association for Computational Linguistics, p 271. Association for Computational Linguistics

Porter A, Yilmaz C, Memon AM, Krishna AS, Schmidt DC, Gokhale A (2006) Techniques and processes for improving the quality and performance of open-source software. Software Process: Improvement and Practice 11(2):163–176

Ruhe G, Greer D (2003) Quantitative studies in software release planning under risk and resource constraints. In: Proceedings of the 2003 International Symposium on Empirical Software Engineering, ISESE 2003, pp 262–270. IEEE

Ruhe G, Saliu MO (2005) The art and science of software release planning. IEEE Softw 22(6):47–53

Mojica Ruiz IJ, Nagappan M, Adams B, Hassan AE (2012) Understanding reuse in the android market. In: IEEE International Conference on Program Comprehension (ICPC), page To appear

Syer MD, Adams B, Zou Y, Hassan AE (2011) Exploring the development of micro-apps: A case study on the blackberry and android platforms. In: Proceedings of the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, SCAM '11, pp 55–64

Syer MD, Nagappan M, Adams B, Hassan AE (2013) Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open source android apps. In: Proceedings of the IBM CASCON Conf.(CASCON'13)

Der Storm TV (2005) Continuous release and upgrade of component-based software. In: Proceedings of the 12th international workshop on Software configuration management, pp 43–57. ACM

Viswanathan P (2014) Android OS vs. apple iOS which is better for developers? http://goo.gl/ApQCb6

Stuart McIlroy received his Master's degree at Queen's University. His thesis topic investigated the unique aspects of mobile app stores and ways to leverage their unique distributon and feedback mechanisms. His research interests include data mining, artificial intelligence and large-scale softare engineering.

Nasir Ali is a Postdoctoral fellow at the Department of Electrical and Computer Engineering of University of Waterloo, working with Prof. Lin Tan. He has received his Ph.D. at Ecole polytechnique de Montreal under the supervision of Prof. Yann-Gaël Guéhéneuc and Prof. Giuliano Antoniol. The primary focus his Ph.D. thesis was to develop tools and techniques to improve the quality of software artifacts' traceability. He received a M.Sc. in computer science and an M.B.A. from the University of Lahore and National College of Business Administration & Economics, respectively. He has more than six years of industrial experience. His research interests include software maintenance and evolution, system comprehension, and empirical software engineering.

Ahmed E. Hassan is a Canada Research Chair in Software Analytics and the NSERC/Blackberry Industrial Research Chair at the School of Computing in Queen's University. Dr. Hassan serves on the editorial board of the IEEE Transactions on Software Engineering, the Journal of Empirical Software Engineering, and PeerJ Computer Science. He spearheaded the organization and creation of the Mining Software Repositories (MSR) conference and its research community. Early tools and techniques developed by Dr. Hassan's team are already integrated into products used by millions of users worldwide. Dr. Hassan industrial experience includes helping architect the Blackberry wireless platform, and working for IBM Research at the Almaden Research Lab and the Computer Research Lab at Nortel Networks. Dr. Hassan is the named inventor of patents at several jurisdictions around the world including the United States, Europe, India, Canada, and Japan. More information at: http://sail.cs.queensu.ca/.