The Influence of App Churn on App Success and StackOverflow Discussions

Latifa Guerrouj, Shams Azad, Peter C. Rigby
Department of Software Engineering, Concordia University, Montréal, Canada
E-mail: {l_guerro,sh_aza}@encs.concordia.ca, peter.rigby@concordia.ca

***Abstract*—Gauging the success of software systems has been difficult in the past as there was no uniform measure. With mobile Application (App) Stores, users rate each App according to a common rating scheme. In this paper, we study the impact of App churn on the App success through the analysis of 154 free Android Apps that have a total of 1.2k releases. We provide a novel technique to extract Android API elements used by Apps that developers change between releases. We find that high App churn leads to lower user ratings. For example, we find that on average, per release, poorly rated Apps change 140 methods compared to the 82 methods changed by positively rated Apps. Our findings suggest that developers should not release new features at the expense of churn and user ratings.**

We also investigate the link between how frequently API classes and methods are changed by App developers relative to the amount of discussion of these code elements on StackOverflow. Our findings indicate that classes and methods that are changed frequently by App developers are in more posts on StackOverflow. We add to the growing consensus that StackOverflow keeps up with the documentation needs of practitioners.

***Keywords*-Android, API elements, Changes, StackOverflow.**

I. Introduction

Google's Play Store has more than 1 million Android Applications (Apps). This success has lead to strong competition among similar Apps. For example, there are more than 80k 'Education' Apps for Android.[1] Although 'Education' is a broad category, from learning a new language to learning how to draw, a search for 'learn Spanish' provides a list of hundreds of Apps. With this level of competition, the empirical software engineering community can provide practitioners with the factors that affect App success. To this end, we address two related research questions. First, does App churn affect App success? Second, is there more community discussion about the API elements that App developers frequently change?

The success of a software system has been difficult to measure as there was not a single site and uniform ranking scheme [1]. In contrast, App stores allow users to rank Apps between one and five, with five indicating a good application. Users are also able to leave comments. To understand why users give poor ratings to Apps, Khalid *et al.* [2] qualitatively coded 6k user comments on the iOS App Store. They found 12 common reasons for low ratings including, functional error, App crash, and privacy issues.

Linares-Vasqueza *et al.* [3] studied how the changes of Android API used by mobile apps impacts the success of individual Apps. They found that the Apps that depend upon API classes and methods that were change and error prone were less successful than the Apps that used more stable and robust API elements. From this study, it should be possible to provide information to developers about the robustness of an API element.

Instead of studying the changes at the level of Android API, we study App churn. We ask, *Does App churn affect App success?* We do not have the version history for the Apps we study. Instead we compare the packages, classes, and methods that are added or removed between two consecutive releases of an App. We find that the more an App has high churn the lower its rating. Counter to the release engineering trend of faster releases of new features [4], our findings suggest that practitioners should carefully consider the tradeoff of adding new features with the potential of receiving low ratings from an unstable release.

Our second research question deals with App churn and StackOverflow posts. We ask, *is there more community discussion about the API elements that App developers frequently change?* Work by Kavaler *et al.* [5] found that larger classes have more questions on StackOverflow indicating a possible relationship between class complexity and number of questions. There is also a relationship between the number of Apps that use a class and the number of StackOverflow posts. In contrast to their work, we examine not only classes but methods. Instead of examining a single release, we examine the the API elements that App developers change between releases. We find that API elements that App developers change frequently have more discussion on StackOverflow.

We triangulate our findings by qualitatively analyzing negative review comments based on a semi-automatic analysis of frequent n-grams from such negative comments plus a manual investigation of the relation between App churn and the API classes and methods that are discussed on StackOverflow.

A. Research Questions

We study the following concrete research questions:

[1]http://www.Appbrain.com/stats/android-market-App-categories

- **RQ1:** *Does App churn affect App success?*
  We want to understand if frequent changes between releases in the API elements that an App uses affects its success. We measure success as the average user rating of an App and churn as the number of API elements that change between releases of an App. We conjecture that Apps that have high churn will have lower ratings. Specifically, we test the following null hypothesis:
  - H_{01}: *There is no significant difference between the number of changes to API elements used by successful and unsuccessful Apps.*

  We triangulate our findings by qualitatively analyzing negative review comments based on a semi-automatic analysis of frequent n-grams from such negative comments.
- **RQ2:** *Is there more community discussion about the API elements that App developers frequently change?* This research question aims at investigating if API elements (*i.e.*, classes and methods) that change frequently trigger more discussions by developers in the StackOverflow. We measure the number of Apps that change an API element and the number of StackOverflow posts that contain the same element. Specifically, we test the following null hypothesis:
  - H_{02}: *There is no significant difference between the number of StackOverflow discussions for API elements that heavily change and API elements that change less frequently*.

  We provide a preliminary qualitative evaluation of some discussions on StackOverflow that relate to frequently changed API elements used by Android Apps.

The remainder of this paper is structured as follows. Section II describes our empirical study design and the data we collected. In Section III we present our quantitative findings and triangulate them with a qualitative analysis. In Section IV we discuss threats to validity. Section V positions our work in the literature. Section VI concludes our work and highlights future work.

II. Empirical Study Design

The *goal* of this study is to understand how App churn influences i) the App success and ii) the number of posts on StackOverflow related to Android development. Success is measured as in terms of ratings posted by the Apps users in the Android market.

The *context* consists of 154 free Apps from the Google Play Store and 1.2K of releases of theses Apps. Table I presents the characteristics of the Apps we study: (i) the different categories (*e.g.*, business, finance, sports, etc.), (ii) the number of Apps releases for each category, (iii) the number of packages, (iv) the number of classes, and (iv) the number of methods that fall into each category. We study Android Apps, because user ratings provide a uniform measure of App success and because Android has become an important software platform. As of July 2013, Android had more Apps than Apple's iPhone App Store. The Android Play Store has over 1 million Apps published and 50 billion downloads[2]. In addition, Android has become the dominate mobile development platform with 71% of practitioners developing for Android[3]. Not only are there many Android Apps, the Android platform evolves quickly and has had 17 major releases over four years [3]. This popularity makes Android interesting for study and analysis.

TABLE I
CHARACTERISTICS OF THE STUDIED ANDROID APPS.

Category	Apps	Releases	Packages	Classes	Methods
Arcade & Action	23	60	4846	73841	112101
Books & Reference	3	9	808	15204	23109
Brain & Puzzle	11	26	3132	48706	72232
Business	5	11	519	7231	10696
Cards & Casino	7	18	2747	41499	61229
Casual	9	21	3032	44662	64581
Communication	10	31	3611	60524	92380
Entertainment	3	6	1713	32248	47725
Finance	5	10	691	10830	14869
Health & Fitness	4	16	525	9622	13675
Lifestyle	7	15	2054	27870	38838
Media & Video	4	12	1150	17014	23968
Music & Audio	5	15	2651	44802	67928
News & Magazines	4	11	710	11374	18285
Photography	4	19	2992	53552	77935
Productivity	11	43	2386	34632	54061
Racing	4	11	916	14640	22128
Shopping	3	7	667	12598	18420
Social	8	30	3974	73632	108756
Sports	5	12	1109	22320	32721
Weather	3	8	2898	40632	60094

A. Data Extraction Process

We use the following data to address our research questions (i) 154 Android Apps and their 1.2K releases, (ii) the list of API elements used by the Apps including packages, classes, and methods, (iii) the changes to these API elements between each release of the App, (iv) the user ratings of the Apps, and (v) the API elements occurring in the Android StackOverflow posts. In this study we consider only apps having at least two releases which is the minimum to use in such a kind of investigation. To get the different releases of each app, we designed a crawler that we used to crawl the Google play store each two days to get newer versions of an individual app during a period of almost five consecutive months. The categories as well as the user ratings and votes were also downloaded from the Google App Store by storing the votes posted for each App's release. Thus, for each app, we had its releases, category, the rating of each release as well as its corresponding votes. We also extracted the comments posted by each user corresponding to each app release. The Google play store gives access to only the 10 recent comments for an app using the Marketplace API. We wrote a script that searches the comments of previous app's releases by accessing and handling the stacks of comments saved by the Google play store using a modified version of the Marketplace API code. Our data set consists of 154 randomly-chosen Android apps.

[2]*http://www.phonearena.com/news/*
[3]*http://www.developereconomics.com/reports/q3-2013/*

We downloaded the StackOverflow posts tagged with 'android' between August 2008 and January 2014. We extracted the API elements used by the Apps from the Android-related StackOverflow posts using Rigby and Robillard's tool that extracts code elements in freeform text [6]. The details of these phases is described below.

Phase 1 - Identifying API calls used by Apps.

To identify API elements used by the Apps, we downloaded the Android PacKage (APK) files for each release of each App. An APK file is a package file format used for the distribution and installation of application software and middleware onto operating systems such as the Google's Android OS. These files contain information such as the resources and software code including the complied classes in the dex (Dalvik EXecutable) format. Then, we extracted the API elements used by each App from the downloaded APK files following the three main phases.

- For each App release, we converted the APK file to jars files using the dex2jar[4] API;
- For each jar file corresponding to each App release, we use the JClassInfo[5] tool to extract API elements, *i.e.*, packages, classes, and methods used by the Apps;
- For each App release, we pruned the code elements obtained from JClassInfo and kept only the ones belonging to the **Android.*** package.

We wrote appropriate scripts to extract API calls and elements using the JClassInfo too. JClassinfo is written in C. It reads java class files and provides information about referenced packages, classes/interfaces as well as methods, etc. We extracted non-Android API elements, however, we were unable to use these in the analysis because the names of the classes and methods had been obfuscated. The obfuscated code elements could not be compared between releases to determine which had been added or removed.

Phase 2 - Computing API Elements Changes.

Once the list of API elements (*i.e.*, packages, classes, and methods) used in each App release is generated, we computed the changes in API elements between each two consecutive releases. We focus on high-level changes including of addition and removal of API elements between two versions. In our work in progress, we are investigating changes that are of implementation type by mining the Git versions histories of each individual Android APIs.

We capture addition and removal of API elements between two releases of an App. We computed these types of changes by computing the difference, in terms of API elements, between two consecutive releases, *i.e.*, V_1 and V_2 of an App. Therefore, an added or removed API element between V_1 and V_2 is defined as follows:

- *Added API element:* any API element that exists in the current release V_2 of an App and that has not been found in its preceding release V_1.
- *Removed API element:* any API element that does not exist in the current App release V_2 but occurs in its preceding release V_1.

Although we did not include unchanged API elements in our study, any API element that exists in both the current release V_2 of an App and its preceding release V_1 is an *unchanged API element*.

The formal definition of API elements changes between two releases V_1 and V_2 of an App is as follows:

The number of added and removed API elements between V_1 and V_2 is respectively A and R:

- $A = {V_1}^c \cap V_2$ where ${V_1}^c$ represents all the API elements that are not present in V_2.
- $R = {V_2}^c \cap V_1$ where ${V_2}^c$ represents all the API elements that are not present in V_1.

The set of changes C between V_1 and V_2 is:

$$C = A \cup R.$$

Consequently, the number of changes for a specific API element between two releases V_1 and V_2 of an App is the number of times this API element has been added or removed between V_1 and V_2.

Phase 3 - Extracting API Elements from StackOverflow.

StackOverflow is a Q&A forum for professional programmers[6]. In StackOverflow, developers post their questions and answers as well as their votes and comments. To determine the level of discussion about each changed API class and method (used by Android Apps) on StackOverflow, we extract the API classes and methods from the freeform text posts. Extracting code elements from software artifacts, such as documentation and requirements, has received significant research attention. For example, information retrieval techniques, such as Vector Space Models and Latent Semantic Index have been tried [7], [8], but have yielded low precision and recall (less than 65%). In our study, we use the Automated Code Element (ACE) extraction approach suggested by Rigby and Robillard *et al.* [6] to link StackOverflow discussions with the API elements that change in each App. The advantage of this approach is that unlike prior works [9], it does not rely on an index of valid elements parsed from the source code of a particular system and can process the massive number of Android related StackOverflow posts. ACE identifies code elements in Java constructs and creates an index of valid elements based on the elements contained in the collection of documents. ACE reparses each document extracting unqualified, ambiguous terms and resolves them to their corresponding code elements by using the term's context. It extracts code elements from freeform text and code that does not necessarily compile with an average precision and recall at or above 90%. The output of the ACE is a list of the code elements associated with each each documents, *i.e.*, Android StackOverflow posts in our case.

In this study, we process all question and answer posts on StackOverflow during the period going from August 2008 to

[4]*http://code.google.com/p/dex2jar*
[5]*http://jclassinfo.sourceforge.net*
[6]*http://stackoverflow.com/*

January 2014. Among these posts, we focus on the 591,555 posts that were tagged with Android. From these posts we extract the API elements that have been changed in the Apps. We had, in total, 479 distinct API classes and 2009 distinct API methods that have been changed by the Apps and discussed on StackOverflow. For each API element that has been changed in an App, we compute the number of discussions, *i.e.*, questions and answers that mention it in the StackOverflow.

B. Analysis Method

App Success: To define the analysis methodology, we analyzed the distribution of successful and unsuccessful Apps in our dataset. Not surprisingly, user ratings are, in general, very high for free Apps as the case in our study. High ratings for free Apps may be due to users' low expectations for free Apps while this is not the case when they pay for an App [3]. For paid apps, users' dissatisfaction quickly increases when they experience high App churn, malfunctions, crashes, and bugs. In our dataset, 95 Apps exhibit an average rating greater than four stars. We have 59 Apps with an average rating lower than four stars. We could, thus, verify whether there is a relationship between App churn and App success in terms of average user rating. It is important to clarify that as in previous works on Android apps [3], it was impossible to find a high number of unsuccessful apps to include in our analysis because the Google play store remove regularly low-rated mobile apps. More precisely, it removed 60K of low-quality apps on February 2013 [10]. Figure 1 reports the distribution of the average ratings assigned by users to the analyzed apps. Note that the number of ratings received by each app vary between 10 (the minimum we considered) and almost 500,000. In particular, 95 apps exhibit an average rating greater than 4 stars. Nevertheless, due to quite large corpus of apps considered in our study, we also have 59 apps with an average rating lower than 4 stars. Thus, we can verify a possible relationship between App churn and App success (in terms of average user's rating). We grouped the apps in two different groups on the basis of their average user ratings.

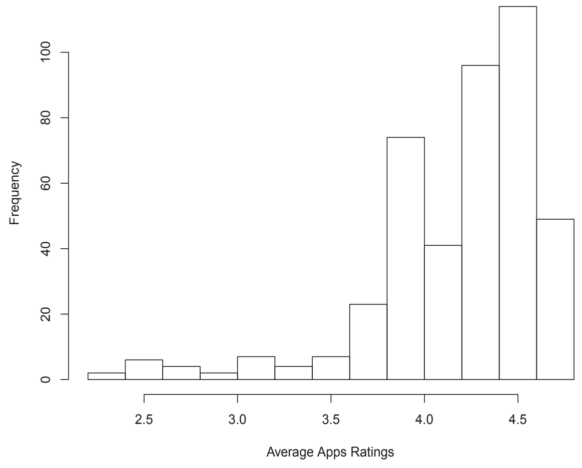

Fig. 1. Average user ratings for the analyzed apps.

TABLE II
AVERAGE RATING: DESCRIPTIVE STATISTICS.

Min	1Q	Median	Mean	3Q	Max
2.327	3.970	4.308	4.183	4.501	4.780

Table II shows descriptive statistics about the average ratings users provide for the studied Apps. As minimum we had 2.327, first quartile=3.970, median=4.308, mean=4.183, third quartile=4.501, and maximum=4.780. Based on the Apps distribution analysis, we grouped the apps in two different groups on the basis of their average user rating. In particular, given r the average user rating, the two sets are: (i) Apps having positive rating (*i.e.*, $r > 4$) (95 Apps), (ii) Apps having poor rating (*i.e.*, $r \leq 4$) (59 Apps). We present the descriptive statistics (*i.e.*, boxplots) and the results of the Mann-Whitney test [11]. We considered the two groups of Apps, *i.e.*, Apps having positive rating vs. Apps having poor rating, and we applied the Mann-Whitney test to analyze statistical significance of the differences between App churn, in terms of API elements changed, for the two groups of Apps. We used the Mann-Whitney test instead of other tests because our goal is to bring empirical evidence on the relationship between App churn and App success and not to claim causation. The results of the test are considered statistically significant at a level $p \leq 0.05$. Since we performed multiple tests, we had to adjust our p-values. To do so, we applied the Holm's correction procedure [12]. This procedure sorts the p-values resulting from n tests in ascending order, multiplying the smallest by n, the next by $n..1$, and so on. To show the extent to which the difference between changes in API elements used by different groups of Apps is statistically significant, we used the non parametric effect size measure Cliff's delta d [13]. Cliff's delta is used on ordinal data and it is interpreted as follows: small for $d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$ [13]. Our analysis approach and presentation of results mirrors that of Linares-Vasquez *et al.* [3].

Apps changes and StackOverflow posts: We grouped API elements (*i.e.*, classes and methods) on the basis of the number of changes they underwent summed across all Apps. As explained in Section II-A, for each API element, we computed its changes, *i.e.*, the number of times it has been added or removed between each two subsequent releases of an App. Based on the descriptive statistics of changes, we had three groups of API classes and API methods to analyze. The first group consists of API classes and methods that underwent on average (i) less than two changes between two releases of an App, *i.e.*, $0 < nc \leq 2$, (ii) the second group has more than two but less than four changes, *i.e.*, $2 < nc \leq 4$, and a group of APIs classes and methods having, on average, more than four changes between two releases of an App, *i.e.*, $nc > 4$. The goal is to analyze whether API classes and methods that underwent more changes stimulate more discussions from Android App developers. Like *RQ1*, our analysis is based on the non-parametric Mann-Whitney test [11] and Cliff's Delta [13] effect size measure. We consider two of the three groups of API classes (or API methods) at a time (*e.g.*, group having $0 < nc \leq 2$ vs. group having $nc > 4$) and then we run the Mann-Whitney test to show whether there exists a statistically significant difference between the number of questions posted for these groups of API classes/methods. The results were statistically significant at $p \leq 0.05$ and p-values were adjusted using the Holm's correction procedure [12]. Cliff's Delta was used to show the magnitude of the difference in terms of StackOverflow discussions for API classes and methods used by Apps belonging to the three studied groups.

III. Analysis of the Results

A. App Churn vs. App Success

RQ1: Does App churn affect App success?

Boxplots in Figures 2, 3, and 4 show the distributions of the number of changes in API packages, API classes, and API methods used by Apps having different average ratings. As explained in Section II, Apps are grouped into two sets on the basis of their average user rating (r). To make easy the readability of the boxplots, we apply the log transformation to the presented data since our distributions are skewed.

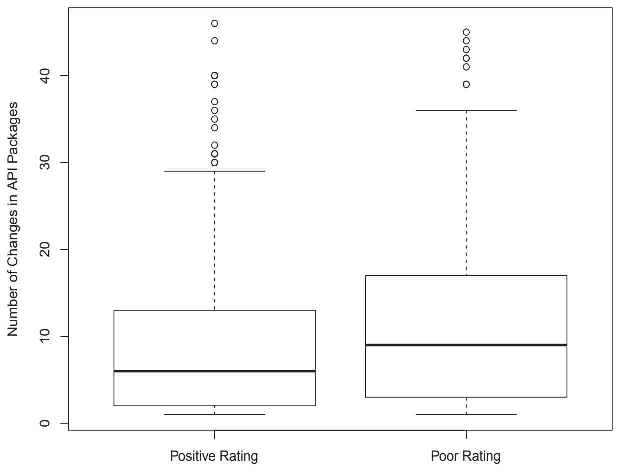

Fig. 2. Boxplots of number of API packages changes used by Apps having different average ratings.

The boxplots in Figure 2 show that Apps having higher ratings exhibit a lower number of changes in the API packages they use. In particular, Apps having positive ratings ($r > 4$) underwent, on average, 9 API packages changes. They have as a minimum of changes 1, first quartile=2, median=6, third quartile=13, and maximum=46. In contrast, Apps having low rating $r \leq 4$ underwent, on average, 12 API packages changes. They have as minimum one change, first quartile=3, median=9, third quartile=17, and maximum=45.

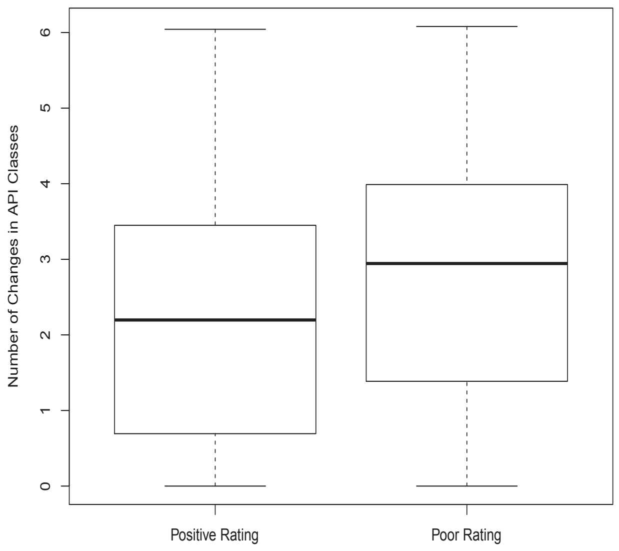

Fig. 3. Boxplots of number of API classes changes used by Apps having different average ratings.

Additionally, we notice from Figure 2 that the median and quartiles show an upward-trend, *i.e.*, as the ratings increases, the number of changes in API packages decreases. Similar trends were observed for the API classes. In fact, the group that consists of the positive ratings shows, on average, 35 changes in API classes. As minimum, they have one change while the first quartile=2, median=9, third quartile=31, and maximum=421. The group having low ratings underwent, on average, 60 API classes changes (cf. Figure 3). As minimum, they show one change while the first quartile=4, median=19, third quartile=54, and maximum=437. For API classes, the median and quartiles show an upward-trend of the number of changes with a decrease if the App average rating increase. Regarding API methods, Apps having positive ratings underwent, on average, 82 API methods changes. As minimum, they show one change while the first quartile=4, median=15, third quartile=55, and maximum=1174. In contrast, Apps with low ratings underwent, on average, 140 changes in API methods (cf. Figure 4). they show one change while the first quartile=7, median=30, third quartile=107, and maximum=1277.

Overall, we conclude from the boxplots that API packages, classes and methods used by Apps having lower ratings underwent more changes than API elements used by the highly

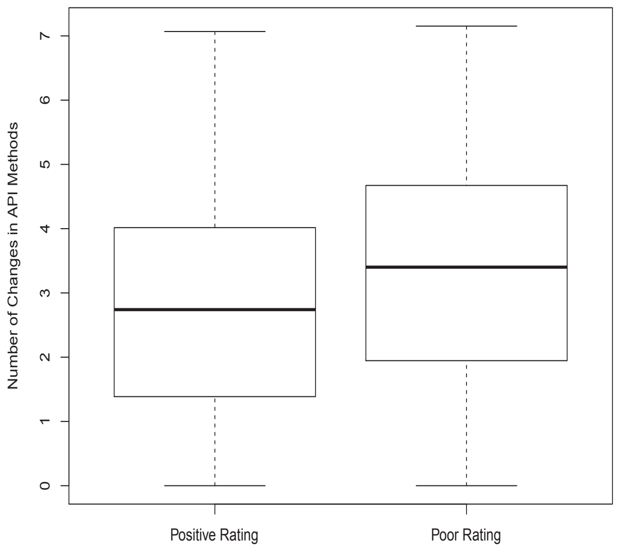

Fig. 4. Boxplots of number of API methods changes used by Apps having different average ratings.

rated Apps. In the following, we show the results obtained from the statistical tests.

TABLE III
CHANGES IN API PACKAGES USED BY APPS HAVING DIFFERENT AVERAGE RATING (r): MANN-WITHNEY TEST (p-VALUE) AND CLIFF'S DELTA (d).

Test	Adj. p-value	Cliff's d
Poor vs. Positive ratings	$<$ **0.007**	**0.63 (large)**

Table III reports the results of the Mann-Whitney test and Cliff delta d when comparing the number of changes in API packages used by Apps belonging to different groups of average user ratings. The main results from Table III can be summarized as follows. There is a statistically significant difference (p-value=0.007), in terms of the number of changes in API packages, when comparing Apps having low ratings with Apps having positive ratings with a large effect size (Cliff delta d=0.63) ($d \geq$ 0.54).

TABLE IV
CHANGES IN API CLASSES USED BY APPS HAVING DIFFERENT AVERAGE RATING (r): MANN-WITHNEY TEST (p-VALUE) AND CLIFF'S DELTA (d).

Test	Adj. p-value	Cliff's d
Poor vs. Positive ratings	$<$**0.005**	**0.64 (large)**

A similar analysis was performed for API classes, Table IV reports the results of the Mann-Whitney test and Cliff Delta d when comparing the changes in API classes used by Apps belonging to different groups of average user ratings. The main results from table IV can be summarized as follows. There is a statistically significant difference (p-value=0.005), in terms of the number of changes in API packages, when comparing Apps having low ratings with Apps having positive ratings with a large effect size (Cliff delta d =0.64) ($d \geq$ 0.54).

TABLE V
CHANGES IN API METHODS USED BY APPS HAVING DIFFERENT AVERAGE RATING (r): MANN-WITHNEY TEST (p-VALUE) AND CLIFF'S DELTA (d).

Test	Adj. p-value	Cliff's d
Poor vs. Positive ratings	$<$**0.001**	**0.63 (large)**

Table V reports the results of the Mann-Whitney test and Cliff delta d when comparing the changes in API methods used by Apps belonging to different groups of average user ratings. The main results from table IV can be summarized as follows. There is a statistically significant difference (p-value=0.001), in terms of the number of changes in API packages, when comparing Apps having low ratings with Apps having positive ratings with a large effect size (Cliff delta d=0.63) ($d \geq$ 0.54).

The results show that our conjecture is valid for the three investigated API element levels. Intuitively, one could assume that the results for API packages and classes should be consistent with the results obtained at the method level since changes in methods reflect changes at classes and packages as well. In this study, we bring empirical evidence on the fact that frequent changes in API elements (used by Android Apps) are, in general, a threat to the success of the Apps using them.

In summary, we can reject our null hypothesis, *i.e.*, API packages, API classes, and API methods used by successful Apps make, on average, less frequent changes in comparison with API elements used by unsuccessful Apps.

B. App Churn vs. StackOverflow Discussions

RQ2: Is there more community discussion about the API elements that App developers frequently change?

Similarly to the analysis method followed in *RQ1*, we analyzed the number of discussions in StackOverflow related to API classes and methods, which are subject to different number of changes. We found that the number of StackOverflow discussions increases for API classes that underwent more changes between releases of Apps. In particular, the number of discussions for API classes having an average number of changes $nc > 4$, *i.e.*, that make frequent changes (244,899 discussion) is higher in comparison with API classes having $2 < nc \leq 4$, *i.e.*, medium changes (197,264 discussion) and $0 < nc \leq 2$, *i.e.*, few changes (51,052 discussion).

TABLE VI
STACKOVERFLOW DISCUSSIONS PER API CLASSES UNDERGOING DIFFERENT AVERAGE CHANGES BETWEEN RELEASES: MANN-WHITNEY TEST (ADJ. p-VALUE) AND CLIFF'S DELTA (d).

Test	Adj. p-value	Cliff's d
Few vs. Medium changes	$<$**0.0001**	**-0.31 (small)**
Few vs. Frequent changes	$<$**0.0001**	**-0.34 (medium)**
Medium vs. Frequent changes	0.10	-0.08 (small)

The number of StackOverflow discussions also increases for API methods that underwent a higher average of changes between subsequent releases of Apps. In particular, the number of discussions for API methods making use of frequent changes (298,301 question) is higher in comparison with API methods undergoing medium changes (208,686 question) and methods having few changes (74,463 question).

Table VI reports the results of the Mann-Whitney test (p-value) and the effect size measure Cliff's delta d for the API classes. We compared each group of API classes (classified on the basis of the number of changes they underwent) with all other groups of API classes that underwent a smaller number of changes. As we can see in Table VI, the API classes that frequently change between releases of an App show a statistically significantly higher number of StackOverflow discussions than API classes that make use of a lower number of changes.

The results can be summarized as follows:

1) there is a statistically significant difference (p-value <0.0001) between API classes that underwent, on average, few changes and medium ones with a small effect size (Cliff's delta d =-0.31).
2) there is a statistically significant difference (p-value <0.0001) between API classes that underwent, on average, few changes and frequent changes with a medium effect size (Cliff's delta d =-0.34).
3) there is no statistically significant difference between API classes that underwent, on average, medium changes and frequent changes.

Similar analysis was performed at the API method level. As we can see in Table VII, API methods that are subject to more changes between releases of Apps are often related to a statistically significant higher number of discussions by StackOverflow developers than API methods that underwent a lower average number of changes (p-value $\leq$ 0.01). The results can be summarized as follows:

1) API methods that make a medium number of changes exhibit more StackOverflow discussions in comparison with API methods that underwent few changes with a statistically significant difference (p-value <0.001) and a medium effect size (Cliff's delta d =-0.34).
2) API methods that underwent frequently changes exhibits more StackOverflow discussions in comparison with API methods making use of few changes with a statistically significant difference (p-value <0.0001) and a medium effect size (Cliff's delta d =-0.43).
3) API methods that underwent frequently changes exhibits more StackOverflow discussions in comparison with API methods making use of a medium number of changes with a statistically significant difference (p-value=0.01) and a small effect size (Cliff's delta d =-0.15).

Interestingly, we find differences between results for methods and classes in two tests: the first test (*i.e.*, group of API classes/methods that underwent, on average, few changes vs. group that underwent medium changes) and the third test (medium vs. frequent changes). In effect, there is no statistical difference between classes that underwent medium changes vs. those that frequently change, in terms of StackOverflow discussions while methods exhibit a significant difference for such a test. Also, API methods exhibit a medium effect size in comparison with API classes that exhibit a small effect size when comparing the number of posts triggered for groups having few and medium number of changes. This is likely due to the fact that methods underwent, in general, more changes than classes.

TABLE VII
STACKOVERFLOW DISCUSSIONS PER API METHODS UNDERGOING DIFFERENT NUMBER OF CHANGES BETWEEN ANDROID RELEASES: MANN-WHITNEY TEST (ADJ. p-VALUE) AND CLIFF'S DELTA (d).

Test	Adj. p-value	Cliff's d
Few vs. Medium changes	**<0.0001**	**-0.34 (medium)**
Few vs. Frequent changes	**<0.0001**	**-0.43 (medium)**
Medium vs. Frequent changes	**0.01**	**-0.15 (small)**

On summary, we can reject our null hypothesis, *i.e.*, API classes and API methods that change very often between two releases of an App are more mentioned in the StackOverflow, *i.e.*, they were more likely difficult to understand by developers.

C. Qualitative Results

We performed a qualitative analysis to analyze the main reasons behind the users dissatisfaction and disappointment.

The quantitative analysis performed to answer our research questions provided us with strong empirical evidence that Apps having higher churn are generally those having lower success. Although we are aware this is not sufficient to claim causation we performed a qualitative analysis to (at least in part) find a rationale of the relation between App churn and the low success of some apps. First, we performed a coarse grained automatic analysis of comments left by users to apps having poor ratings, for a total of 95,499 comments. The goal of this analysis is just to get an idea of the main reasons behind the users dissatisfaction with unsuccessful apps. In particular, we are interested in understanding if negative comments are mainly related to lack of features in the apps (and thus, no relation with the app churn can be hypothesized), to bugs/malfunctions of apps (and thus, a possible relation with apps churn could exist), or both. For such a purpose, we extracted from the negative comments the n-grams composing them after pre-processing the comments (*i.e.*, removing stop-words, punctuation, etc.), considering $n \in [1..3]$. We found that most frequent n-grams from the top 100 frequent ones that we generated are related to problems associated with the normal functioning of the app: *does not work*, *crashes*, *please fix*, *app sucks*, *stopped working*, *start false*, *does not work*, *no longer works*. However, there are also comments that seem have a link with unsatisfactory features offered by the app: *useless*, *service stopped*, *device not compatible*, *wifi off*, etc. Therefore, as expected crashes and bugs/malfunctions, which could be due to high App churn, represent one of the main reasons behind users dissatisfaction with downloaded apps. The next step is to find insights about the relation between the app churn and the apps success. This steps consists of manually analyzing some of the unsuccessful apps on Google Play trying to understand if App churn directly impacted the apps' user experience. By analyzing the App chrun of one of the very poorly rated application belonging to the category *Media & Video* and perceived as useless by App users between two of its consecutive releases: releases 4.5.4 and 4.5.5, we found that 9 packages int it were subject to change between these two releases. In addition, 19 classes and 38 methods have been changed between the same releases. Examples of complaints reported by users of this app are : *Constant video errors*, *Garbage App Crashes after 20 seconds on S4*. A second very poorly rated App belonging to the category *Sports* was subject to 29 changes in its packages, 363 changes in its classes, and 1090 changes in its methods between two of its releases: 0.9.21 and 1.1.12. One of the users reports a complaint about crashes and bugs regarding this app : *It crashes before it even opens now. Wow. The app is tolerable, too bad it didn't work. May be the bugs can't be fixed because you're working on the new website, which is also crap*. A second review reports the following: *Loosing faith in App. This app just keeps getting worse and worse, please fix it so that it doesn't keep crashing*, and *It takes a long time to load, regardless of device. Please fix or I'll replace it with something else that works properly*. Another illustrative example concerns an App used for communication and rated as unsuccessful. The comments provided by the users, for this App, were mostly related to issues associated with App churn, presence of bugs, and issues related to functionality and features of this App. In the following, we present an example of review that we found for the this App in the Google Play Store:

Rating: **
A Google User - December 10, 2013
A lot of bugs. Program doesn't start properly. Crashes on first attempt to start. Usually opens on second attempt. Slow to load scores.

This can suggest that, very likely, it is difficult, for app developers, to stay tuned with changes performed in such Apps with high chrun which, more likely, leads to bugs in such apps. In general, the performed qualitative analysis confirmed the results of the quantitative one: App churn represent impact the success of Android apps.

IV. Threats to Validity

Construct validity deals with how well our measures represent real-world quantities. One concern is that the use of average ratings as an indicator of success can be based on subjective user votes. We are aware that such ratings can be highly subjective and imprecise. To mitigate such a threat, we analyzed a large sample of data, *i.e.*, 154 Android Apps and 1.2K of their releases and we discarded apps having less than ten ratings. Another threat is related to the cost of an App which may have a significant effect on the App rating. In effect, free Apps are, in general, highly rated since users have low expectations for such Apps. We mitigated such a threat by dealing with a large number (*i.e.*, 154) of Apps including 59 unsuccessful Apps. One more threat is related to the way the Apps use the API elements (*e.g.*, inheritance). In our study, the focus is on the changes of API elements and not on the type of their usage.

Internal validity deals with alternative explanations that may explain our results. We draw our conclusions based on the use of appropriate, non-parametric statistical tests, *i.e.*, Mann-Withney. In addition, we used a procedure to adjust the p-value obtained from multiple tests. Furthermore, we show the magnitude of the obtained differences in terms of a suitable non-parametric effect size measure, *i.e.*, Cliff's Delta. In this study, we bring empirical evidence of the relation between App churn and App success as well as between App churn and the number of discussions triggered in StackOverflow. However, we do not claim any causation but we believe App churn can be essential for the App success besides other factors such as its usability, features, etc. We supported our quantitative findings by a qualitative analysis consisting of automatically analyzing negative users' comments posted in the Google Play Market and manually checking some reviews and relating them to high Apps churn and discussions posted by the Android StackOverflow community.

Threats to *external validity* concern the generalization of our findings. We studied 154 Apps and 1.2K releases from the Android Play Store. While we downloaded over 2000 Apps, we considered only 154 to deal with apps having more than two releases. A larger sample that include iOS and other platforms might lead to different conclusions; however, we are confident in the conclusion that we have drawn from our sample. Finally, we make the data of this research investigation fully available for replication purposes up on request.

V. Related Work

Recent research has targeted the analysis of mobile Apps especially Android Apps. These works can be clustered as follows: App API usage; Android API stability; and Documentation, StackOverflow, and the Android API.

App API usage

Shabtai *et al.* [14] and Sanz *et al.* [15] have used machine-learning techniques to categorize Android Apps based on the bytecode extracted from Android Application Package (APK) files.

Mojica Ruiz *et al.* [16] analyzed code reuse in Android applications through a study of mobile Apps belonging to different categories in the Android Market. Their approach uses a technique previously applied by Davies *et al.* [17], [18] on the Maven Repository. They used the APK files of the Android Apps to extract the bytecode and, thus, generate class signatures to compute usage frequencies via inheritance and class reuse. The results of their study have shown that almost 23% of the classes inherit from a base class in the Android API, and 27% of the classes inherit from a domain specific base class.

Unlike previous work that examines the API elements in a single App, we examine multiple releases and determine how frequently Apps change the API elements they use.

Android API stability

Martie *et al.* [19] observed trends in the bug discussions in the Android open source project public issue tracker. The results of their study show that there is a relation between the issue topic distributions overtime and major development releases of the Android OS. Assaduzzaman *et al.* [20] mined buggy changes in the Android by computing the textual similarity between commit messages and bug reports. Additionally, they identified potential problematic parts in development of Android that can lead to maintenance implications.

Linares-Vasquez and *et al.* [3] analyzed how the fault-proneness and change-proneness of the Android platform's API influenced App success, as perceived by the users, in terms of votes, ratings, etc. The results of their empirical study indicate that fault-prone and change-prone API methods can negatively impact the success of the apps using them.

These studies examine trends in the Android platform. They identify problematic API elements based the change and error proneness of each element. Our work complements these works by examining the churn of individual Apps and relates this churn between App releases to success. We find that frequently changed Apps have lower ratings than more stable Apps. These findings suggest that release engineers should be caution when releasing new features that might introduce high Apps churn and lead to lower ratings.

Documentation, StackOverflow, and the Android API

Creating traceability links between code elements and developer discussions, documentation, and other textual sources has received significant research attention [8], [9], [21], [6]. Parnin *et al.* [22] linked code elements in StackOverflow threads discussions and API classes by using heuristics based on exact matching of classes names with words in posts (title, body, snippets, etc.). They provide interesting results on how quickly developers adopt change to APIs. Using a similar approach, Kavaler *et al.* [5] studied the relationship between Android usage of API elements and the discussion of these elements on StackOverflow. Both studies found a positive relationship between API class usage and number of StackOverflow discussion.

Recently, researchers [23] have shown that that developers in the SO community react to changes in Android APIs, in particular when the body of API methods is modified.

Our work expands these works by considering not only classes, but methods as well. Our linking is more accurate as we use the context of each API code element [6]. We also consider changes to Apps, instead of a static view of a large number of Apps.

VI. Conclusion

We make three novel contributions in this paper:

1) Using the element changes between releases, we relate App churn to App success through user ratings.
2) We relate the number of API class and method discussions on StackOverflow to the number of times these classes and methods were added or removed from the Apps in our sample.
3) We semi-automatically analyze 95,499 negative user App comments to understand the main reasons behind the dissatisfaction and disappointment of the users of unsuccessful Apps.

We find that less stable Apps tend to have lower ratings. In Section III-A we find, for example, that poorly rated Apps change on average 140 methods, while positively rated apps change only 82 methods. The trend of less successful Apps having more changes holds for packages and classes as well. Our findings could influence the decision to release new features at the expense of churn and possible low ratings.

API classes and methods that are changed more frequently by App developers have more posts on StackOverflow. For example, in Section III-B we find that methods that are involved in more than four changes have significantly more posts than methods involved in less than four changes. Our findings add to a growing body of knowledge [22], [5] that suggests that community documentation keeps up with the demands of software developers.

In Section III-C we triangulated our quantitative findings. Automatically analyzing 95,499 negative comments from users of poorly rated Apps showed that users mostly complain about issues related to crashes, bugs as well as malfunctions.

Future work could combine our method of extracting code elements between App releases and relating these to the quality of the underlying Android platform as studied by Linares-Vasquez *et al.* [3]. We expect that relating App changes to App success will allow empirical software researchers to tie other characteristics to software success.

References

[1] D. M. German, "Using software distributions to understand the relationship among free and open source software projects," in *Mining Software Repositories Conference*, 2007.

[2] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan, "What Do Mobile App Users Complain About? A Study on Free iOS Apps," *IEEE Software*, 2013.

[3] M. Linares-Vasquez, G. Bavota, C. Bernal-Cardenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: a threat to the success of android apps," in *ESEC/SIGSOFT FSE*, 2013, pp. 477–487.

[4] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *Mining Software Repositories Conference*, 2012, pp. 179–188.

[5] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov, "Using and asking: Apis used in the android market and asked about in stackoverflow," in *International Conference of Social Informatics*, 2014, pp. 405–418.

[6] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *International Conference on Software Engineering*, 2013, pp. 832–841.

[7] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[8] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *International Conference on Software Engineering*, 2010, pp. 375–384.

[9] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *International Conference on Software Engineering*, 2012, pp. 47–57.

[10] S. Perez, *Nearly 60K Low-Quality Apps Booted From Google Play Store In February, Points To Increased Spam-Fighting*.

[11] W. J. Conover, *Practical Non-parametric Statistics.*, 1998.

[12] S. Holm, "A simple sequentially rejective Bonferroni test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.

[13] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[14] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *International Conference on Computational Intelligence and Security*, 2010, pp. 329–333.

[15] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. Bringas, "On the automatic categorization of android applications," in *Consumer Communications and Networking Conference*, 2012, pp. 149–153.

[16] I. M. Ruiz, M. Nagappan, B. Adams, and A. Hassan, "Understanding reuse in the android market," in *International Conference on Program Comprehension*, 2010, pp. 113–122.

[17] J. Davies, D. M. German, M. W. Godfrey, and A. J. Hindle, "Software bertillonage: Finding the provenance of an entity," in *Working Conference on Mining Software Repositories*, 2011.

[18] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle, "Software bertillonage - determining the provenance of software development artifacts," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1195–1237, November 2013.

[19] L. Martie, V. Palepu, H. Sajnani, and C. Lopes, "Trendy bugs: Topic trends in the android bug reports," in *Working Conference on Mining Software Repositories*, 2012.

[20] M. Assaduzzaman, M. Bullock, C. Roy, and K. Schneider, "Bug introducing changes: A case study with android," in *Working Conference on Mining Software Repositories*, 2012, pp. 116–119.

[21] N. Bettenburg, S. Thomas, and A. Hassan, "Using fuzzy code search to link code fragments in discussions to source code," in *European Conference on Software Maintenance and Reengineering*, 2012, pp. 319–328.

[22] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and dynamics of api discussions on stack overflow." Technical Report GIT-CS-12-05, Georgia Tech, Tech. Rep., 2012.

[23] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014, pp. 83–94. [Online]. Available: http://doi.acm.org/10.1145/2597008.2597155