How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK

Mario Linares-Vásquez[1], Gabriele Bavota[2], Massimiliano Di Penta[2], Rocco Oliveto[2], Denys Poshyvanyk[1]
[1]The College of William and Mary, Williamsburg, VA, USA [2]University of Sannio, Benevento, Italy
[3]University of Molise, Pesche (IS), Italy
mlinarev@cs.wm.edu, gbavota@unisannio.it, dipenta@unisannio.it, rocco.oliveto@unimol.it, denys@cs.wm.edu

ABSTRACT

The growing number of questions related to mobile development in StackOverflow highlights an increasing interest of software developers in mobile programming. For the Android platform, 213,836 questions were tagged with Android-related labels in StackOverflow between July 2008 and August 2012. This paper aims at investigating how changes occurring to Android APIs trigger questions and activity in StackOverflow, and whether this is particularly true for certain kinds of changes. Our findings suggest that Android developers usually have more questions when the behavior of APIs is modified. In addition, deleting public methods from APIs is a trigger for questions that are (i) more discussed and of major interest for the community, and (ii) posted by more experienced developers. In general, results of this paper provide important insights about the use of social media to learn about changes in software ecosystems, and establish solid foundations for building new recommenders for notifying developers/managers about important changes and recommending them relevant crowdsourced solutions.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Documentation

Keywords

Android, StackOverflow, API changes, Social media

1. INTRODUCTION

When learning about new development techniques or in general to solve any specific development problem, developers may rely on different sources of information. Recently, other than using mailing lists or project-specific forums, developers rely on crowdsourced resources [23, 26, 38, 41, 42, 43] such as Stack Overflow (SO) [21], a question/answer site about software development. The same phenomenon is true for mobile development; a recent study by Barua *et al.* [9] on SO concluded that mobile development is a SO trend topic, and the trend increments even faster than Web development. The SO dump as of August 2012 [7] contains 213,836 questions, which were tagged with labels including the "android" keyword (e.g., *android*, *android-layout*, *google-maps-android-api-2*). Also, the average number of Android-related questions posted monthly on SO was 12,515 during 2012, and the number of questions posted each year were 41 (2008), 2,054 (2009), 28,472 (2010), 95,658 (2011), and 87,611 (2012).

More specifically, in the case of mobile development, there is a rapid diffusion of hand-held devices and tablets, and many developers are getting interested in mobile development, in particular in using Android as the software platform because of its open-source model and tremendous commercial success. Business models associated with mobile-development motivate developers to build applications using languages and frameworks that are slightly or drastically different than the traditional ones. Consequently, developers learn new APIs on demand by doing or by finding solutions/advice on different sources of information, such as official documentation or discussion forums [31, 37]. Therefore, knowledge portals like SO become main actors and widely used sources of documentation when developers look for answers on specific topics related to programming languages and technologies.

According to Mamykina *et al.* [26], some reasons for contributing in Question and Answer (Q&A) systems are altruism, learning, rewards, and reputation. In particular, when using frameworks and APIs, developers ask for advice when implementing a solution based on unfamiliar or undocumented APIs, when using a wrong API-based solution, or when using an API-based solution incorrectly [20]. However, mobile-related questions in SO are not only related to programming, and there is no existing empirical evidence of specific causes that motivate mobile developers to find and/or provide answers in a Q&A system like SO. A study by Linares-Vásquez *et al.* [25] suggests that hot-topics in SO mobile development related questions include general topics such as device compatibility and IDE-related issues, and specific topics related to database, media, and maps programming. For instance, two examples of questions in SO with more than 700 votes as of January 2013, are the

ICPC '14, June 2-3, 2014, Hyderabad, India

question 101754 [2], that asks if there is a way to run Python on Android, and the question 1554099 [4] with more than 1000 votes, which looks into solving an issue with an Android emulator running on a slow machine.

Thus, *what are the main reasons that encourage Android developers to ask questions on SO*? Our conjecture is that *instability (change-proneness) of the Android APIs is one of those reasons that motivates developers to ask questions and seek answers in SO*. Table 1 lists the first 17 Android versions, the number of classes that were added to the APIs, and the changes in methods (i.e., methods added, methods removed, signatures modified) that were implemented in each version. Since October 2008 to November 2012, 17 Android API revisions were released, and each version had an average number of 6,033 changes in methods. Android API change-proneness is a threat for the success of Android applications [24], and previous studies by Mojica-Ruiz *et al.* [29, 28] and Syer *et al.* [40, 39] showed that those apps heavily rely on the Android APIs, by using inheritance or API calls. Consequently, new Android releases could potentially trigger developers' rush to identify—in official and unofficial documentation—new API features, or solve backward-compatibility issues and bugs introduced by the API changes.

To evaluate our conjecture, we conducted an empirical study aimed at investigating how developers react to API instability, in particular their reaction on SO when changes are released on the Android APIs. We implemented an approach based on the one reported by Parnin *et al.* [33] to identify the SO questions related to Android APIs after a new API version was released. Then, we mapped SO questions to APIs at method level granularity. To trace those links, we adopted the approach proposed by Panichella *et al.* [32] for mining method descriptions in informal documentation such as emails and/or bug reports. To validate the precision of tracing SO questions to API methods, we manually validated a sample of randomly selected links.

Once we established the methods-to-questions links, we analyzed whether methods in the Android API with higher levels of change-proneness are discussed more on SO as compared to methods with lower levels of change-proneness. Also, we analyzed which kinds of changes in APIs—e.g., changes in method body, public methods removed—stimulate more questions from developers and which trigger the most relevant questions on the basis of two factors: (i) the number of answers provided by SO contributors, and (ii) the question score as recorded in SO. Finally, we verified which type of changes trigger questions by more experienced software developers.

Results of our study suggest that *change-proneness of API methods impacts the volume of the discussions among the developers in the SO community*, in particular, the changes performed on methods' body, because developers are usually confused if a method's behavior is different from what is expected on previous Android releases, in their experience. In addition, *deleting public methods from APIs is a major trigger for questions that are (i) more discussed and of major interest for the community and (ii) posted by more experienced developers.*

Our findings can be used by consumers of APIs as a reminder for checking carefully release information before upgrading applications to support new API versions. Moreover, these results could be also useful for API designers/ developers as a motivation for finding effective ways of notifying developers' community about sensitive changes (e.g., methods removed, changes in exceptions, changes to methods behavior) introduced in new API releases. Our dataset of Android-related questions in SO and links to the Android APIs is available online for other researchers interested in replicating, validating or building upon our results.

Table 1: Android API Versions and Changes.

Version	Release date	New classes	Changes in methods
1.0 Base	10/2008	2,236	-
1.1 Base	02/2009	3	2,618
1.5 Cupcake	05/2009	110	42,789
1.6 Donut	09/2009	43	5,178
2.0 Eclair	11/2009	107	1,259
2.0.1 Eclair	12/2009	0	811
2.1 Eclair	01/2010	7	899
2.2 Froyo	06/2010	127	10,935
2.3 Gingerbread	11/2010	117	11,092
2.3.3 Gingerbread	02/2011	16	4,170
3.0 Honeycomb	02/2011	229	1,411
3.1 Honeycomb	05/2011	22	1,940
3.2 Honeycomb	06/2011	3	675
4.0 Ice Cream	10/2011	118	2,600
4.0.3 Ice Cream	12/2011	15	1,122
4.1 Jelly Bean	06/2012	72	4,259
4.2 Jelly Bean	11/2012	44	4,766

Structure of the paper. Section 2 defines our empirical study and the research questions, and provides details about the data extraction process and analysis method. Section 3 reports the results, and discusses them from a quantitative and qualitative points of view. Section 4 discusses the threats that could affect the validity of the results achieved. Section 5 relates this study to previous works. Finally, Section 6 concludes the paper and outlines directions for future work.

2. STUDY METHODOLOGY

The *goal* of this study is to analyze Android related discussions on SO and Android API changes, with the *purpose* of investigating (i) whether the change-proneness correlates with the number of discussion threads among developers; and (ii) whether certain types of API changes trigger more questions by developers, and create helpful/relevant questions for them. The *perspective* is of researchers interested in building recommenders aimed at notifying developers/managers about important changes and recommending them relevant solutions. The *context* of the study consists of (i) 213,836 Android-related questions extracted from the SO dump as of August 2012; and (ii) the Android APIs together with their change history. We chose Android as the context of our study since it is the only platform among the top developer-mindshare platforms (i.e., Apple iOS and Google Android) with open-source APIs. Moreover, as shown in Table 1, Android APIs have been subject of numerous changes in a relatively short span of time.

2.1 Research Questions

In the context of our study, we formulated the following research questions:

- RQ_1: *Does API change-proneness correlate with the number of discussion threads?* This research question

serves as a preliminary question to the other two, and aims at verifying whether API methods having higher levels of change-proneness are discussed more heavily by developers in SO.

- RQ_2: *Which are the types of API changes triggering more questions from developers?* This research question aims at analyzing which types of source code changes trigger more questions from developers using the changed API methods.
- RQ_3: *What kind of API changes trigger more relevant questions and involve more experienced developers?* This analysis is useful to understand which types of API changes are more relevant for developers. We used the number of answers received by a question and its score as measures of the more helpful answers for developers as in [31]. In addition, we also used the reputation of the question owners as measure for determining highly relevant questions; our assumption here is that the most experienced developers provide well specified questions that may represent more relevant and serious questions.

The **independent variable** considered for the first research question (RQ_1) is the number of changes in an API method having occurred in the time window between two subsequent releases of the Android APIs. Concerning the other two research questions (RQ_2 and RQ_3), we still analyze the number of changes an API method undergoes between two releases, however distinguishing between:

- the overall number of method changes;
- the number of changes to public methods;
- the overall number of changes in a method body;
- the number of changes in public method signatures (method names, parameters, return types, visibility);
- the number of public methods added and removed; and
- the number of changes to the set of exceptions thrown by public methods, as detected by analyzing their signatures.

The **dependent variable** for the first two research questions (i.e., RQ_1 and RQ_2) is the number of questions posted by developers for each Android API method in a specific time frame. Our assumption is that questions related to changes performed between two subsequent versions of APIs (i.e., ver_i and ver_{i+1}) are posted in SO from the date when the version ver_{i+1} is released to the date when a new version (i.e., ver_{i+2}) is released.

Concerning the third research question (RQ_3), the **dependent variables** are represented (for the same set of questions identified for RQ_1 and RQ_2) by three factors indicating the relevance of posted questions, namely (i) the number of answers, (ii) the question score, and (iii) the question owner reputation. Such information is available from SO. While the number of answers for a posted question is quite straightforward to understand, the other two indicators need more details. The question owner reputation, as explained in the SO Frequently Asked Question page [22], is an approximate measurement of how much SO users trust that user. The reputation of a SO user grows when he/she posts good questions and useful answers. Thus, in the context of our study we assume that users having a higher reputation are more experienced developers and thus, questions posted by them may represent more relevant and serious questions. Concerning the question score, it is an indication of how many SO users consider a question as relevant. Each SO user can increase or decrease the score of a question by one unit. Also in this case, for the focus of our paper, questions having a higher score are assumed to pose problems or concerns that are experienced by more developers.

2.2 Data Extraction Process

The data required to answer our research questions consists of (i) the SO questions related to the Android APIs; (ii) the links between the Android APIs SO questions and the API methods; and (iii) the change history of those APIs.

To extract Android-related questions in SO, we used the August 2012 SO dump provided as an SQL script for the MSR 13 challenge [7]. We considered all the questions with tags including the *android* keyword (e.g., *android*, *android*-asynctask, *android*-sqlite). To link the Android-related questions to methods in the Android APIs we first identified for each question, the Android classes related to the question using an approach similar to the one reported by Parnin *et al.* [33]. In addition, we removed Java and logcat [16] stacktraces inside <*code*></*code*> HTML tags, by matching the content to the regular expressions in Figure 1. We removed the stacktraces, because in our initial investigation we found them as a source of a considerable number of false positives, rather than as a source of information that helps precisely link questions to source code.

We identified four types of links between a SO question and Android classes:

- **Type 1 (code-markup link):** an exact match of a class name occurring in the text inside <*code*></*code*> HTML tags.
- **Type 2 (href markup link):** an exact match of a class name occurring in the text inside <*a*></*a*> where the link points to a webpage in the official Android reference guide [15].
- **Type 3 (word link):** an exact match of a class name in the question body after removing the content inside <*a*></*a*> and <*code*></*code*> tags.
- **Type 4 (title link):** an exact match of a class name in the question title.

For each type of link, we built a detector (filter) that uses a class in the official Android reference guide [15] and an SO question as arguments. Then, we concatenated the four detectors in a conditional pipe-and-filter sequence using the order relationship defined by the types of link enumeration (Type 1 $\prec$ Type 2 $\prec$ Type 3 $\prec$ Type 4). We used this strategy in order to reduce the number of false positives when linking Android API class names to SO questions, because names of Android classes in some cases are common words.[1] The lower the position of the type of the link in the order relationship, the lower the chance for it to be a false positive; for example when SO contributors mention an API class inside the tags

[1]This phenomena is described as *ambiguity* by Dagenais and Robillard [14].

Stacktrace

```
java.lang.RuntimeException: Unable to start
activity
java.lang.NullPointerException
at
android.app.ActivityThread.performLaunchActivit
y(ActivityThread.java:2059)
```

```
09-20 22:36:22.293: ERROR/AndroidRuntime(311):
09-20 22:36:22.394: ERROR/dalvikvm(311):
```

Regexp

```
(Exception:|Error:)(.)*?\\([A-Z][a-
z0-9]*\\.java:[0-9]*\\)
```

```
(V|D|W|I|E|F|INFO|DEBUG|WARN|ERROR|
FATAL|VERBOSE)/(.)*?\\((\\s+){0,1}[0-9]
*\\)
```

Figure 1: Android stacktraces and regular expressions.

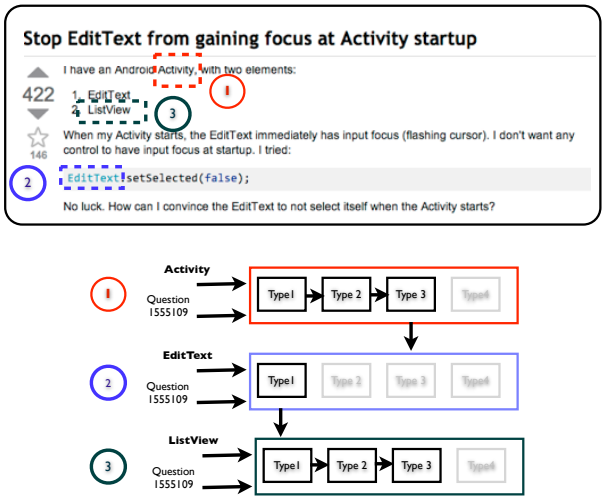


Figure 2: Pipes-and-filters for linking SO questions to API classes.

<code></code> or have a link to the documentation of a class at the official developer guide is because there is a relationship between the question and the API; however, having an API class name in the body or in the title of the question does not assure that the question is related to the API (e.g., window, activity, menu).

Consequently, for each Android-related question, we used pipe-and-filter sequences aimed at recognizing all Android classes belonging to the 17 API releases. We coined this approach as conditional pipes-and-filter, because any time a filter identifies a link, the next filter is not executed. Figure 2 depicts the linking process applied to question number 1555109 [3]. Let us assume that the target list of classes only includes *Activity*, *EditText*, and *ListView*, and thus we want to trace links to those classes from the question 1555109 (Figure 2-top). There are three sequences of pipe-and-filter to identify the classes (Figure 2-bottom). The sequence labeled with 1 identifies a Type 3 link to *Activity*; sequence 2 identifies a Type 1 link to *EditText*; and sequence 3 identifies a Type 3 link to *ListView*.

By using this approach, we identified 963,126 links between SO questions and Android classes with the following distribution: 791,377 Type 1, 4,097 Type 2, 141,003 Type 3, and 26,649 Type 4. Those links trace to 2,158 classes, representing 66% of the 3,266 classes belonging to Android

Algorithm 1 Question2Methods(C, Q)

```
1: results ← ∅
2: for all class C linked to question Q do
3:    for all method m defined in class C do
4:       if m appears in Q title, body, or code fragment
         then
5:          results ← m
6:       end if
7:    end for
8: end for
9: if results == ∅ then
10:    discard Q from the analysis
11: end if
```

packages. We were able to link 151,988 SO questions to at least one Android API class. Then, for each of these questions we linked questions to the API methods following the procedure reported in Algorithm 1. Specifically, a method belonging to one of the classes related to a specific question Q is related to such a question if it is cited in the question title, or body (including code fragments). If no methods belonging to classes related to Q are cited, then the question is discarded.

Using such an approach we found 165,570 links between SO questions and API methods. Since the linkage of questions to methods is not trivial, the approach reported in Algorithm 1 could generate some false positives. To have an indication of the accuracy of the approach we used, three authors and one independent collaborator manually validated a random sample of 100 questions aiming at identifying in how many cases the approach correctly links the question to method(s). Note that the exploited sample of 100 questions ensures a confidence interval of 9.8% with a confidence level of 95%. After validation, we computed for each subject the precision (PREC) as the number of declared *true positives* out of 100, and the false positive rate (FPR) as the number of declared *false positives* out of 100. In addition, we estimated the agreement between the subjects as the number of times (true positive or false positive) the four subjects gave the same answer. On average the PREC and FPR were 84.5% and 15.4% respectively with a total agreement in 78% of the links. The links used for the manual validation and the results are within our replication package.

Finally, we mined the change history of the Android APIs from their Git repositories [1]. We analyzed 35,703 developers' commits submitted between September 2009 to January

2013, and having a total of 370,180 method changes. We used a code analyzer developed in the context of the Markos European project [10] to compare the APIs before and after each commit at a fine-grained level of granularity. In particular, while the Git logs just report the changes in a commit at file level granularity, we used the *Markos code analyzer* to capture changes at method level, and to categorize them in four types: (i) generic changes (including all kinds of changes); (ii) changes applied to the method body; (iii) changes applied to the method signature (i.e., visibility change, return type change, parameter added, parameter removed, parameter type change, method rename); and (iv) changes applied to the set of exceptions thrown by the methods. Moreover, we distinguished between changes performed to *public* and to *non public* methods.

After having analyzed all the Android APIs, we used that information to compute the changes (at method level) performed between two subsequent versions of the APIs.

2.3 Analysis Method

To answer RQ_1, for each time frame under analysis (i.e., the period of time going between two subsequent Android releases), we grouped the API methods in four different groups on the basis of the number of changes (n_c) they underwent. The four sets consist of methods that underwent (i) no changes ($n_c = 0$), (ii) $0 < n_c \leq 5$ changes, (iii) $5 < n_c \leq 10$ changes, and (iv) $n_c > 10$ changes. Then, we analyzed whether methods that underwent more changes stimulated more questions from developers. The analysis was performed through descriptive statistics and the Mann-Whitney test [13]. For the latter, we considered two of the four groups of methods at a time (e.g., *methods having* $n_c > 10$ *vs. methods having* $n_c = 0$), and we used the Mann-Whitney test to analyze statistical significance of the differences between the number of questions posted for those methods by developers. The results were intended as statistically significant at $\alpha = 0.05$. Since we performed multiple tests, we adjusted our p-values using Holm's correction procedure [18]. 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. We also estimated the magnitude of the difference in terms of asked questions for methods belonging to the four described groups; we used Cliff's Delta (or d), a non-parametric effect size measure [17] for ordinal data. We followed the guidelines in [17] to interpret the effect size values: small for $d < 0.33$ (positive as well as negative values), medium for $0.33 \leq d < 0.474$ and large for $d \geq 0.474$.

To answer the other two research questions (i.e., RQ_2 and RQ_3) we report descriptive statistics, and analyze the presence of significant differences using Mann-Whitney test and evaluate their magnitude using Cliff's Delta effect size measure. In particular, for RQ_2 we compared the distribution of questions for the different types of changes considered in our study. For RQ_3, we compared the distribution of the three independent variables presented in Section 2.1 (i.e., number of answers, question owner reputation, and question score) for questions linked to methods underwent the different types of changes.

Finally, we should point out that in our data analysis barcharts are preferred to boxplots due to the highly skewed distributions of the data, which make boxplots hard to read even in log scale. However, as explained above, our conclusions are supported by appropriate statistical tests and effect size measures.

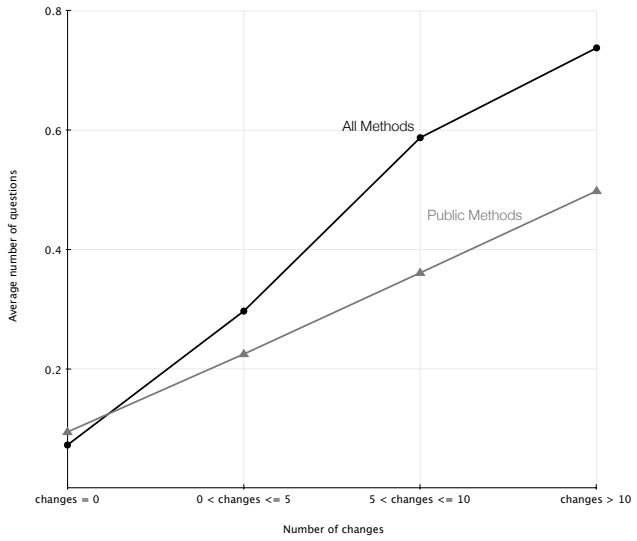


Figure 3: Average number of questions per methods, which are subject to different number of changes

2.4 Replication Package

All the data used in our study are publicly available at http://www.cs.wm.edu/semeru/data/ICPC14-so-android/. In particular, we provide: (i) the list of Android-related question in the August 2012 dump of SO, (ii) the list of links between SO questions and Android classes, (iii) the list of links between SO questions and Android methods, and (iv) complete information on the changes that occurred in the Android APIs, and the list of links used in the manual validation.

3. ANALYSIS OF THE RESULTS

This section reports the results aimed at answering the three research questions formulated in Section 2.1.

3.1 RQ1: Does API Change-Proneness Correlate With the Number of Discussion Threads?

Figure 3 shows the average number of questions in SO related to methods, which are subject to different number of changes. Data are reported by considering all the methods (black line) as well as by only counting public methods (gray line). It is quite obvious to observe the trend showing a growing number of questions for API methods that underwent more changes between two Android API releases. For example, by focusing on all the methods, unchanged methods are objects, on average, of 0.07 questions each as compared to the 0.30 (4.3 times more) of methods having $0 < n_c \leq 5$, 0.59 (8.4 times more) for methods having $5 < n_c \leq 10$, and 0.74 (10.6 times more) for methods that underwent more than 10 changes.

While focusing on public methods only the trend remains similar, even if with smaller increases. The average number of questions for methods subject to more than ten changes (0.50 questions each) is 5.5 times higher than for unchanged methods (0.09 questions each).

Table 2 reports the results of the Mann-Whitney test (p-value) and the Cliff's d effect size. We compared each set of methods (grouped by number of changes they underwent to) with all other sets that underwent to a smaller number of changes (e.g., $n_c > 10$ vs. all other cases). For each comparison, the group of methods (if any) subject of more questions by developers is reported in **bold** face when statistically significant differences are observed. As we can notice from Table 2, methods that are subject to more changes between two releases are often related to a statistically significantly higher number of questions by SO developers than methods that undergo a lower number of changes (p-value < 0.05). This is not true when comparing methods (all as well as just public methods) subject to more than ten changes *vs* those with a number of changes between five and ten. Also, we did not observe statistically significant results when isolating the analysis to public methods and comparing methods having $10 < n_c \leq 5$ with those having $0 < n_c \leq 5$ and when comparing these latter with unchanged methods (last row in Table 2).

The Cliff's d for the achieved statistically significant results is *medium* (-0.44) when comparing methods having $n_c > 10$ and methods having $0 < n_c \leq 5$, as well as when comparing the most modified methods with less ones. As for the other comparisons when considering all the methods, we always observed a *small* effect size. Also, when just focusing on public methods we observed *medium* effect size when comparing methods having $n_c > 10$ and those having $n_c \leq 5$.

We also compared the methods with the highest number of changes—i.e., those that underwent more than 20 changes between two Android releases—with the unchanged methods. While the latter exhibits on average just 0.07 questions each, the top changed methods reach, on average, 1.77 questions per method (25 times more), with a statistically significant difference (p-value < 0.01) and a *large* effect size (-1.12).

> In summary, the results for RQ_1 highlight that *the change-proneness of API methods impacts the volume of developer discussions in the SO community.* In fact, *the higher the number of changes to a method between two Android releases, the higher the number of questions talking about that method when the new release of the Android APIs is made publicly available.*

3.2 RQ2: Which Are the Types of API Changes Triggering More Questions From Developers?

To answer RQ_2, we separated methods into different sets, based on the type of changes they underwent between two Android releases. The seven considered sets are (i) methods with at least one change, (ii) public methods (PM) with at least one change, (iii) methods with at least one change in their body, (iv) public methods with at least one change in their signature, (v) public methods added, (vi) public methods deleted, and (vii) public methods with added or removed thrown exceptions.

Figure 4 reports, for each of these sets of methods, the average number of questions concerning each method. The first two bars reported in gray–i.e., methods with at least one change and PM with at least one change–are shown as baselines to better understand the magnitude of the difference between the number of questions posed by developers for different types of changes.

Table 2: Questions per methods, which are subject to different number of changes between two Android releases: Mann-Whitney test (adj. p-value) and Cliff's Delta (d).

Considering all methods

Test	adj. p-value	d
$(n_c > 10)$ *vs* $(5 < n_c \leq 10)$	0.09	-0.29 (Small)
$\mathbf{(n_c > 10)}$ *vs* $(0 < n_c \leq 5)$	<0.01	-0.44 (Medium)
$\mathbf{(n_c > 10)}$ *vs* $(n_c = 0)$	<0.01	-0.39 (Medium)
$\mathbf{(5 < n_c \leq 10)}$ *vs* $(0 < n_c \leq 5)$	<0.01	-0.31 (Small)
$\mathbf{(5 < n_c \leq 10)}$ *vs* $(n_c = 0)$	<0.01	-0.30 (Small)
$\mathbf{(0 < n_c \leq 5)}$ *vs* $(n_c = 0)$	<0.01	-0.15 (Small)

Just public methods

Test	adj. p-value	d
$(n_c > 10)$ *vs* $(5 < n_c \leq 10)$	0.44	-0.35 (Medium)
$\mathbf{(n_c > 10)}$ *vs* $(0 < n_c \leq 5)$	0.04	-0.36 (Medium)
$\mathbf{(n_c > 10)}$ *vs* $(n_c = 0)$	0.01	-0.35 (Medium)
$(5 < n_c \leq 10)$ *vs* $(0 < n_c \leq 5)$	0.14	-0.20 (Small)
$\mathbf{(5 < n_c \leq 10)}$ *vs* $(n_c = 0)$	0.04	-0.23 (Small)
$(0 < n_c \leq 5)$ *vs* $(n_c = 0)$	0.13	-0.13 (Small)

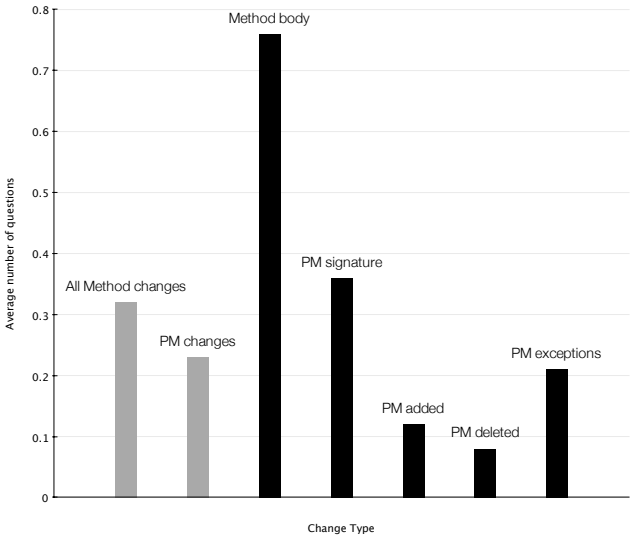


Figure 4: Average number of questions per methods that underwent different types of changes

As we can notice, changes to method body are those triggering, on average, more questions from developers. In fact, while the average number of questions for changed methods is 0.32, this number grows up to 0.76 (+138%) for methods underwent at least one "body change". By manually analyzing questions related to methods object of body changes, we found as often questions are triggered by the fact that developers are confused when the behavior of a method they use changes between two subsequent Android releases. An example is question 12354725[6] triggered by the changed behavior of method `onKeyDown` between Android Jelly Bean 4.1 and 4.2:

> *In older releases I was able to catch the search key (keyCode == KeyEvent.KEYCODE_SEARCH) when I was using the onKeyDown method. But now with new release of android it is not working anymore.Is there another possibility to catch the search key?*

Another type of change triggering questions from developers occurs when the signature of a public method is modified (column "PM signature" in Figure 4). In fact, methods that underwent this type of change receive, on average, 0.36 questions each, against the 0.23 (+57%) of all public methods that underwent to at least one change (column "PM changes" in Figure 4). We manually analyzed these questions to understand what are the problems experienced by developers in these cases. What we found is that in most cases developers' questions are targeted to understand how to manage the new method signature including new parameters. In some cases, the old version of the method was also left in the APIs (as deprecated) creating also more doubts in developers (see e.g., question 4904660 [5]).

Added and deleted public methods do not generate a high number of questions by developers. In other words, developers ask more questions when something changes in the methods they use and they would like to continue using (changes to method body and to PM signatures). Instead, they ask fewer questions about new services offered by the APIs (PM added) and services that are no longer available in the APIs (PM deleted).

Finally, we did not observe a strong difference in the average number of questions posted for public methods with changes in the exceptions (0.22) as compared to the average number of questions for changed public methods (0.23), representing our baseline in this case.

As explained in Section 2.3, we also computed the Mann-Whitney test and the Cliff's d to compare the distribution of questions for different types of changes considered in our study. We compared each type of change with all others types of changes. Given the high number of performed tests, we just report the statistically significant results in Table 3. Changes triggering a statistically significant higher number of questions from developers are highlighted in **bold** face.

We achieved significant results when comparing the number of questions related to methods that underwent changes to method body with those related to added and deleted PM, as well as to PM with changes in signature, always observing a medium effect size. Also, changes to PM signatures generate more questions than added and removed public methods (small effect size). Finally, added public methods, as well as methods undergoing changes in thrown exceptions, generate more questions than deleted public methods (in all the cases, a small effect size is observed)–see Table 3.

Summarizing, the analysis performed to answer our RQ_2 highlighted that, on one side, *changes performed on methods' body are those triggering more questions from developers. This is likely due to the fact that developers tend to get confused if method's behavior is different from that one expected by their experience on previous Android releases.* On the other side, *added and deleted public methods are the changes generating fewer questions.*

3.3 RQ3: What Kind of API Changes Trigger More Relevant Questions and Involve More Experienced Developers?

While results for RQ_2 provided us with a ranking of the types of changes triggering more questions by developers, in RQ_3 we analyzed the relevance of questions posed by developers for the same type of changes investigated in RQ_2.

Table 3: Questions per methods underwent to different types of changes between two Android releases: Mann-Whitney test (adj. p-value) and Cliff's Delta (d). Just significant results are reported.

Test	adj. p-value	d
Method body *vs* PM added	<0.01	-0.30 (Medium)
Method body *vs* PM deleted	<0.01	-0.30 (Medium)
Method body *vs* PM signature	<0.01	-0.42 (Medium)
PM signature *vs* PM added	<0.01	0.23 (Small)
PM signature *vs* PM deleted	<0.01	0.23 (Small)
PM added *vs* PM deleted	<0.01	0.07 (Small)
PM exceptions *vs* PM deleted	<0.01	-0.20 (Small)

Table 4: Relevance of the asked questions per methods that underwent through different types of changes between two Android releases: Mann-Whitney test (adj. p-value) and Cliff's Delta (d). Just significant results are reported.

Number of answers

Test	adj. p-value	d
PM deleted *vs* Method body	<0.01	-0.91 (Large)
PM deleted *vs* PM signature	<0.01	-1.23 (Large)
PM deleted *vs* PM added	<0.01	-1.18 (Large)

Question score

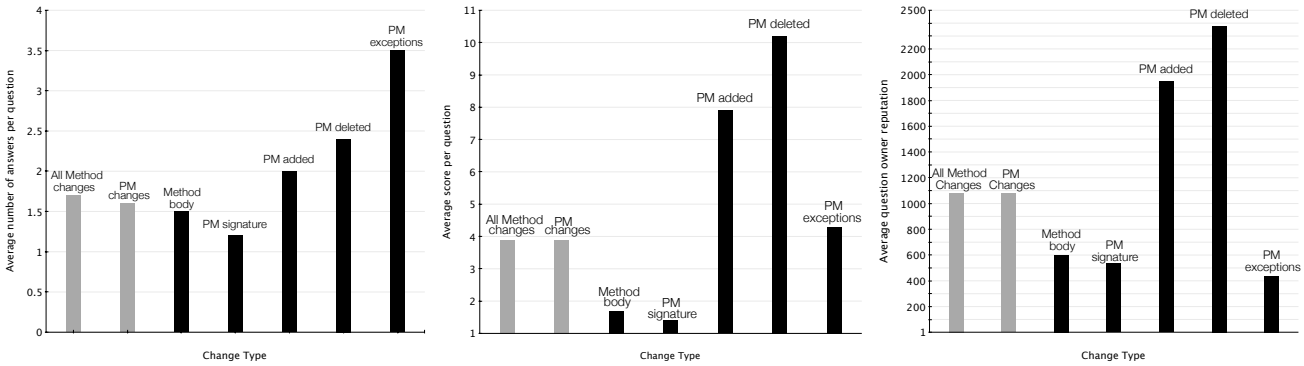
Test	adj. p-value	d
PM deleted *vs* Method body	<0.01	-5.46 (Large)
PM deleted *vs* PM signature	<0.01	-5.07 (Large)
PM deleted *vs* PM added	<0.01	-2.59 (Large)
PM added *vs* Method body	0.02	-3.27 (Large)

Question owner reputation

Test	adj. p-value	d
PM added *vs* Method body	<0.01	-3.54 (Large)
PM added *vs* PM signature	<0.01	-2.77 (Large)
PM deleted *vs* Method body	<0.01	-1.01 (Large)
PM deleted *vs* PM signature	<0.01	-5.47 (Large)
PM deleted *vs* PM added	<0.01	-1.52 (Large)

Figure 5 shows, for the categories of methods that undergo different types of changes: (a) the average number of answers per question, (b) the average score per question, and (c) the average question owner reputation. Table 4 reports the results of the Mann-Whitney test and the Cliff's d when comparing the distribution of number of answers, question scores, and question owner reputation, for questions related to the different types of changes considered in our study. Also in this case, we compared the distributions of each type of change with all the others and report statistically significant results in Table 4.

Figure 5(a) shows how changes to PM exceptions generate more discussion between developers, resulting in an average number of answers per question of 3.5 as compared, for example, to the 1.2 (+192%) of changes applied to PM signatures. However, these differences are not statistically significant. Also, questions about added PM (2.0 answers) and deleted PM (2.4) generally attract more answers than other types of changes, such as those performed to method body (1.5) and PM signature (1.2). Results shown in Table 4 shows that questions related to deleted public methods are the only ones receiving a statistically significant higher number of answers than other types of changes (effect size always *large*).



(a) (b) (c)

Figure 5: API changes triggering more relevant questions from users

Note that, on the basis of the results of our RQ_2, this is the type of change generating fewer questions from developers. However, it seems to be the one for which more developers join the discussion to solve the posed problem.

Concerning the average score per question—see Figure 5(b)—questions formulated about added and removed PM have a much higher score than those related to the other types of changes. For instance, questions about deleted PM exhibit an average score of 10.2, against the 1.7 of changes in methods' body (6 times higher) an 1.4 of changes in PM signatures (7 times higher). Note that higher average score for a question implies more interest in the SO community for that question. Again, results of statistical tests (Table 4) confirm that deleted PM are the one triggering significantly more relevant questions. In fact, we observed statistically significant higher question scores for questions related to deleted PM as compared to other types of changes (with always a *large* effect size). The results are statistically significant also when comparing the scores of questions related to added PM with those of questions related to changes in the method body.

Finally, as for the results achieved on the average question owner reputation shown in Figure 5(c), it is clear that questions about added PM and especially deleted PM are posted by more experienced developers and thus, likely represent relevant problems to solve as compared to the other types of changes. In particular, the average reputation of developers asking questions about deleted public methods is 2,379 as compared to the 602 (4 times higher) than questions about changes in the methods' body, 535 (4.4 times higher) than questions about changes in PM signatures, and 437 (5.4 times higher) than questions about changes in exceptions thrown by PM. A similar trend is also observed for questions related to added PM, having an average question owner reputation of 1,948. Statistical tests confirm that questions related to deleted PM are the ones posted by most experienced developers, followed by questions having as object newly added API methods.

Summarizing, the results for RQ_3 demonstrate that *deleting PM from APIs is the type of change that generates most relevant and discussed questions in the developer community.* In fact, questions related to this change are (i) more discussed and of major interest for the community and (ii) posted by more experienced developers.

4. THREATS TO VALIDITY

A threat to *construct validity* concerns the study design decision about using the number of answers, the question's owner reputation, and question's scores as proxies for relevant questions in RQ_3. Assumptions in this case are that relevant questions are posted by developers with high reputation, and also are highly scored because they might be helpful for the developer community as suggested by Nasehi et al. [31]. However, further studies are required to validate with real SO contributors whether or not those measures represent problematic questions. A source of imprecision of our approach could be in the way we map paragraphs to methods, when a discussion is related to multiple classes having methods with the same name. The used mapping approach, validated by Panichella *et al.* [32], exhibited a precision between 79% and 87%.

A threat for the *internal validity* of the study is the number of Type 3 and Type 4 links (i.e., matches between names of classes and words in a question's title and body) that we found. Those types of links can potentially be false positives because non-compound names of Android classes also represent concepts commonly used by people and are unrelated to Android code elements. However, our manual validation with a sample of 100 links between SO questions and Android methods suggests that only around 20% or fewer of the links identified by our approach are false positives. In the set of our methods-questions links, the exploited sample of 100 questions ensures a confidence interval of 9.8% with a confidence level of 95%.

For what concerns the relationship between the data treatment and the results (i.e., threats to *conclusion validity*), our conclusions are supported by using non-parametric statistics (p-values were properly adjusted when multiple comparisons were performed). Moreover, the practical relevance of the observed differences is highlighted by effect size measures.

Our findings are based on the Android APIs and Android-related questions in SO. The release history of the Android APIs, in addition to the model for developing Android apps, are threats to *external validity*, because particularities such as release frequency, commercial success, and high dependency of the apps on the API. Therefore, our results may not necessarily generalize to the other APIs, frameworks, and developer communities.

5. RELATED WORK

This section discusses related work about (i) issues when using APIs, and (ii) linking documentation to source code.

5.1 Issues when using APIs

Difficulties when using APIs are related to learning issues, effects of API evolution (e.g., API instability and backward compatibility), and problematic features. Robillard and DeLine [36] found that the top two obstacles for using APIs are inadequate API documentation and the API structure. The findings of Robbes *et al.* [35] and Businge *et al.* [12] also provide some evidence about the usage of API guidelines. Robbes *et al.* [35] present a study on the impact of deprecated API classes and methods in a Smalltalk ecosystem. According to [35] deprecation instructions are not always useful, because they are absent or unclear, or developers decided not to take them into account; in addition, half of the developers (30) surveyed by Robbes *et al.* reported that they do not know about the existence of the guidelines and only one of the developers answered that she always follows the guidelines.

Hou and Li [20] categorized the questions that developers ask when using the Java Swing API. Usually, developers look for advice when implementing a solution based on unfamiliar or undocumented APIs, when using a wrong API-based solution, or when using an API-based solution incorrectly.

In the particular case of the Android ecosystem, Linares-Vásquez *et al.* [24] analyzed the impact of API change- and bug-proneness on the success of android apps, and McDonnell *et al.*[27] investigated the relationship between unstable APIs and their adoption in client code. Main conclusions are that change- and bug-proneness of the Android API is a threat to success of mobile apps [24], and developers avoid frequent upgrades to change-prone (unstable) APIs [27]. Our results are complimentary to both studies in the sense that provide evidence of the reaction of developers to changes in Android APIs from a different perspective. In particular, we found that developers in the SO community react to changes in Android APIs. Therefore, we guess that the slow adoption of API changes could be also explained by the time it takes to developers find solutions to the effects of the API changes, by using Q&A systems such as SO.

5.2 Linking documentation to source code

Linking informal documentation—e.g., emails, bug reports, forums—to source code elements is usually achieved by extracting predefined code elements from textual descriptions [8, 14, 32, 33, 44]. Other techniques that slightly differ from typical textual-analysis-based approaches are the ones presented by Bettenburg *et al.* [11], Rigby and Robillard [34], and [38]. Table 5 lists these techniques, reporting their main characteristics and some performance indicators of the underlying study.

The coverage of APIs in SO discussions was analyzed by Parnin *et al.* [33]. To measure the coverage, the authors identified traceability links between SO threads (i.e., questions and answers) and API classes by using heuristics. Those heuristics are based on exact matching of classes names with words in textual elements (i.e., question title and body, snippets, HTML anchors), but, exact matching with words on the title and body leads to false positives. In our study, we identified the same types of links as Parnin *et al.* [33], however, we pipelined the links detectors instead of using them independently. Dagenais and Robillard [14] used meta-models to link developer documentation and support channels to fine-grain code elements (i.e., classes, attributes, variables, methods). Instead of using exact matching of classes names as Parnin *et al.* [33], Dagenais and Robillard [14] recover traceability links using code-like terms in the documentation and augmented contexts. After the links are traced, a pipeline of filtering heuristics are applied to remove potential code elements that can be false positives because of different levels of ambiguity (declaration, overload, external reference, language). In our study, we also used a pipeline of steps, but to reduce the chance of getting false positives that are introduced by the exact matching. Moreover, we traced links only at class and method granularity levels.

Panichella *et al.* [32] present an automatic approach for mining method descriptions in informal documentation. Candidate method descriptions in emails or bug reports are matched against API methods using a three steps approach: (i) identifying fully-qualified class names or class file names in the text of emails/bug reports, (ii) extracting paragraphs from the text using preprocessing techniques, and (iii) tracing such paragraphs to specific methods of the classes, computing the textual similarity between the paragraphs and method signatures. We used the same heuristics for tracing descriptive text fragments to API methods, however we considered code fragments in SO questions as descriptive text fragments. Moreover, we linked descriptive fragments (SO questions) to API classes as Parnin *et al.* [33].

Topic modeling was used by Wang and Godfrey [44] on SO, to identify API usage obstacles in iOS and Android developer questions. As the first step in the process, the authors traced SO questions to API classes by collecting all SO tags that can be mapped to API classes in the official reference of the iOS and Android. Therefore, this is another type of links that can be considered in future work, in addition to the ones that we considered in this study.

Rigby and Robillard [34] proposed an approach that does not require a term index, and conjectured that not all elements are equally essential to a document. The target list of code-like terms (term index) is extracted from the documents using an island parser [30] and validated using contexts as done by Dagenais and Robillard [14]; then, the documents are parsed again to identify terms that match code elements in the term index. In our case, we used the list of classes and methods in the Android APIs as the term index, because we were interested in a particular API. However, by using the approach by Rigby and Robillard [34] we could potentially improve our results because some SO questions can refer to the Android APIs without using specific names of API elements.

Zhang and Hou [45] identified problematic features in the Java Swing API by analyzing negative sentiment sentences in forum threads. Nouns, verbs, and adjectives in the official Java Swing tutorial were extracted to represent the features and build a dictionary. Then, negative sentiment sentences containing terms in the dictionary were identified as indicators of potential problems in API design or usage. In our study we did not use sentiment analysis because we were not looking for specific reactions of developers. However, looking for negative/positive sentences related to API elements could be used to identify problematic changes in APIs.

Subramanian *et al.* [38] proposed a technique for linking types and methods in code snippets to API documentation. Incomplete ASTs extracted from the snippets are analyzed

Table 5: Techniques for linking informal documentation to code elements. The table lists the code elements used for each technique (Classes, Packages, Annotations, Types, Methods, Fields, Code Fragments, and specifies whether the approach uses a dictionary (term index).

Technique	Elements	Dictionary	Validation	Precision
Bacchelli *et al.* [8]	C	Yes	Mailing lists of six different software systems	23.33% (using exact matching), 53.27% and 32.83% (using regular expressions), 23.33% (using VSM), and 42.50% (using LSI)
Parnin *et al.* [33]	C	Yes	Questions and answers about Java, Android and GWT in SO, but without validation	–
Dagenais and Robillard [14]	P,A,T,M,F	Yes	Documentation, and support channels of four open source systems	95.9%
Panichella *et al.* [32]	M	Yes	Bug reports from Eclipse, and emails and bug reports from Apache Lucene	79% (Eclipse), 87% (Lucene)
Betternburg *et al.* [11]	CF	No	Bug reports from Eclipse	70.13%
Wang and Godfrey [44]	C	Yes	iOS- and Android-related SO posts, but without validation	–
Rigby and Robillard [34]	P,A,T,M,F	No	Questions and answers about HttpClient, Hibernate and Android in SO	96% (HttpClient), 91% (Hibernate), 90% (Android)
Subramanian *et al.* [38]	T+M in CF	Yes	Java and Javascript snippets from SO and GitHub	98%(Java),97%(Javascript)
Our study	**M**	**Yes**	**Android-related questions in SO**	**84.5 %**

iteratively until all relevant nodes (i.e., some elements involved in declarations, invocations, and assignments) are associated with a single fully qualified name, or the iteration has converged (i.e., there is not improvement on the results for any element). The qualified names belonging to the APIs are identified by using an oracle (a.k.a., dictionary) for Java and Javascript. The validation shows that the approach outperforms previous work, however it only considers snippets reducing the ambiguities introduced by non-snippet text in the titles and bodies of SO discussions.

6. CONCLUSION AND FUTURE WORK

In this paper we reported a study investigating a relationship between API changes in Android SDK and developers' reaction to those changes, motivated by three facts: (i) the increasing number of questions about Android in discussion forums, (ii) the speed of release of new versions of the Android APIs with considerable number of changes, and (iii) the high dependency of Android apps on the corresponding APIs. As a proxy of the developer community, we used the questions posted in SO and tagged to Android-related labels between July 2008 and August 2012; and as a proxy of the changes, we analyzed 35,703 developers' commits performed in a period going from September 2009 to January 2013 for a total of 370,180 method changes.

Our findings suggest that developers in the SO community react to changes in Android APIs, in particular when the body of API methods is modified. In addition, deleting public methods from APIs is a trigger for questions that are (i) discussed more and of major interest for the community, and (ii) posted by more experienced developers. Therefore, new strategies should be used by API designers/developers to notify API consumers when elements are deprecated in the APIs, or when sensitive changes are introduced in new releases.

Future work will be devoted to identifying the resilience of API classes and methods when tracing links from discussions to source code as in [34]. Moreover, to reduce the impact of sensitive changes in the Android API, recommender systems such as the one proposed in [19] should be developed to recommend change events to a specific developer or notify the community (e.g., Android apps developers) when new changes are released with the APIs, in particular when the changes modify the behavior of methods, or remove pubic methods.

7. ACKNOWLEDGEMENTS

We would like to thank Carlos Bernal-Cárdenas for his help in the manual validation of links between SO questions and Android API methods. We also acknowledge Bogdan Dit for his help in verifying the results and proofreading the paper. This work is supported in part by the NSF CCF-1016868, NSF CCF-1218129, and NSF CAREER CCF-1253837 grants. Gabriele Bavota and Massimiliano Di Penta are partially funded by the EU FP7-ICT-2011-8 project Markos, contract no. 317743. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

8. REFERENCES

[1] Android Git repositories. `https://android.googlesource.com`.

[2] Is there any way to run Python on Android?.

http://stackoverflow.com/questions/101754, September 2008.

[3] Stop EditText from gaining focus at Activity startup. http://stackoverflow.com/questions/1555109, October 2009.

[4] Why is the Android emulator so slow?. http://stackoverflow.com/questions/1554099, October 2009.

[5] Android BitmapDrawable Constructor Undefined. http://stackoverflow.com/questions/4904660, February 2011.

[6] Android how to catch search key. http://stackoverflow.com/questions/12354725, September 2012.

[7] A. Bacchelli. Mining challenge 2013: Stack overflow. In *The 10th Working Conference on Mining Software Repositories*, 2013.

[8] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 375–384. ACM, 2010.

[9] A. Barua, S. Thomas, and A. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering (EMSE)*, 2012.

[10] G. Bavota, A. Ciemniewska, I. Chulani, A. De Nigro, M. Di Penta, D. Galletti, R. Galoppini, T. Gordon, P. Kedziora, I. Lener, F. Torelli, R. Pratola, J. Pukacki, Y. Rebahi, and S. Garcia Villalonga. The MARKet for Open Source: An intelligent virtual open source marketplace. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14)*, pages 399–402. 2014.

[11] N. Bettenburg, S. Thomas, and A. Hassan. Using fuzzy code search to link code fragments in discussions to source code. In *16th European Conference on Software Maintenance and Reengineering (CSMR'12)*, pages 319–328, 2012.

[12] J. Businge, A. Serebrenik, and M. van den Brand. Analyzing the eclipse API usage: Putting the developer in the loop. In *17th European Conference on Software Maintenance and Reengineering (CSMR'13)*, pages 37–46, 2013.

[13] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 3rd edition edition, 1998.

[14] B. Dagenais and M. Robillard. Recovering traceability links between an API and its learning resources. In *34th International Conference on Software Engineering (ICSE'12)*, pages 47–57, 2012.

[15] Google. Android Developer Reference Guide. http://developer.android.com/reference/.

[16] Google. Reading and Writing Logs. [**?**].

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

[18] S. Holm. A simple sequentially rejective Bonferroni test procedure. *Scandinavian Journal on Statistics*, 6:65–70, 1979.

[19] R. Holmes and R. Walker. Customized awareness: Recommending relevant external change events. In *32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*, pages 465–474, 2010.

[20] D. Hou and L. Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *IEEE 19th International Conference on Program Comprehension (ICPC'11)*, pages 91–100, 2011.

[21] S. E. inc. Stack Overflow- http://stackoverflow.com.

[22] S. E. inc. Stack Overflow FAQ- http://stackoverflow.com/faq.

[23] H. Li, Z. Xing, X. Peng, and W. Zhao. What help do developers seek, when and how? In *20th Working Conference on Reverse Engineering (WCRE'13)*, pages 142–152, 2013.

[24] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. 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, pages 477–487. ACM, 2013.

[25] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk. An exploratory analysis of mobile development issues using stack overflow. In *10th IEEE Working Conference on Mining Software Repositories (MSR'13)*, pages 93–96, 2013.

[26] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest Q&A site in the west. In 2857-2866, editor, *SIGCHI Conference on Human Factors in Computing Systems (CHI'11)*, 2011.

[27] T. McDonnell, B. Ray, and M. Kim. An empirical study of API stability and adoption in the Android ecosystem. In *IEEE International Conference on Software Maintenance (ICSM'13)*, pages 70–79, 2013.

[28] I. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. Hassan. A large scale empirical study on software reuse in mobile apps. *IEEE Software Special Issue on Next Generation Mobile Computing*, 2013.

[29] I. Mojica Ruiz, M. Nagappan, B. Adams, and A. Hassan. Understanding reuse in the Android market. In *20th IEEE International Conference on Program Comprehension (ICPC'12)*, pages 113–122, 2012.

[30] L. Moonen. Generating robust parsers using island grammars. In *8th IEEE Working Conference on Reverse Engineering (WCRE)*, pages 13–22, 2001.

[31] S. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming Q&A in stackoverflow. In *28th IEEE International Conference on Software Maintenance (ICSM'12)*, page 25.34, 2012.

[32] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In *IEEE 20th International Conference on Program Comprehension (ICPC'12)*, pages 63–72, 2012.

[33] 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, 2012.

[34] P. Rigby and M. Robillard. Discovering essential code elements in informal documentation. In *35th*

International Conference on Software Engineering (ICSE'13), 2013.

[35] R. Robbes, M. Lungu, and D. Rothlisberger. How do developers react to API deprecation?: the case of a Smalltalk ecosystem. In *ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE'12)*, 2012.

[36] M. Robillard and R. DeLine. A field study of API learning obstacles. *Empirical Software Engineering (EMSE)*, 16:703–732, 2012.

[37] C. Rupakheti and H. Daquing. Evaluating forum discussions to inform the design of an API critic. In *IEEE 20th International Conference on Program Comprehension (ICPC'12)*, pages 53–62, 2012.

[38] S. Subramanian, L. Inozemtseva, , and R. Holmes. Live API documentation. In *36th International Conference on Software Engineering (ICSE'14)*, page to appear, 2014.

[39] M. Syer, M. Nagappan, B. Adams, and A. Hassan. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source android apps. In *CASCON 2013*, 2013.

[40] M. D. Syer, B. Adams, Y. Zou, and A. E. Hassan. Exploring the development of micro-apps: A case study on the BlackBerry and Android platforms. In *IEEE 11th International Working Conference on Source Code Analysis and Manipulation (SCAM'11)*, pages 55–64, 2011.

[41] C. Treude and M.-A. Storey. Effective communication of software development knowledge through community portals. In *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE'11)*, pages 91–101, 2011.

[42] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov. How social q&a sites are changing knowledge sharing in open source software communities. In *17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW 2014)*, 2014.

[43] S. Wang, D. Lo, and L. Jiang. An empirical study on developer interactions in stackoverflow. In *28th Annual ACM Symposium on Applied Computing*, pages 1019–1024, 2013.

[44] W. Wang and M. Godfrey. Detecting API usage obstacles: A study of iOS and Android developer questions. In *10th IEEE Working Conference on Mining Software Repositories (MSR'13)*, 2013.

[45] Y. Zhang and D. Hou. Extracting problematic API features from forum discussions. In *IEEE 21th International Conference on Program Comprehension (ICPC'13)*, 2013.