Using and Asking: APIs Used in the Android Market and Asked About in StackOverflow

David Kavaler*, Daryl Posnett*, Clint Gibler
Hao Chen, Premkumar Devanbu**, and Vladimir Filkov**

Univ. of California, Davis Davis, CA 95616, USA,
{dmkavaler,dpposnett,cdgibler,chen}@ucdavis.edu,
{devanbu,filkov}@cs.ucdavs.edu

Abstract. Programming is knowledge intensive. While it is well understood that programmers spend lots of time looking for information, with few exceptions, there is a significant lack of data on what information they seek, and why. Modern platforms, like Android, comprise complex APIs that often perplex programmers. We ask: which elements are confusing, and why? Increasingly, when programmers need answers, they turn to StackOverflow. This provides a novel opportunity. There are a vast number of applications for Android devices, which can be readily analyzed, and many traces of interactions on StackOverflow. These provide a complementary perspective on *using* and *asking*, and allow the two phenomena to be studied together. How does the market demand for the USE of an API drive the market for *knowledge* about it? Here, we analyze data from Android applications and StackOverflow together, to find out what it is that programmers want to know and why.

1 Introduction

The cottage industry of Android application development is booming, and the number of applications, colloquially apps, is growing exponentially [5]. While writing good apps is contingent upon solid understanding of the Android OS, many app writers begin programming with little skill and learn on the fly. Naturally, the Android components include self-documentation, which aids amateur developers in their task. However, such documentation often needs supplementing with other sources. Amateur developers supplement their learning by reading online documentation and participating in knowledge exchanges in online question and answer communities, like StackOverflow. One recent field study [14] reports that StackOverflow has more examples, better coverage, and is visited by real developers more often than the online documentation. In this distributed, or crowdsourced, education system, people ask questions and others answer them, usually for no remuneration. Here we are interested in the relationship between the amount of API *use*, i.e. API popularity, and the *questions asked about* APIs, i.e. help requested for those APIs. Some APIs attract more questions and recedive more answers from the community than others. As APIs see more usage, more questions are asked about them,

* These authors contributed equally to this work
** These authors contributed equally to this work

so popularity plays a role. But does API complexity drive a greater nimber of questions? Furthermore, does the amount of available documentation influence the number of questions asked about an API? Such questions have previiously been difficult to approach quantitatively due to the many APIs, and because a large number of applications are required to ensure sufficient statistical variation in API use.

However, it is now possible to mine *app marketplaces*, with hundreds of thousands of applications; many of them are free, so it is feasible to download a large number for analysis. Furthermore, Android byte code is fairly high-level, so it is possible to determine exactly which APIs are used, and to what extent. This data, together with the readily-available data from StackOverflow, provides a a fresh opportunity to relate the degree to which programmers *use* APIs to the degree to which they *ask* about them. This data provides a starting point to define the popularity of an API, from the app marketplace, and the amount of help requested on StackOverflow for that API. Our contributions in this paper are as follows:

- We show that careful data gathering and cleaning will produce a data set of linked Android Market data and StackOverflow traces; this data set is of independent interest to both software engineers and documentation writers;
- We show that there is a non-trivial, sub-linear relationship between class popularity in Android apps and the requests for their documentation in StackOverflow, i.e. the number of questions on StackOverflow about a class lags its usage in the apps.
- We find a complex relationship between the size of a class, its internal documentation and the StackOverflow documentation supply and demand. The size of a class and the *total* amount of available documentation for a class both have sizable and significant impact on the number of questions, *i.e.* larger classes and more documented classes are associated with more questions. The situation is reversed for answers, although much smaller in magnitude.
- We give a case study examining several classes in detail to gain insights in the relationship between usage and documentation requests. A secondary goal of the case study is to understand the relationship between *mentions* of a class in the text of a question, and *salience* of the class to the meaning of the question.

2 Related Work

As APIs and libraries increase in complexity there has been an increased focus on leveraging, combining, and understanding multiple sources of developer documentation. Dagenais and Robillard study the decisions open source developers make while creating developer documentation for frameworks and libraries [9]. They find that developer choices to document their own code during the code creation process can not only affect the quality of documentation, but, in fact, the quality of code itself. Their results are relevant here as we are describing a relationship between internal API code, and the discussions that developers have regarding that code.

Following the public release of the StackOverflow data, researchers have studied how developers gain knowledge and expertise through online question and answer sites. Research in this area has studied the success of Q and A sites, the growth of askers and

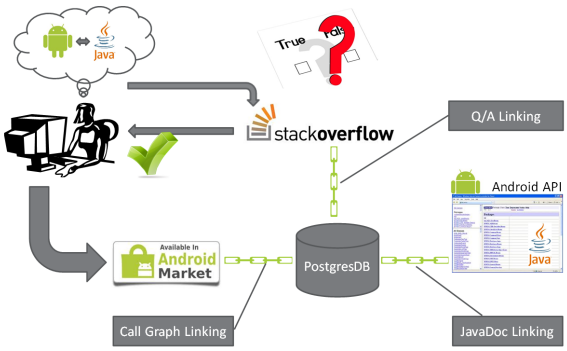

Fig. 1: Programming involves consulting documentation, internal, *viz.*, embedded in the code, and external, *e.g.*, in StackOverflow. Shown here are the data sources and their inter-connections.

answerers, and how developers use online resources. Our focuse is on the last category. Treude *et al.* evaluated the nature of questions asked on StackOverflow with respect to programming language, question tags, and question types determined by manual coding of a sample of questions. They concluded that the community answers review, conceptual, and how-to questions more frequently than any other type, with review questions achieving a 92% accepted answer rate [21]. Posnett *et al.* studied the nature of expertise among Stack Exchange participants and found that participation in online Q/A sites does not necessarily lead to increased expertise as measured by peer scoring [16]. These two results may be viewed as mutually supportive suggesting that programmers answer questions about what they know. However, in what way does existing documentation impact the tendency to ask or answer questions about an API? There has been some previous work on this question. Parnin and Treude study availability of documentation across APIs within StackOverflow and find that it is very uneven, some APIs are well covered, while others receive very little attention [20]. Jiau and Yang address this concern and assert that more obscure API components benefit from a trickle down affect of similar API components [11].

The work most similar to ours is recent work by Parnin *et al.* who also evaluated the degree to which specific APIs are discussed within the community [15]. Our work differs from theirs in two ways. First, we use a larger pool of free apps that contain both open and closed source code, and second, we gather metrics of existing documentation and study these disparate data sources in a multiple regression context. This allows us to measure impact of some variables on a response while controlling for other covariates.

3 Theory

Writing software is a knowledge-intensive activity. Understanding the structure and purpose of large software systems well requires significant effort; this has been an active area of research [7,13]. The creation and use of software knowledge impacts developer

productivity, such as the upfront time cost of writing documentation, updating documentation as the software evolves, and the time spent in informal explanations between colleagues. This knowledge barrier to entry arguably leads to the "thin spread of knowledge", where relatively few developers [8] in large teams have deep knowledge about the system under development.

A recurring challenge in software engineering is the difficulty of studying the "demand" for program understanding. Which components of the system are most frequently asked about? What influences the demand for knowledge about these components? Generally, previous studies of program understanding have employed human studies that have yielded valuable insights [12,18]. Given the cost of doing such studies however, it is difficult to get large-sample data that allows us to distinguish how the programmer demand for information varies from element to element.

The advent of on-line forums like StackOverflow, and powerful search engines, have changed the way programmers work. It has by now become a routine experience to probe a search engine with a compiler or run-time error and finding a remedy on StackOverflow. Such experiences have left numerous traces for researchers to mine and study. Thus, it is now possible to investigate programmers' quest for knowledge, and the factors that influence their various quests, using a large-sample mining methodology over StackOverflow. In this work, we focus on the relationship between Android API-focused questions on StackOverflow and the actual usage of those APIs in real world, published apps. This allows us a novel opportunity to examine how a large body of developers use an API in practice, and how this affects the frequency of mention of these APIs in StackOverflow. Figure 1 illustrates, in a simplified fashion, the process of Android app programming and documentation mining.

We expect a relationship between API usage and the frequency of questions on StackOverflow (SO), as the more developers use a given API, the more likely there are to be associated questions. We expect a relationship between the amount of documentation for a class and the number of questions about it, as classes with less documentation may be harder for developers to understand. One must control for the usage of the API, however, as highly used APIs will likely be the source of many questions, regardless of documentation. Finally, after controlling for usage, we expect fewer answerers for questions about classes with less documentation. The intuition is that fewer developers will have expertise on classes with less documentation, so questions on those classes will have a smaller set of answerers. We study the following specific questions.

Research Question 1: What kind of relationship exists between uses of classes in free apps and the mentions of those classes in SO questions and answers?

Research Question 2: Do the properties of the existing Java documentation influence the relationship between SO and usage? In other words, how does the volume of documentation relate to the number of questions in SO?

Research Question 3: How do the number of available answers depend upon the USE of a class, and the volume of available documentation?

We note several unexpected issues when attributing questions to API classes, including stack traces, code blocks, poorly formatted questions and answers (posts), incorrect user-defined tags, multi-word class name splitting, highly coupled classes, and base classes with many subclasses.

We also present a case study of the relationship between usage and demand for documentation for specific classes at different ratios of usage to demand. In this case study, we look at three clusters of API classes: those that have equal numbers of questions and uses, those with more uses than questions and vice versa. In addition, we look at random points on the major trend line. We randomly sample ten questions for each class in each group, and draw conclusions based on our findings.

4 Data and Methodology

Data. Android class invocation data was taken from a pool of $109,273$ free applications. The majority of the applications in our data set were pulled from the official Android market as well as several third-party English and Chinese markets.[1] 82% of the applications were from English markets with 66% of those coming from the official Android market. 18% of the applications were taken from Chinese markets.[2] Using data from multiple markets diversifies our data set, as different groups of developers may post to different markets. Thus our data is robust against bias that may be introduced due to application market politics.

StackOverflow provides data dumps, under a Creative Commons license, sporadically on their blog [19]. Our data is based on a data dump from August 2011. We imported this data dump into our database and used each post's content and title to perform class name matching. We also used the *tags* column in our data gathering process outlined in Section 4.

The Android documentation and lines of code information were extracted by running *Javadoc* with a custom Doclet on Android source code [10]. The StackOverflow data includes questions and answers from Android releases up to version $3.2.2$ (as Android version $3.2.2$ was released in August 2011).

However, not all of Android version $3.x$ source code is available through the repository. Android $3.x$ was intended for use on tablets, and Google felt this code was incomplete and didn't want developers building on the source [3]. To address this issue, whenever possible, we took documentation and lines of code data from Android $2.3.5$ source code; otherwise, we took data from Android $4.0.1$ source code. This resulted in 90% of our documentation and lines of code data coming from Android $2.3.5$. We believe this is an acceptable compromise as the number of new classes between versions is minimal in comparison to the total number of classes, reducing the influence that this problem has on data gathered from our large data set. We included deprecated classes in our data set because although the use of deprecated classes is not encouraged by API developers, users still write programs utilizing them, and thus, ask questions about them.

[1] The applications were acquired between December 2011 and October 2012.

[2] Due to changes in the crawling process these are rough estimates.

Fig. 2: An example of a question with multiple links. This also illustrates an issue with multi-word class names being split in natural language, not generating a link.

Linking Methodology. To gather our data, we used a process adopted from Parnin [15]. In our case, a *link* is a connection between a StackOverflow question and an Android class. The types of links are *Tag links*: A class name match occurring in the tags section of a StackOverflow question, *Word links*: A class name match occurring directly in the natural language text of a question, and *Href markup links*: A class name match enclosed by HTML `<a></a>` tags. For our purposes, all link types were treated the same as shown in Figure 2 (*i.e.* we union all links and treat them without priority).

Empty string word boundaries were used to isolate class names within question bodies. Negative lookahead (*viz.*, making sure a regex pattern is *not* followed by another specified regex pattern) was used to deal with problems in our question parsing with inner class definitions. Without negative lookahead, a search for a class such as *SharedPreferences* would also result in an incorrect match for *SharedPreferences.Editor*. All parsing was done in a case insensitive manner.

A single-word class is defined by camel case, *e.g. Activity* is a single word class, whereas *ImageView*, is a two word class. For single-word classes, we include every type of link except *word links* as single-word classes are often commonly used words that turn up a large number of false positives. However, in contrast with Parnin's method, we did not consider any class name matches appearing within `<code></code>` tags. Novice users often post complete code blocks in with their questions; this leads to frequent confounding mentions of popular classes, just because they happen to occur in many code blocks. In addition, we attempted to remove standard Android log output from the body of questions as users will sometimes post entire log traces of errors they encounter, occasionally generating false positives. Also, all links to subclasses are attributed to their parent, thus maintaining a consistent level of aggregation.

Stack Overflow. We parsed the online Android developer javadoc based documentation [1] to generate a list of android classes. We pulled all API calls that existed as of Android version 3.2.2, API level 13, as this was the latest Android version released as of the date of our StackOverflow data dump[3].

[3] Manual samples showed that most questions were of a more general form, so the use of classes from a single Android version is a reasonable choice.

Class names were used to generate links within StackOverflow questions as outlined above. We aggregated these links and recorded the class name that triggered each link along with the associated post *ID*. It is possible for a question to have multiple links, and each was recorded. Figure 2 clarifies this process. If a single question has multiple link types for the same class, only a single link for that class-question pair is recorded. We used the same method for gathering data from StackOverflow answers as for questions, with a change in generating *tag links*. In StackOverflow, there is no concept of an answer having a user-defined tag. To account for this, we linked each answer to the linked tag of the corresponding question.

This process results in data on a per-class basis that includes the number of questions, number of unique askers, number of answers, number of answer threads, and number of unique answerers.

Android. We converted Android application code (APK) to a human-readable format using APKTool [2] APKTool generates a folder from each APK file containing the Android Manifest file, resources, and a directory consisting of byte code files corresponding to original source files. We processed the extracted byte code files looking for function invocations (`invoke-virtual`, `invoke-super`, `invoke-direct`, `invoke-static`, and `invoke-interface`). We also gathered package names, class names, method names, return types, arguments, and number of calls per APK. We used this function invocation data to calculate the total number of calls for each class as well as the number of distinct APKs that call each class. This process is depicted in Figure 1. We joined the resulting APK data with our StackOverflow question links to generate the finalized data used to answer our research questions

4.1 Model Building

For several of our research questions we wanted to understand the impact of a set of *explanatory variables* on a *response* while controlling for known covariates that might influence the response. We use *negative binomial regression*, NBR, a generalized linear model used to model non-negative integer responses.

Of concern is that many of the Android classes do not have quesztions; in other words, our data contains many rows with a response value of zero which presents a challenge for the standard NBR model. *Zero inflated negative binomial regression* and *hurdle regression* are two methods designed to address this challenge by modeling the existence of excess zeros [4]. It is common practice to fit both types of models, along with the standard NBR model, and compare model fits to ascertain which structure is the most appropriate [4]. Since these models cannot be viewed as nested models we employ both *Akaike's Information Criterion*, (AIC), and Vuong's test of non-nested model fit to determine appropriateness of method [22]. We employ *log* transformations to stabilize the variance and improve model fit when appropriate [6]. Because explanatory variables are often highly correlated, we consider the VIF, *variance inflation factor*, of the set of predictors and compare against the generally accepted recommended maximum of 5 to 10 [6]. Finally, when we consider whether explanatory variables should remain in a model we use either a Chi-Square goodness of fit test or the aforementioned Vuong's

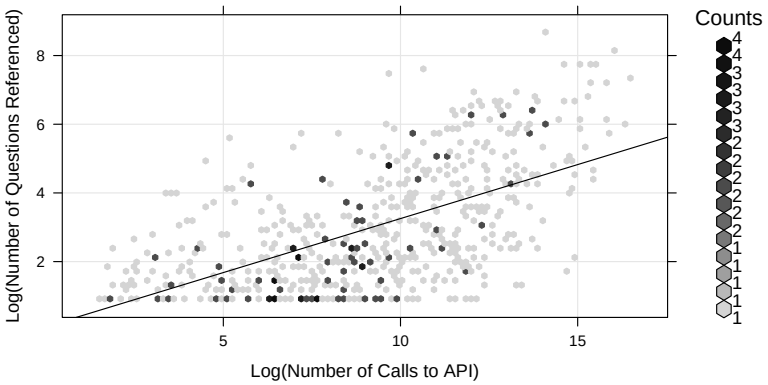

Fig. 3: A log-log scatter plot of how much each class is used vs how many questions are asked about it. The apparent linear trend with slope smaller than 1 (≈ 0.31) points to a saturation in the number of questions needed to document a class.

test of non-nested along with AIC to ensure that inclusion of the additional variables is justified [6].

5 Results and Discussion

Class Popularity vs Questions Asked About It. Each class in our data has an associated total number of calls from Android applications which we use as a proxy for its popularity and a total number of StackOverflow questions that mention the class. Figure 3 is log-log scale scatterplot showing the relationship between a class's popularity (x-coordinate) and the number of mentions (y-coordinate) in questions. The color indicates how many data points share a region on the plot. The distribution of the points, is suggestive of the relationship between popularity (or usage) of a class and the demand for external documentation about it. The plot shows a strong linear trend between $log\ x$ and $log\ y$, with a slope of 0.31, which translates into a sublinear dependence of y on x. As might be expected, a class' popularity, as measured by the number of APK calls, is strongly associated with the number of questions generated about it. However, it is noteworthy that the trend becomes sub-linear toward higher uses, and we conclude that:

Result 1: *On the average, the demand for class documentation from SO grows slower than class usage in the code.*

One possible interpretation for this phenomenon is that the more a class is used, or called, by APKs the more people are using it. And thus, the more people ask about it. Since SO make available, online, all questions asked and all answers to them, it is likely that after a certain number of questions, a person asking a question will usually find that

Table 1: The relationship between uses of an API call and discussion of that call in both questions and answers is mediated by both class and method documentation for the API call.

	Hurdle Model:		*Count Model:*	
	numquestions	numanswers	numquestions	numanswers
	(1)	(2)	(3)	(4)
log(numquestions+0.5)		1.662^{***}		1.021^{***}
		(0.140)		(0.015)
log(numcalls+0.5)	0.330^{***}	0.254^{***}	0.333^{***}	0.036^{***}
	(0.029)	(0.043)	(0.020)	(0.008)
log(sourceloc+0.5)	0.176^{*}	-0.139	0.241^{***}	-0.044^{*}
	(0.096)	(0.129)	(0.078)	(0.025)
log(numinnerclasses+0.5)	0.189	-0.121	0.440^{***}	0.074^{**}
	(0.174)	(0.249)	(0.142)	(0.032)
log(classdocloc+0.5)	0.328^{***}	0.302^{**}	0.141^{**}	-0.051^{***}
	(0.095)	(0.129)	(0.067)	(0.019)
log(avgmethdocloc+0.5)	-0.073	-0.020	0.007	0.079^{**}
	(0.126)	(0.177)	(0.102)	(0.031)
Constant	-2.677^{***}	-1.344^{**}	-1.264^{***}	-0.096
	(0.492)	(0.634)	(0.439)	(0.131)
Note:	*p<0.1; **p<0.05; ***p<0.01			

his/her question (or a very similar one) has already been asked and answered on SO. So, instead of asking it again, s/he will use the answers already provided.

API Self-Documentation vs StackOverflow Questions. To address this question we model a count of the number of questions that mention a particular class as a *response* against *explanatory variables* that measure the level of API documentation for that class while controlling for the usage and the size of the class. The API documentation for an Android API class was captured in two variables, the line count for the class documentation proper, as well as the mean line count of documentation for each method. To control for the usage of the class we included the number of calls to that class found in the APKs. To control for the size, we used the number of lines of code in the class. We *log* transformed all variables. In addition to stabilizing variance, the transformation induces a natural interpretation; a small increase to a short document is, on average, more likely to improve understanding than a small increase to a long document. In all cases, Vuong's test of non-nested model fit favored the transformed variant yielding a better fit to the data with p-value < 0.001. One cause of concern is that the mean of method documentation size is significantly rank (spearman) correlated with class documentation size (0.69 p=value < 0.001). The VIF, *variance inflation factor*, of these variables in a model, however, was well within bounds at < 1.35 (See Sec. 4.1).

We compared standard NBR models with their zero-inflated and hurdle counterparts. The standard NBR model slightly underestimated the zero count whereas the zero inflated models overestimated them. The hurdle models had the lowest AIC and

precisely predicted the observed number of zeros. In all cases Vuong's method preferred the hurdle models with p-value < 0.001.

The count component of the model is presented in the third and fourth columns of Table 1. As expected we can see a positive relationship between the usage of a class (numcalls) and the number of questions that mention that class. On the other hand, class documentation is a positive and significant predictor for number of questions, but is slightly negative, while still significant, for number of answers referencing that class. We conclude that class documentation is important in driving the number of questions that reference a class for askers. Interestingly, the average method documentation is not significant. We also note that both the size of the class and the number of inner classes have a positive effect on the number of questions referencing the class, a result not surprising having in mind that both of those predictors are a facet of complexity. Larger, potentially more complex classes are mentioned more often in questions.

The zero portion of the hurdle model is presented in the first two columns of Table 1. Here we model the factors that influence the ability of a class to cross the "hurdle" of obtaining even a single mention in a question (first column) or answer (second column). The coefficients represent the affect of that variable on overcoming the zero hurdle. Usage also has a positive effect in the model for both number of questions and number of answers. This means that the more popular a class, the greater the chance the class has of overcoming the hurdle of acquiring a reference in either question or an answer. Interestingly, total class documentation is highly significant and strongly positive in the zero portion of the model (left column), indicating that more documentation is associated with more questions. It can be argued that when faced with impersonal pages of documentation, one may be driven to asking a question on Stack Overflow where a personalized response is minutes away.

Class size, number of inner classes, and method documentation size, on the other hand, are not significant to overcoming the hurdle. This suggests that available class documentation, and not size/complexity, drive questions for at least one person. Taken together with the count model, the average size of method documentation drives the first mention to a large extent, but subsequent mentions are also dependent on class size. Positive coefficients suggest that greater volume of documentation tends to be associated with more questions, rather than fewer.

Result 2: *Class documentation size and class size have a relationship with question mentions on Stack Overflow, while controlling for the usage in free APKs.*

API self-documentation vs StackOverflow Answers. We asked the same questions with respect to the number of answers for a given question and the count model results are presented in the third and fourth columns of Table 1. In addition to the controls we used for questions, when modeling answers we also control for the number of questions. This is necessary, despite potential multicollinearity issues, as the number of answers will have a strong dependence on the number of questions. In fact, VIF values for this model were below 3, which is within acceptable limits [6]. We see that, while still positive, the effect of usage has a much lower impact on the number of answers than it

does on the number of questions. This suggests that people who answer questions are *motivated more by questions than by the usage of the classes*; SO is a "gamified" environment where people are motivated to accrue higher scores by providing more valuable answers to more questions. The dependence of answers on class documentation is also somewhat different; the coefficient for class documentation being very significant and slightly negative. Given that a class is already mentioned in an answer, greater class documentation is associated with fewer references in answers. Perhaps most interesting is that class size has no significant impact on answer references. Larger classes drive more questions, but not more answers, when controlling for the number of questions.

We turn now to the zero model presented in the first and second columns of Table 1. The coefficient for the number of questions is positive which implies that the more questions that mention a class, the more likely that at least one of the answers will also mention the class. The number of calls to a class has somewhat lower influence on the occurrence of an answer that mentions the class, than in the zero model for questions. It is a small difference, however, its lower value suggests (as above) that answerers will be somewhat less impacted by usage than simply by the existence of the question. As for questions, documentation size has a significant affect on the first answer, with a slightly smaller coefficient. Documentation size, therefore is important for answerers, when giving the first answer to questions about a given class. Perhaps this is an effect of the gamified platform underlying Stack Overflow, where the best response gets the answerer more points, thus being first to answer provides competitive advantage. The size of the class documentation, arguably, only makes the competition more interesting, *viz.* who can digest it and provide an answer first, within a short turnaround time.

Result 3: *Class documentation size has a strong positive relationship with getting the first answer which mentions it on Stack Overflow, controlling for usage in free APKs.*

5.1 Case Study: Class Usage and Their Mentions in Questions

To gain insight into our data, we sampled, and manually evaluated, classes falling in different regions of the plot in Figure 3. We identified three distinct clusters in Figure 3: one outlier group above the regression line, *top-left cluster*, one outlier group below the regression line, *bottom-right cluster*, and a group with extremal values in both, *top-right cluster*. To isolate these groups, we examined the plot of the data points and manually selected the clusters by range (nearby points). We then sampled classes corresponding to these data points and referenced the Android documenation for each class. After randomly sampling ten StackOverflow questions linked to each class name in each group, we categorized the classes in each cluster in an attempt to identify any systematic differences. We also manually marked each sampled question with a flag indicating whether or not the linked class name was the question's topic. The classes mentioned below are notable examples from this study.

Bottom Right Cluster. The bottom right region contains classes that are not explicitly created and called by users in Java code, so their function invocation counts are high, but

their question references are low. This group also includes self-explanatory classes that have relatively low barriers to entry or classes or calling conventions that are analogous to standard Java or other common platforms, reducing question references. This latter cateogory includes classes such as *Editable* and *Environment*. Programmers find such classes relatively straightforward, despite frequent use largely owing to their abstract nature and simple role.

Top Left Cluster. The top left region contains classes that are more application-specific, and less frequently used. Still, they have a fair number of question references as they may have confusing APIs and relatively small amounts of documentation for their complexity. For example, the class *MediaStore* deals with audio files and has 30 static public fields - a considerable number in comparison to the average of 4.98 public fields across other classes. Similarly, the class *Parcelable* deals with serialization. Efficient serialization is often a complex topic owing to pointers and recursive data. Finally, although the class *ContactsContract* is likely application-specific, it is not necessarily uncommon. A close investigation revealed that this class is a base class with numerous subclasses that provide for interaction with a device's contact information. This base class is seldom directly called when accessing contact information, explaining its low use count in our APK data. Interestingly, there are only a total of 22 question references combined for all 61 subclasses but 131 question references for the base class. This indicates that users will sometimes ask about the base class rather than the subclass which is actually used.

Top Right Cluster. The top right cluster contains classes that are necessary to use the Android platform and thus have high invocation counts. These classes also have many question references as they are where newcomers to the platform begin their understanding. To spare repetition, we only discuss one of the sampled classes as all classes in this cluster have a similar explanation for their existence in this group. The *Activity* class is the core of Android UI; each screen or page in an application extends the Activity class. Thus, the fact that it has a very high invocation count and has a large number of question references is not surprising.

6 Threats & Caveats

In this study we considered only one topic of questions, the Android platform. It is not clear whether conclusions generalize beyond this setting. Second, our study cumulates effects (questions, answers, uses, documentation lines, lines of code) over time, and thus can only point out associations, not causes and effects. A more detailed time-series study would be required to reveal, for example, whether more questions lead to more class documentation or vice versa; and likewise, whether average documentation lines per method is a cause or effect of more answers.

The remaining threats listed below arise from the fact that StackOverflow is unstructured natural language text, and extracting information therefrom is naturally a noisy process. We anticipate that emerging work in the area of salience and named entity recognition can be fruitfully used in this context, and help address the issues below.

We primarily consider the *mentions* of a class name in answers and questions. We have taken care to ensure that the mentions are actual mentions of a class name in the natural language text, and not merely in code included in the question or the answer. However, a *mention of a class* in an SO question or answer does not necessarily mean that the message is *about* that class; but it does mean that the message is relevant somehow to the class. Manual examination suggests that while a mention indicates some relevance, more often than not, a message that mentions a class isn't necessarily mostly "about" that class. There are some new, emerging tools that promise to find the most salient mentions [17].

Code fragments included within questions and answers present a quandary. Including complete code fragments (that are pasted into Q & A) in the analysis would be tantamount to sampling mentions of classes twice, once in the code itself, and once in the code pasted into the unstructured text. On the other hand, smaller fragments may include more relevant mentions of classes than full fragments. We plan to redo the analysis with smaller fragments included. A related issue is the posting of code without `<code></code>` tags; this is a minor concern, however, as it is usually corrected by moderators.

There are multiple sources of possible error in class name recognition within the unstructured StackOverlow text. First, some classes have common English language class names, *e.g. Security*. The word "security" is often mentioned in posts that have nothing to do with the *Security* class, thus giving rise to false positives. Secondly, users sometimes split multi-word class names with a space, *e.g.* referring to *TabWidget* as *Tab Widget* as shown in Figure 2. This can cause problems when attempting to match a question with a class name. Fortunately, this problem was not common in our case study.

6.1 Conclusion

We have studied the relationship between the *use* of Android API classes in over $100,000$ applications, and the *questions* and *answers* that mention these classes. We find that: a) questions do increase with use, although there is a non-linear saturation effect, suggesting that knowledge of the most popular classes do get internalized to some extent, or become easier to find; b) the *total* amount of available documentation for a class has no impact on the number of questions, but the size of classes does, suggesting that more complex classes increase the need for information; and c) that the situation is reversed for answers: the total amount of class documentation decreases the number of answers per question while class size has no effect, and the documentation per method increases the number of questions. To our knowledge this is one of the first studies clearly relating Android market place usage of APIs to the occurrence of these APIs in StackOverflow questions. We believe that this is a fruitful area of study, and we invite other colleagues in joining us to improve data cleaning, examine causes and effects via time-series analysis and so on. Clear findings on the causes of documentation demand, and the effects of good and bad documentation, can have a profound impact on the effective use of complex platforms like Android.

References

1. Android Documentation, http://developer.android.com/reference/classes.html
2. APK Tool, http://code.google.com/p/android-apktool
3. businessweek.com. http://www.businessweek.com/technology/content/mar2011/tc20110324_269784.html
4. A. C. Cameron and P. K. Trivedi. *Regression analysis of count data*, volume 30. Cambridge University Press, 1998.
5. CNN, http://edition.cnn.com/2013/03/05/business/global-apps-industry
6. J. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.
7. T. A. Corbi. Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.
8. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.
9. B. Dagenais and M. P. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of 18th ACM SIGSOFT international symposium on Foundations of Software Engineering*, p. 127–136. ACM, 2010.
10. Javadoc Documentation, http://docs.oracle.com/javase/6/docs/technotes/guides/javadoc/doclet/overview.html
11. H. C. Jiau and F.-P. Yang. Facing up to the inequality of crowdsourced api documentation. *ACM SIGSOFT Software Engineering Notes*, 37(1):1–9, 2012.
12. A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th international conference on Software Engineering*, pages 344–353. IEEE Computer Society, 2007.
13. S. Letovsky. Cognitive processes in program comprehension. *Journal of Systems and Software*, volume 7, number 4, pages 325–339. Elsevier, 1987.
14. ninlabs blog, http://blog.ninlabs.com/2013/03/api-documentation/
15. C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*.
16. D. Posnett, E. Warburg, P. Devanbu, and V. Filkov. Mining stack exchange: Expertise is evident from earliest interactions.
17. P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *ICSE 2013*, 2013.
18. J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *Software Engineering, IEEE Transactions on*, 34(4):434–451, 2008.
19. StackOverflow Data, http://blog.stackoverflow.com/category/cc-wiki-dump/
20. C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web?: Nier track. In *Software Engineering (ICSE), 2011 33rd International Conference on*, p. 804–807. IEEE, 2011.
21. C. Treude and M.-A. Storey. Effective communication of software development knowledge through community portals. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 91–101. ACM, 2011.
22. Q. H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica: Journal of the Econometric Society*, pages 307–333, 1989.