When Stack Overflow Slows Down: Latency of Answers to Questions About New Android APIs

David Kavaler[1*], Vladimir Filkov[1¤]

1 Department of Computer Science, University of California, Davis, Davis, California, USA

* dmkavaler@ucdavis.edu ¤filkov@cs.ucdavis.edu

Abstract

Stack Overflow is a popular crowdsourced question and answer website for programming-related issues. It is an invaluable resource for software developers; on average, questions posted there get answered in minutes to an hour. Questions about well established topics, *e.g.*, the coercion operator in C++, or the difference between canonical name and class name in Java class, get asked often in one form or another, and answered very quickly. On the other hand, questions on previously unseen and on niche topics take a while to get a good answer. This is particularly the case with questions about current updates to or the introduction of new application programming interfaces (APIs). In a hyper-competitive online market, getting good answers to current programming questions sooner could increase the chances of an app getting released and used. So, can developers anyhow hasten the speed to good answers to questions about new APIs?

Here we study empirically Stack Overflow questions and answers pertaining to new Android APIs, and contrast the interest in them, their answer quality, and timeliness of the answers to questions about existing APIs. We find that Stack Overflow answerers in general prioritize with respect to currentness: questions about new APIs do get more answers, but good quality answers take longer. We also find that incentives in terms of

question bounties, if used appropriately, can significantly shorten the time to a good answer.

Introduction

The social coding movement and the phenomenon of crowdsourcing have made free and eminently useful software development resources and services possible. Stack Overflow and Open Source Software are transformative in that they enable the creation, promulgation, and archiving of new knowledge and artifacts on an as-needed basis. They are often also very responsive: most questions on Stack Overflow are answered within minutes, and pull requests get reviewed, merged, and released into the codebase of large projects within days. In fact, Stack Overflow and Open Source work quite well as a coupled, interdependent system, with the former providing almost instantaneous documentation for the latter, and developers of the latter serving as askers and answerers in the former's gift economy. For example, software developers often use social media sites (such as Stack Overflow) as part of their normal workflow [1] to ask a wide variety of questions [2], and many tools have been developed to aid in this process [3, 4]. This system is critically predicated on the turnaround time to answers being very short and on there being enough knowledgeable eyeballs to provide the expertise needed. As soon as one or both of those conditions are unmet, the programming and documentation resources get decoupled.

On Stack Overflow, users are accustomed to having their questions answered rapidly; the standard speed to an answer is between 17 and 47 minutes, depending on subject area [5]. In addition, users understandably want as high of a quality as possible out of the answers they receive. It has been shown that Stack Overflow is effective at code reviews and conceptual questions [6], as well as providing adequate API coverage [7]. In addition, highly used APIs are also generally discussed more [8]. But many articles and questions get asked which don't get fast enough attention from the crowd [7]. Although Jiau and Yang argue that more obscure questions benefit from a "trickle-down" effect from similar questions [9], some questions are more time-critical and may need an answer even faster. Though work has been done to reduce the number of low-quality posts on Stack Overflow [10, 11], these issues remain.

Stack Overflow has implemented various incentive mechanisms to encourage user participation, including badges (which serve as rewards for achieving various feats), reputation (gained through participation), and various privileges awarded upon reaching reputation milestones. These incentive mechanisms have proven to be effective in garnering activity and popularity [12, 13]. Movshovitz-Attias *et al.* found that high reputation users are the primary source of high quality answers [14]. Grant and Betts examined three specific Stack Overflow badges in detail, finding that users tend to increase their activity in order to attain these badges [15]. Parnin *et al.* have shown that Google searches for code-related questions often link to Stack Overflow, *e.g.*, 84.4% of jQuery methods had a Stack Overflow post returned on the first page of the related Google search [16].

However, it has been noted that Stack Overflow's incentive mechanisms can be at odds with question and answer quality. Jin *et al.* studied gamification-influenced member tendencies on Stack Overflow, arguing that the fastest response often "wins" the most reward [17]. Bosu *et al.* studied exactly what actions a user can take to build reputation quickly [18], concurring with Jin *et al.* and finding that a number of non-expertise related strategies can effectively increase reputation (*e.g.*, activity during off-peak hours). In addition, there have been discussions about declining quality due to the emergence of an "old boys' club" mentality [19], and the existence of "one-day flies"; the vast majority of Stack Overflow users only post once [20]. Posnett *et al.* found evidence that users on Stack Exchange (the umbrella under which Stack Overflow lies) do not increase in answering expertise over time [21]. In light of this, it is important to understand how to attract attention to one's questions in an effective manner and from the true experts. This is especially true for questions about novel topics that have only recently arisen, *e.g.*, new APIs.

Motivated by the above, here we take a look at a specific threat to the documentation/coding coupled system of Stack Overflow and Open Source: questions related to recently introduced APIs, which present a special challenge to getting fast and good answers. In fact, our data shows that, on average, questions referencing new APIs are answered *8,000* minutes (about 5.5 days) slower than questions referencing only non-new APIs. Other researchers have studied the topic of answer speed [22–24] in community question and answer sites, with varying degrees of success using a variety of

methodologies. Here, in contrast to most prior work, we are interested only in questions and answers related to Android - that can be linked to Android APIs - and use a standard regression framework for inference. Linares-Vásquez *et al.* [25] found that Android API behavior modifications trigger much discussion on Stack Overflow, indicating that there is interest within the community regarding new or changed Android APIs, meaning our restriction to studying only Android APIs should not be debilitating.

Research Questions

Older APIs, having been around longer, have a higher chance to be well-documented by the crowd. On the other hand, new APIs, by definition, have no existing crowd documentation for users to rely upon. In addition, new APIs may not be as well documented to begin with, as their developers have not yet had enough feedback from general users to indicate the confusions that arise requiring more documentation. Developers may thus have higher need for a fast and good quality answer to a question related to a new API. First, we ask to what extent does this show in Stack Overflow?

> **Research Question 1**: Are answers to new API questions more, faster, or of higher quality?

Reputation on Stack Overflow is gained through various methods, primarily by receiving up votes on questions and answers and by having an answer being marked as "accepted", indicating that the asker believes the answer is the best among those received. Related to reputation is the *bounty* system. Bounties can be seen as a layer on top of the existing Stack Overflow knowledge exchange system which allows a user to "pay" for additional services on top of the basic, public ones. The ability to attach a bounty to a question requires some amount of participation in Stack Overflow (*i.e.*, a total of 75 reputation). Anderson *et al.* set out to predict the long-term value of a question, as well as whether a question has been sufficiently answered [26]. To accomplish the latter, they attempt to predict whether or not a question will attain a bounty, which serves as an indicator that the question was not adequately answered given the answers that exist before the bounty is started. Berger *et al.* studied bounties and their effect on question performance compared to non-bountied questions [27].

However, it is still unclear exactly what (if any) is the outcome of the bounty offering. Some potential outcomes are: increased quality of answers, reduced time until a quality answer, attracting people who can answer difficult questions better or faster, *etc.*. But are any of these potential outcomes realized? And if so, are they manifest?

Research Question 2: What are the effects of one type of incentives, the bounties?

Finally, Stack Overflow users may conceivably be able to affect the process to receive faster and higher quality answers, by, *e.g.*, increasing the effort required in framing and posing a question, or posting and manipulating a bounty. In the last question we focus on extracting practical advice from the answers to RQ1 and RQ2.

Research Question 3: What are the lessons learned?

To answer the questions above we fused data from two sources: Stack Overflow and the Google Play store. From the latter we gathered function invocation data on 20,325 Android apps, and from the former we gathered questions and answers that mention Android APIs used in those apps. In this work, references to APIs are specifically for Android APIs. Then, we build separate regression models for time to a good answer, number of answers, and answer quality as functions of question attributes, bounty usage, and many confounding variables. Our findings show:

- Questions involving new APIs attract more answers.
- Questions involving new APIs receive good answers slowly (compared to those questions involving older APIs), given this answer comes within 2 days. However, if the first good answer comes after 2 days, new APIs attract faster answers. Of questions which are answered only after 2 days, only 46% are answered within 1 month.
- The bounty not only reduces time to answer on average, but also flattens the long tail and increases density towards faster answers. Bountied also questions receive more answers. However, the exact reputation value of the bounty does not seem to matter.
- Answers that come during a bounty period are of higher quality. Questions referencing new APIs have no significant effect in receiving higher quality answers.

Materials and Methods

In the following sections, we describe our data and how it was collected, our strategy for identifying and linking APIs to Stack Overflow questions, various statistics we calculated for use in our models, our modeling strategy, and how we filtered our data to ensure model robustness.

Data Collection

Stack Exchange provides public data dumps periodically for all the sites within the Stack Exchange network, including Stack Overflow. We use data from the Stack Overflow data dump dated March 16, 2015 (retrieved from `https://archive.org/details/stackexchange`). From this data, we extracted a rich set of variables, including question view count, user-defined question tags, question and answer scores, question asker and answerer reputations, *etc.* In addition, we calculated a large set of variables based on this data including number of words in the body of a post, amount of code in a post, question title length, question asker and answerer "wisdom" scores (explained below), *etc.*

In addition to Stack Overflow related data, we developed and used a metric that requires API call counts from real Android applications. To serve this purpose, we wrote a custom crawler to download free applications from the official Google Play app store (`https://play.google.com`). The crawler operates by selecting a random application on the front-page of the Google Play store, and emulates a search through applications by "clicking" through all application links seen on each page. Note that this search is not entirely random, but attempts to emulate a random search through the space of applications. This pseudo-random search is necessary as there is no simple method of extracting a random application from the Google Play store. This crawler downloaded a total of $20,325$ applications. We then converted the apps to a more human-readable byte code format using APKTool [28]. We processed the extracted byte code files by looking for function invocations (`invoke-virtual`, `invoke-super`,`invoke-direct`,`invoke-static` and `invoke-interface`), and recorded (among other things) usage counts. In addition, we gathered documentation data from Android source code by running *Javadoc* with a custom Doclet [29]. This allowed us to

gather data such as class documentation line counts, number of inner classes (*e.g., Animator.AnimatorListener*), and average method documentation lines. Below we discuss more in-depth points about our data.

Android Change Data

To collect API change data, we use the official Android change lists provided by the Android SDK manager. However, some of these change lists are incomplete. For example, according to the documentation website, the class *android.accounts.AccountManager* was added in API level 5. However, the change list packaged with the SDK release has no mention of this class (change list can be viewed here: `https://goo.gl/I4tsPl`). For APIs with this issue, we assume that the API was added in API level 1. In this work, **we identify an API as "new" if it has not been modified between the time of its introduction and the time the question is asked**. We note that it is likely clearer to classify new APIs as those that were added only in the most recent framework change. However, there are a number of reasons we do not define new APIs in this manner. Developer adoption of new frameworks can be relatively slow for existing applications, as updating to the newest framework versions always involves risk. Although the Android framework claims strict backwards compatibility for their APIs, and rarely remove APIs outright, there is always an inherent risk of breaking the current code base with any underlying framework update. In addition, there are periods of time in which new Android frameworks are released very rapidly; for example, API levels $2-7$ all released within the same year. If we define new APIs as those newly introduced in the latest update, we are severely limiting our data for a number of time points, as there is very little time for new APIs to be discussed. Thus, in order to have enough data to reliably model, we define a new API as described in bold above.

Stack Overflow Question API Links

To link questions to relevant APIs, we examine the text body of Stack Overflow questions to extract *links* to APIs. The link types considered here are:

1. *Tag links*: A class name match occurring in the tags section of a Stack Overflow

question.

2. *Href markup links*: A class name match enclosed by HTML `<a></a>` tags, referring back to the Android documentation site.

3. *Title links*: A class name match occurring in the title of the Stack Overflow question.

4. *Code links*: A class name match *exactly* occurring within HTML `<code></code>` segments - this means large code blocks are not considered when identifying API links (Fig 1, "inline code segment"). Large code blocks were not considered in API linking as they create large numbers of false positive links as users post long code segments to show how they have tried to solve their problem in question.

Fig 1. A question and answer on Stack Overflow. Some relevant variables are outlined in red.

This is a similar strategy as that used in prior work [7,8], with some alterations. These alterations were made to focus on identifying *true positive* links, while minimizing *false positives*. For our models to be useful in answering our research questions, we believe it is more important to make sure our data set includes only properly linked APIs than covering all questions referencing APIs; hence the emphasis on true positives. In addition, we consider an API as a particular *class mention*, *e.g.*, *android.app.Activity*, rather than by *method mention*, *e.g.*, *android.app.Activity.onCreate()*. This is due to the fact that method names are often more generic than class names, *e.g.*, a method named *start()* may belong to many classes. Thus, to disambiguate API mentions, we consider classes rather than methods. When determining links, searches for both fully qualified class names (*e.g.*, *android.app.Activity*) were considered along with class names alone (*e.g.*, *Activity*).

Wisdom Scores

As we are primarily interested in assessing the effect of new APIs on various outcomes (*e.g.*, response time), we must control for answerer expertise, which can also affect our outcomes of interest. As such, we require a metric to measure expertise within the framework of Stack Overflow *i.e.*, not necessarily purely technical expertise. There has

been much interest in measuring user expertise on Stack Overflow, with researchers investigating multiple dimensions that contribute to expertise, along with applications of measures [30–33]. However, most definitions of expertise are coarse-grained; *e.g.*, merely using reputation as a measure of expertise. Here, we leverage work by Yang *et al.* [34]. They introduce a novel metric called Mean Expertise Contribution (MEC), referred to as a "wisdom" score. In essence, this metric considers two dimensions of user wisdom or expertise: the *debatableness* of a question, and the *utility* of an answer. MEC is defined as:

$$\mathrm{MEC}_{u,t} = \frac{1}{|Q_t^u|} \sum_{\forall q_i \in Q_t^u} \mathcal{AU}(u, q_i) * \frac{\mathcal{D}(q_i)}{\mathcal{D}_t^{avg}}$$

where:

- Q_t^u is the set of questions from user u on topic t.
- $\mathcal{AU}(u, q_i)$ is the *utility* of the answer provided by user u to question q_i; $\mathcal{AU}(u, q_i) = \frac{1}{Rank(a_{q_i})}$ *i.e.* the inverse rank of the answer provided by u for question q_i. A rank of 1 indicates the highest scoring answer for a question post. Thus, a larger $\mathcal{AU}$ indicates a higher expertise level shown by user u for question q_i.
- $\mathcal{D}(q_i)$ is the *debatableness* of question q_i, calculated as the number of answers $|A_{q_i}|$ provided for question q_i.
- $\mathcal{D}_t^{avg}$ is the *average debatableness* of all questions related to topic t, calculated as $\frac{1}{|Q_t|} * \sum_{\forall q_j \in Q_t} |A_{q_j}|$

A value of $\mathrm{MEC_{u,t}} = 1$ indicates that user u, on average, provides the best answer to averagely debated questions. $\mathrm{MEC_{u,t}} = 0.5$ indicates that user u ranks second in answering averagely debated questions, or ranks first in answering less debated questions. We use this metric in our models.

Text-based Variables

The Stack Overflow data dump includes the body of all posts including HTML markup as displayed on the website. Using this data, we can extract variables in addition to

API links, including the number of words in a post, the amount of code in a post, and specific information about the structure of the HTML used in the post.

To extract the amount of code in a post, we take care to differentiate between code blocks and inline code segments, as shown in Fig 1. These two types have slightly different HTML markup on Stack Overflow. We calculate both the lines of code and total characters of code in both code blocks and inline code segments.

To extract word-based variables, we use JSoup (`http://jsoup.org/`) to remove code *blocks* and send the resulting raw text (*i.e.* without HTML tags) to the Stanford CoreNLP library [35] to tokenize and detect sentences. This way, our word-based variables include inline code *segments*, but not code *blocks*. This is because inline code segments are often used as part of a natural language sentence, and we believe they should be treated as words. On the other hand, code blocks are purely formatted code, which shouldn't be analyzed as natural language text. Prior work also shows that natural language text is just as important as code in a Stack Overflow question [36], thus we must have some representation of language in our models.

In addition, we extract the number of URLs in the body of the post, the number of user-defined tags for the associated question, and the length of the title of the associated question. This is done as often posters will include links to documentation and related Stack Overflow questions and answers.

Finally, we calculate the number of switches between HTML tag types in the base-level body of a post. This is a measure of structural complexity of the post. We theorize that the more switches between natural language text and code in a post, the more complex the post is in terms of content. In addition, we believe that some structural information should be included in the models as more structure can increase readability in terms of visual clarity. To calculate this, we extract HTML tag sequences and count the number of switches between different tag types at the base-level. For example, if we see a sequence of tags such as:

```
<a></a><p></p><p></p><code></code><pre><code></code></pre>
```

we would count three switches; note that there is an embedded `<code>` tag within the `<pre>` tag - as this is not at the base-level of the post body, we do not count this. This

also avoids double counting code blocks which are visually a single unit, but could be considered two structural units if one does not count the HTML tags in the aforementioned way. In Fig 1, the number of switches for the answer would be equal to 2, even though there is an inline code segment in the final paragraph.

Question and Answer Quality

The task of assigning a quality label to posts comes down to answering the question: what makes a post "good"? Baltadzhieva and Chrupala surveyed various metrics from prior work in determining Stack Overflow question quality, including tags and terms within the question itself [37]. Tian *et al.* use answer acceptance as a proxy for measuring a "good" answer [38]. Similarly, Shah and Pomerantz examined the Yahoo! Answers data set and used human assessments through Amazon Mechanical Turk to build a model for predicting which answer would be chosen as best by the question asker [39]. In our analysis of the data, we found that very few answers are actually designated as "accepted", even though the answer quality might in fact be quite high. In addition, as noted by Gantayat *et al.* [40], often the accepted answer is not the best according to community popular vote. Thus, using the accepted answer as an indicator of answer quality may not accomplish what is intended.

For guidance, we refer to the methods and arguments provided by Ravi *et al.* [41]. In their work, they address issues of conflating quality with popularity, as a question that is viewed many times has higher chances to get votes. Through arguments and some empirical analysis, they decide to consider the quantity $p_i = s_i/v_i$, where s_i is the score for question q_i and v_i is the view count. Here, the view count acts as a control for popularity. They go on to argue for labeling questions with $p_i = 0$ as "bad", and labeling questions with $p_i > 0.001$ as "good".

We argue that similarly for questions, answer quality should be a function of associated view count and answer score. However, for answers, the use of view count is slightly different as we only have access to view count at a question-level. Thus, it is likely that some answers are viewed more than others, and that the view count variable does not accurately reflect this. As a result, in all relevant models we control for the time difference in question creation to answer creation. This serves as a control to alleviate the bias that the view count variable has towards answers that are created

earlier.

Residual Question Need for Documentation

In previous work [8], we addressed the idea of Stack Overflow as a documentation source, and built a model to predict the number of API linked Stack Overflow questions using actual API usage in free Android applications and a number of controls. The model is of the form:

$$\begin{aligned}\text{Number of API linked questions} = \beta_0 + \\ \beta_1 \text{Number of API calls in free apps} + \beta_2 \text{Source documentation lines} + \\ \beta_3 \text{Number of inner classes } + \beta_4 \text{Class documentation lines} + \\ \beta_5 \text{Average method documentation lines per class}\end{aligned}$$

where the β_i are estimated model coefficients. In-depth information about this model can be found in the mentioned work.

As this model predicts the number of linked Stack Overflow questions per API, where questions correspond to documentation, we view the **negative** of the residuals of this model as representing *documentation need*. If the negative residual of the documentation need model is negative, our model predicts a lower amount of documentation than exists on Stack Overflow, indicating that the API is *over-documented*; if the negative residual is positive, the API is *under-documented*. We emphasize that the idea of using this metric is to represent API documentation need as a function of API usage in real applications, and a number of controls. The theory behind this is that, generally speaking, with more knowledge seekers there is an increased probability of nuanced, specific questions, as the general usage questions have already been answered and thus are less likely to turn up again. As a result, an API that is used more will likely require more documentation to satisfy users' needs than one that is used less. This also provides another metric for the *currentness* of Stack Overflow. If Stack Overflow is very current, *i.e.* up-to-date in terms of API documentation, then documentation need for APIs will generally be low. This metric is taken into consideration along with analysis of new APIs to measure currentness.

Stack Overflow Bounty

On Stack Overflow, users can choose to place a *bounty* on a question after the question is 2 days old. A bounty is an extra reputation point bonus applied to a question, funded by the bounty creator's own reputation score. The bounty creator can choose to spend between 50 and 500 reputation (in accordance to various rules) on a bounty.

The predominant function of the bounty system is to attract *extra attention* to a question. Questions with active bounties are put into a special "featured" section in the main Stack Overflow question list, granting them increased visibility. We can use this mechanism to shed light on whether or not the bounty increases answer quality, answer timeliness, or answer count, and thus whether a user can reliably spend reputation to gain these potential benefits.

Data Filtering

Posts that are older than one year that meet a certain set of criteria are deleted from Stack Overflow and the underlying data dump [42]. To address this, we only consider questions and answers created before March 16, 2014 (1 year prior to the dump date) to avoid issues of sample bias in our models. Note that the data we use and the resulting metrics calculated based on the data are from the *snapshot date*, *e.g.*, reputation for users is calculated as of the date of the snapshot, not the date of the posting. This is due to the way that Stack Overflow structures its data dump.

We primarily use two supplied tables from the Stack Overflow data dump: the Posts and Votes tables. The Posts table contains the posts themselves along with meta-information. The Votes table contains each vote (*e.g.*, up, down, flagged as inappropriate, *etc.*) for each post. There are a number of consistency issues with these two tables that must be addressed before they are used in our models.

Posts which are deleted are not contained in the dumped Posts table. However, votes for these posts are sometimes not deleted from the Votes table. Additionally, if a post is migrated from Stack Overflow to somewhere else in the Stack Exchange network and a bounty was started while the post was still on Stack Overflow, the Votes table will contain an entry for the start of the bounty while it will not contain an entry for the end of the bounty. As a result, we only look at question threads which have not

been migrated or deleted as of the data dump. There are a number of observed discrepancies in the Stack Overflow data set, mostly arising due to deleted posts, migrated posts, and related administrative actions. We made a best-effort attempt to clean the data of these inconsistences. These issues affect a vast minority of our data points and should have a negligible effect on our outcomes.

For our models, we do not consider answers from users who have deleted their accounts or answered without an account, as this causes their reputation scores to be lost in the data. Similarly, we do not consider questions in which the question asker has deleted their account or asked without an account. After filtering for all of these issues, our data set reduces from $633,659$ Android-tagged questions to $410,287$ questions. The final step in filtering is to consider only those questions which are positively linked to an API, leaving us with $22,366$ questions for all models presented.

Regression Analysis and Model Selection

To answer each of our research questions, we have separate models using various forms of linear regression. This allows us to inspect the relationship between our response (*dependent variable*) and our explanatory variables of interest (*predictors* or *covariates*, *e.g.*, documentation need), under the effect of various *controls*.

For examining the number of answers per question, we use a Poisson generalized linear model (GLM), as is standard with count data [43]. For examining the time to first good answer, we use ordinary least squares (OLS) regression with a logged dependent variable. Though time can be considered a count variable, we tested model fit between the OLS regression models and Poisson GLMs and found better fit with the OLS models. Finally, for answer quality models, we use logistic regression with a binary dependent label of "bad" or "good".

In this work, all models except the model for answer quality are at the *question level*, *i.e.*, each observation is a question. For the answer quality model, each observation is an answer. As a result, for our time-to-answer models, we model the time to first *good answer*, where "good" is defined by answer quality label. We considered modeling at the *answer level* for all models; however, this would lead to multiple observations of the same question. Multiple observation can lead to high levels of correlation between

covariates, potentially negatively affecting model inference. Among methods able to handle multiple observations are *mixed-effects* (or *random effects*) models. To test whether or not a mixed-effects model is necessary compared to a fully fixed-effects model (*i.e.*, if a random effect for question ID is necessary), we compare the *Akaike's Information Criterion* (AIC) of the models with and without the corresponding random effect [44]. In the end, we decided against mixed-effects models both by their comparison of AIC and according to the principle of parsimony [45]; if the more complicated model is only marginally better, use the simpler model.

In addition, in order to observe the effect of the bounty on time-to-answer we separate our time-to-answer models into two parts: one for answers that come within 2 days of the question being asked, and one for answers after 2 days. This is because bounties can only be added 2 days after a question's creation. Since most questions are answered within 2 days (88%), these questions necessarily cannot have a bounty, causing a very large skew in a combined model towards non-bountied questions. As a result, we believe that combining these two models would cause the bounty factor to be ineffective for inference, as the combined model is likely to be heavily biased towards non-bountied questions; in other words, the model will likely mostly capture the variance in non-bountied questions, as they are the vast majority of the data set. In fact, when examining the residuals vs. fitted values plot for the combined model, there is a comparatively poor fit for fitted values at and over 2 days. Due to this poor fit for higher fitted values in the combined model, and the heavy skew towards non-bountied questions in the data, we separate the two models to make sure the bounty factor can be safely used for inference.

We employ *log* transformations on predictor variables to stabilize the variance and improve model fit when appropriate [46]. As explanatory variables are often highly correlated, we consider the *variance inflation factor* (VIF) of the set of predictors and compare against the recommended maximum of 5 to 10. All models presented have a maximum VIF of 3. To determine whether explanatory variables should be kept or removed, we compare models using likelihood ratio tests [47].

Variable names and descriptions can be found in Table 1. Note that some variables with a calculable answerer counterpart were computed, but not used in models due to issues of multicollinearity. In addition, some other variables were computed but not

used due to issues of multicollinearity, *e.g.*, number of lines of code in the post body.

Table 1. Model variable descriptions. All numeric explanatory variables are logged, except APIDiffTime, TimeToBounty, and QNeed.

Variable name	Description
F.QQualityLabelGood	Label for question quality.
TimeToBounty	Time to bounty start (days). Equal to 0 for questions that never receive a bounty.
TimeToAnswerMins	Time-to-answer, in minutes.
QCreationDate	Number of days between the first Stack Overflow post (ever) and the question creation date.
QOwnerNQ	Total number of questions created by the question owner.
Q/AOwnerReputation	Reputation for the post owner.
QOwnerAge	Number of days between question owner's account creation and the question creation date.
Q/AMEC	(Mean Expertise Contribution) MEC for the post owner.
QTitleLength	Title length for the question.
QNTags	Number of tags for the question.
Q/ABodyNWords	Number of words in the post body, not including code.
Q/ABodyCharsOfCode	Number of characters of code in the post body, including both code blocks and inline code segments.
Q/ANSwitches	Number of structural changes in the post body.
Q/ABodyURLCount	Number of URLs in the post body.
QNComments	Number of comments for the question.
QNeed	Calculated question need, scaled.
F.Bounty	A factor indicating whether or not the first good answer was provided during a bounty period.
F.Added	A factor indicating whether or not the question references a newly added API.
APIDiffTime	Minimum number of days from which a linked API was changed for all linked APIs in the question. If a new API is present, this is the number of days since the new API was added.

Results and Discussion

Case Study: New API Interest in Stack Overflow

One of our goals is to study the effect that newness of APIs mentioned in a question has on answer timeliness and quality. The implicit assumption is that question askers on Stack Overflow care about and use new APIs. For our models to be relevant, we must justify this assumption. Manually inspecting all questions linked to new APIs is infeasible, as we have 22,366 questions found by our linking strategy. Thus, we took a random sample of 50 questions linked to new APIs and categorized them as "explicitly about", "involving", or "not about" the linked new API. The "explicitly about" class is as its name: if the question is explicitly about the new API, it is classed as such. An example of such a question is:

Question ID: 14620974, Linked API: SeekBar

Title: Seekbar increase value up to 100

I have a seek bar with max=25. What I want to do is when a user drags the seekbar to max value and it is in a pressed state [...]

The "involving" class consists of questions that explicitly state the new API, but the question does not address it directly. An example of such a question is:

Question ID: 11485026, Linked API: SeekBar

Title: Seekbar creating EditTexts and then getting entries for further use

This code creates a seekbar and makes the seekbar create as many EditText fields as the slider is at / remove ones that would be too much. This code is in OnActivityCreated [...]

As shown, the "involving" question above does explicitly mention the new API (*SeekBar*), but it is not clear that the question is entirely about the new API itself. We found that the difference between these two groups is often small, but still worth separating.

The results of our case study are as follows. Only 2 linked questions in the sample are not about the linked API. The two linked APIs in this case are *NetworkOnMainThreadException* and *ImageButton*. In the case of the former, the question is about the exception generated by the Android operating system itself, not the exception class. We avoid most of these issues with *Exception* classes by the way we discover APIs in questions, outlined in previous sections. For the latter, the question asker provided an incorrect user-defined tag.

As 48 of the 50 questions in the case study are in either the "involving" or "explicitly about" groups, we have confidence that our models can be used for inference. To assuage potential concern about the difference between the "involving" and "explicitly about" groups, we sought to identify a control that can be used to separate between the two groups. Fig 2 shows time (days) since API addition for the new APIs referenced in the 50 case study questions, by manually classified group. The box plot shows that questions in the "explicitly about" group are generally posed closer to the date of their referenced API's addition than those questions in the "involving" group. This indicates that the number of days since the addition of a referenced API can be a useful control in dividing the "involving" from the "explicitly about" subgroups within

the group of questions referencing new APIs.

Fig 2. Time since API addition for 50 question case study, per manual classification group (2-sided t-test $p < 0.05$).

This case study shows that Stack Overflow users indeed ask questions about new APIs, and we can conclude that Stack Overflow users actually do use and care about new APIs. In addition, it provides confidence that our linking strategy indeed proritizes true positive links, as false positives are rare. Meeting these core assumptions allows us to use the models we build for inference.

RQ1: Are Answers to New API Questions More, Faster, or of Higher Quality?

Number of Answers to New API Questions: Table 2 shows our model for the number of answers per question. Column 1 serves as a base model; only controls for question creation date and question-related expertise metrics are used. Column 2 adds question-specific descriptive variables *i.e.*, textual variables and user-defined tag count. Column 3 introduces a variable that is not in control of the question asker (**QNComments**), API-related variables (**APIDiffTime, F.Added**), and variables related to the bounty (**F.Bounty, TimeToBounty**).

We see that questions linked to new APIs (**F.Added**) receive more answers (0.076), when controlling for other relevant variables. This is a positive result for Stack Overflow - new APIs are a topic of interest to developers, and one would hope that their needs for the most current documentation is met.

Additionally, this model lets us examine how Stack Overflow reacts to other types of questions in need of an answer. The variable **QNeed** measures documentation need based on usage in real Android applications. We see that although new APIs have a positive effect (0.076) on number of answers, documentation need has a negative effect (-0.014). Recall that **F.Added** is a factor, while **QNeed** is a continuous variable. In our data, the range of **QNeed** is $[-6.85, 3.912]$. Thus, considering both **F.Added** and **QNeed**, the minimal value that their coefficients can jointly contribute to the prediction equation (assuming a question with a new API that is also in high documentation need) is $0.076 + (-0.014 * 3.912) = 0.021$, a positive number. Thus,

Table 2. Number of answers per question, Poisson GLM

	Dependent variable:		
	Number of answers per question		
	(1)	(2)	(3)
QCreationDate	-0.110^{***}	-0.100^{***}	-0.175^{***}
QOwnerNQ	0.022^{***}	0.016^{**}	0.012^{*}
QOwnerReputation	0.011^{**}	0.012^{**}	0.010^{*}
QOwnerAge	-0.019^{***}	-0.016^{***}	-0.013^{***}
QMEC	-0.015	-0.009	-0.006
QTitleLength		-0.056^{***}	-0.033^{*}
QBodyNWords		-0.048^{***}	-0.054^{***}
QBodyCharsOfCode		0.012^{***}	0.007^{*}
QNSwitches		-0.004	-0.014
QBodyURLCount		-0.018	-0.031^{**}
F.QQualityLabelGood		0.083^{***}	0.049^{***}
QNeed		-0.018^{***}	-0.014^{***}
QNTags		-0.018^{***}	-0.017^{***}
QNComments			0.150^{***}
F.Added			0.076^{***}
APIDiffTime			-0.00005^{**}
F.Bounty			0.203^{***}
TimeToBounty			0.0003
Constant	1.305^{***}	1.584^{***}	2.075^{***}
Log Likelihood	$-32{,}400.610$	$-32{,}303.640$	$-31{,}589.460$
Note:			*p<0.05; **p<0.01; ***p<0.001

although the beneficial effect of **F.Added** is somewhat mediated by the negative effect of **QNeed**, their overall effect is a net positive on number of answers. Additionally, as **QNeed** can be negative, a question with low documentation need that is also linked to a new API receives even more answers, probably indicating that the question is easy to answer, or well-known.

Latency of Answers to New API Questions: Tables 3 and 4 show our models for time to first good answer, for answers that come before and after 2 days. Column 1 serves as the base model. Column 2 adds answer-related variables, such as answerer expertise and answer text metrics. Column 3 adds question-related variables.

For questions with a first good answer within 2 days (Table 3), we see that questions linked to new APIs receive slower answers (0.133). For questions with a first good answer after 2 days (Table 4), we see that questions linked to new APIs receive faster answers (-0.216).

When comparing our two models for time to first good answer, the situation seems

Table 3. Time to first good answer models, before 2 days

	Dependent variable:		
	Log time to answer (minutes, answer before 2 days)		
	(1)	(2)	(3)
QCreationDate	-0.553^{***}	-0.994^{***}	-1.070^{***}
QOwnerNQ	-0.184^{***}	-0.158^{***}	-0.110^{***}
QOwnerReputation	0.050^{***}	0.047^{***}	0.012
QOwnerAge	0.044^{***}	0.036^{***}	0.039^{***}
QMEC	0.249^{***}	0.229^{***}	0.192^{***}
ABodyNWords		0.312^{***}	0.262^{***}
ABodyCharsOfCode		0.056^{***}	0.055^{***}
ANSwitches		-0.042^{**}	-0.044^{**}
ABodyURLCount		-0.008	-0.002
AMEC		-1.693^{***}	-1.673^{***}
AOwnerReputation		-0.139^{***}	-0.135^{***}
QNTags			0.071^{***}
QTitleLength			0.098^{***}
QBodyNWords			0.262^{***}
QBodyCharsOfCode			0.009
QNSwitches			-0.030
QNComments			0.095^{***}
QBodyURLCount			0.192^{***}
F.QQualityLabelGood			0.181^{***}
QNeed			0.045^{***}
F.Added			0.133^{***}
APIDiffTime			-0.0001^{**}
Constant	7.035^{***}	9.939^{***}	8.994^{***}
R^2	0.032	0.133	0.164
Note:			*p<0.05; **p<0.01; ***p<0.001

contradictory at first. For questions linked to new APIs with a first good answer within 2 days, we see that answer comes slower; for after 2 days, we see that answer comes faster. For the former case, the explanation could be that questions with new APIs are harder to answer, and thus answers come slower. This is supported by the fact that question text variables which serve as a proxy for complexity (*e.g.*, **QTitleLength**, **QBodyNWords**, **QBodyURLCount**) all have significant positive values. In addition, our data shows that there are far fewer unique people who answer questions that reference new APIs $(2,266)$ than those who answer questions that are not about new APIs $(7,264)$ - this may be due to new APIs requiring specific knowledge that is not yet widespread. However, this effect can be mitigated by the effect of **QNeed** (0.045). As stated above, the range of **QNeed** is $[-6.85, 3.912]$. Thus, for a question

Table 4. Time to first good answer models, after 2 days

	Dependent variable:		
	Log time to answer (minutes, answer after 2 days)		
	(1)	(2)	(3)
QCreationDate	-2.126^{***}	-2.125^{***}	-1.590^{***}
QOwnerNQ	0.059^{*}	0.062^{*}	0.091^{***}
QOwnerReputation	-0.068^{**}	-0.023	-0.020
QOwnerAge	0.004	0.008	0.026
QMEC	-0.349^{***}	-0.437^{***}	-0.358^{***}
ABodyNWords		-0.008	0.028
ABodyCharsOfCode		0.002	0.00001
ANSwitches		0.012	0.032
ABodyURLCount		0.113^{*}	0.090^{*}
AMEC		0.097	0.131
AOwnerReputation		-0.147^{***}	-0.115^{***}
QNTags			-0.060^{*}
QTitleLength			-0.031
QBodyNWords			-0.093
QBodyCharsOfCode			-0.002
QNSwitches			-0.068
QNComments			0.032
QBodyURLCount			0.023
F.QQualityLabelGood			0.377^{***}
QNeed			0.023
F.Added			-0.216^{*}
APIDiffTime			0.0002^{**}
F.Bounty			-1.240^{***}
TimeToBounty			0.011^{***}
Constant	26.115^{***}	26.900^{***}	23.176^{***}
R^2	0.157	0.184	0.291
Note:			*p<0.05; **p<0.01; ***p<0.001

linked to a new API that is also in low documentation need, the slowing effect of a new API can be entirely nullified if the documentation need is at most $-0.133/0.045 = -2.95$ - well within our range of observed values. Thus, we see that new APIs experience longer latency, but this effect can be mitigated or nullified by a low documentation need, *i.e.*, new APIs with low documentation need can experience low answer latency.

For the case of questions linked to new APIs with a first good answer after 2 days, the explanation could be as follows. Questions with a first good answer after 2 days are harder to answer; otherwise, they would likely have received a faster answer (median time to first good answer in our data is 17 minutes). In addition, our data shows that of questions that are answered only after 2 days (2,525), only 46% are answered within 1

month. This result is more nuanced than what has been discussed in the past; questions that are hard enough to not receive an answer within 2 days often take longer than 1 month to answer - a far cry from the median answer time of 17 minutes. However, users want to document new APIs - this is supported by prior work that shows Android classes are highly documented, and generally done so quickly [7]. Thus, for questions that already take longer to answer, questions referencing new APIs receive comparatively faster answers. In this model, **QNeed** is not significant, and is thus not considered.

Quality of Answers to New API Questions: Table 5 shows our model for answer quality. Column 1 serves as the base model. Column 2 adds answer-related variables, and column 3 adds question-related variables.

We see a positive effect of question quality, indicating that higher quality questions receive higher quality answers, even when controlling for time-to-answer. This is in agreement with prior work [48]. We see no effect of new APIs on answer quality. However, we do see that documentation need has a positive effect on answer quality (0.032), indicating that APIs with high documentation need are more likely to receive a higher quality answer. Thus, questions linked to APIs in higher need receive higher quality answers.

> **Research Answer 1**: Questions referencing new APIs receive more answers. For questions with a first good answer within 2 days, questions referencing new APIs receive slower answers, which can be mitigated or nullified by low documentation need; for questions with a first good answer after 2 days, questions referencing new APIs receive faster answers. We see no significant effect of new APIs in identifying answer quality; however, APIs with higher documentation need are more likely to receive a higher quality answer.

RQ2: What Are the Effects of Bounties?

In regards to the bounty, we see net beneficial effects across the board. We see that questions with a bounty receive more (0.203, Table 2) and faster (-1.240, Table 4) answers, with higher quality (0.593, Table 5). With respect to number of answers, however, the positive effect can be mitigated by APIs with higher documentation need

Table 5. Answer quality models

	Dependent variable:		
	Answer Quality Label (Bad, Good)		
	(1)	(2)	(3)
QCreationDate	-0.509^{***}	-0.294^{***}	-0.163^{***}
TimeToAnswerMins	-0.095^{***}	-0.064^{***}	-0.092^{***}
QOwnerNQ	-0.050^{***}	-0.041^{***}	-0.001
QOwnerReputation	0.203^{***}	0.176^{***}	0.136^{***}
QOwnerAge	0.018^{***}	0.024^{***}	0.032^{***}
QMEC	-0.114^{***}	-0.074^{***}	-0.068^{***}
ABodyNWords		0.150^{***}	0.142^{***}
ABodyCharsOfCode		0.042^{***}	0.048^{***}
ANSwitches		0.105^{***}	0.107^{***}
ABodyURLCount		0.107^{***}	0.075^{***}
AMEC		1.679^{***}	1.638^{***}
AOwnerReputation		0.147^{***}	0.138^{***}
QNTags			0.020^{**}
QTitleLength			-0.106^{***}
QBodyNWords			-0.085^{***}
QBodyCharsOfCode			-0.018^{***}
QNSwitches			0.002
QNComments			-0.088^{***}
QBodyURLCount			-0.030
F.QQualityLabelGood			0.764^{***}
QNeed			0.032^{***}
F.Added			-0.018
APIDiffTime			0.00003
F.Bounty			0.593^{***}
TimeToBounty			0.001
Constant	2.638^{***}	-1.350^{***}	-1.429^{***}
AUC	0.64	0.69	0.71
Note:			*p<0.1; **p<0.05; ***p<0.01

(−0.014, Table 2), though only slightly - the minimal benefit considering **F.Bounty** and **QNeed** jointly is $0.203 + (-0.014 * 3.912) = 0.15$, still a net positive. Looking further at our time to first good answer model (Table 4), we see that bounty, with all other variables constant, decreases time to first good answer by a factor of $e^{-1.240} = 0.289$, or by 71%. As shown in Fig 3, the bounty also has the effect of flattening the long tail of answer times, with a larger density towards smaller values of time. As discussed above, only 46% of questions without a good answer within 2 days are answered within 1 month; flattening of the long tail helps combat this issue. The stated goal of the bounty is to draw more attention to the bountied question - this is in hopes that the question asker will receive help due to the added attention. Here, we see

that the bounty is effective in not only reducing the time to first good answer on average, but also in reducing the tail weight of the distribution of answer times. Thus, the bounty is a powerful tool in achieving the benefits we discuss here.

Fig 3. Time-to-answer density for non-bountied and bountied questions.

We also tested the inclusion of bounty amount into the models to see if higher reputation value bounties receive benefits compared to lower reputation value bounties (not shown in tables). In all forms individually tested (raw numeric 50, 100, *etc.*, scaled numeric 1, 2, *etc.*, factors for each raw value, and a binary factor of 50 vs. more than 50), there was no significant effect of bounty amount on any predictors tested.

> **Research Answer 2**: Questions with a bounty receive more, faster, and higher quality answers than those without a bounty. We find no significant effect of bounty amount on any predictors tested, for any operationalization of bounty amount in each model.

RQ3: What Are the Lessons Learned?

Finally, we address what a Stack Overflow question asker can do in order to attain any benefits from the the models in this work. In all cases, adding a bounty has a beneficial effect. As discussed above, the bounty amount has no significant effect for any predictors tested, for any operationalization of bounty amount tried. In other words, it is relatively safe to put the lowest bounty amount possible on a question to attain the benefits discussed.

With respect to answer quality, using more tags leads to a higher likelihood of receiving a good answer (0.020). The effect of more tags is slower answers for those questions answered within 2 days (0.071), but faster answers for questions answered after 2 days (-0.060). If a question asker believes their question is harder to answer, then adding more tags may lead to higher quality and faster answer.

In terms of documentation need, APIs in higher need receive higher quality answers (0.032), though slower answers for those questions answered within 2 days (0.045). This is good news for Stack Overflow, as although answers may come slower for underdocumented APIs, the answers they do receive are of higher quality.

Research Answer 3: Adding a bounty has a net beneficial effect across the board, while there are both specific benefits and disadvantages to adding more tags. Bounty amount has no significant effect on any predictors tested.

Threats and Conclusions

In this study we considered Stack Overflow questions referencing Android APIs and analyzed the number of answers a question gets, how fast these answers come, and their quality. Specifically, we sought to elucidate the factors affecting latency of answers and their quality when the questions refer to new Android APIs. We found that questions referencing new APIs receive more answers, after controlling for confounds, but there are several things hiding in this overall result. Namely, among questions with a first good answer within 2 days, those referencing new APIs receive slower answers, although that can be mitigated or even nullified by low documentation need. On the other hand, among questions with a first good answer after 2 days, questions referencing new APIs receive faster answers. One major reason for this is that questions with a bounty receive benefits across all variables of interest, and bounties can only be placed after 2 days. Based on these findings, we discussed what a question asker can do to receive the benefits discussed - in general, adding a bounty is the most efficient way to receive more, faster, and higher quality answers. However, the bounty reputation value does not appear to be significant in any context. To our knowledge this is the first study that specifically focuses on new APIs, and we use a novel metric to determine documentation need. These two points provide different descriptions of Stack Overflow's ability to maintain currentness in terms of API documentation. We have identified both shortcomings and places where Stack Overflow excels in terms of maintaining currentness, and show that the bounty indeed accomplishes its intended effect.

We acknowledge some threats to validity. Splitting the time-to-answer models into two parts may not be the best solution, as those phenomena likely overlap. We do this for a number of reasons. To properly study the effect of the bounty, we need compare only within bounty eligible questions. As a question cannot receive a bounty until 2 days after it is asked, it would be appropriate to include questions in this model that are fully or adequately answered before 2 days, since these questions would likely never receive a

bounty. In addition, we sought to split the question groups into those that are "easy" and those that are "hard" to answer. This labeling task is challenging, and past work has defined more difficult questions as those which take longer to answer [49]. Thus, by splitting the models at the 2 day mark, we accomplish both the aforementioned goals.

One may question our model validity because of the relatively low R^2 values in the time-to-answer models. In ordinary least squares regression, R^2 measures the percentage of variance captured by a model. However, a low R^2 alone does not mean that the model cannot be inferred from [50–53]. The phenomenon we are modeling is a difficult one to fully capture – most questions are either answered very quickly, or reside in a very long tail; the range of values is large, but is heavily concentrated towards lower values. The differences between values within the heavy concentration is very small, and thus hard to model. We control for many factors that we believed may contribute in describing the variance in time-to-answer, guided by prior research. We also took great care to ensure that our models meet the assumptions of OLS regression by performing standard model diagnostics, and thus are still useful for inference, even if the R^2 values may be considered low.

Acknowledgements

We acknowledge Premkumar Devanbu for helpful discussions on the direction of this work. We are also grateful to DECAL lab members for their patience and insights during various conversations about this project.

References

1. Treude C, Figueira Filho F, Cleary B, Storey MA. Programming in a socially networked world: the evolution of the social programmer. The Future of Collaborative Software Development. 2012; p. 1–3.

2. Barua A, Thomas SW, Hassan AE. What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering. 2014;19(3):619–654.

3. Campos EC, de Souza LB, Maia MdA. Nuggets Miner: Assisting Developers by Harnessing the StackOverflow Crowd Knowledge and the GitHub Traceability. Proc CBSoft-Tool Session. 2014;.

4. Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM; 2014. p. 102–111.

5. Vasilescu B, Serebrenik A, Devanbu P, Filkov V. How social Q&A sites are changing knowledge sharing in open source software communities. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM; 2014. p. 342–354.

6. Treude C, Barzilay O, Storey MA. How do programmers ask and answer questions on the web?: Nier track. In: Software Engineering (ICSE), 2011 33rd International Conference on. IEEE; 2011. p. 804–807.

7. Parnin C, Treude C, Grammel L, Storey MA. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. Georgia Institute of Technology, Tech Rep. 2012;.

8. Kavaler D, Posnett D, Gibler C, Chen H, Devanbu P, Filkov V. Using and asking: Apis used in the android market and asked about in stackoverflow. In: Social Informatics. Springer; 2013. p. 405–418.

9. Jiau HC, Yang FP. Facing up to the inequality of crowdsourced API documentation. ACM SIGSOFT Software Engineering Notes. 2012;37(1):1–9.

10. Ponzanelli L, Mocci A, Bacchelli A, Lanza M, Fullerton D. Improving low quality stack overflow post detection. In: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE; 2014. p. 541–544.

11. Dalip DH, Gonçalves MA, Cristo M, Calado P. Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow. In: Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval. ACM; 2013. p. 543–552.

12. Cavusoglu H, Li Z, Huang KW. Can gamification motivate voluntary contributions?: the case of stackoverflow Q&A community. In: Proceedings of the 18th ACM Conference Companion on Computer Supported Cooperative Work & Social Computing. ACM; 2015. p. 171–174.

13. Low JF, Svetinovic D. Data analysis of social community reputation: Good questions vs. good answers. In: Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on. IEEE; 2015. p. 1193–1197.

14. Movshovitz-Attias D, Movshovitz-Attias Y, Steenkiste P, Faloutsos C. Analysis of the reputation system and user contributions on a question answering website: Stackoverflow. In: Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on. IEEE; 2013. p. 886–893.

15. Grant S, Betts B. Encouraging user behaviour with achievements: an empirical study. In: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE; 2013. p. 65–68.

16. Parnin C, Treude C. Measuring API documentation on the web. In: Proceedings of the 2nd international workshop on Web 2.0 for software engineering. ACM; 2011. p. 25–30.

17. Jin Y, Yang X, Kula RG, Choi E, Inoue K, Iida H. Quick trigger on stack overflow: a study of gamification-influenced member tendencies. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press; 2015. p. 434–437.

18. Bosu A, Corley CS, Heaton D, Chatterji D, Carver JC, Kraft NA. Building reputation in stackoverflow: an empirical investigation. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press; 2013. p. 89–92.

19. Slegers J. The decline of Stack Overflow. Hackernoon. 2015. Available from: `https://hackernoon.com/the-decline-of-stack-overflow-7cb69faa575d`.

20. Slag R, de Waard M, Bacchelli A. One-day flies on stackoverflow-why the vast majority of stackoverflow users only posts once. In: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on. IEEE; 2015. p. 458–461.

21. Posnett D, Warburg E, Devanbu P, Filkov V. Mining stack exchange: Expertise is evident from initial contributions. In: Social Informatics (SocialInformatics), 2012 International Conference on. IEEE; 2012. p. 199–204.

22. Chua AY, Banerjee S. So fast so good: An analysis of answer quality and answer speed in community Question-answering sites. Journal of the American Society for Information Science and Technology. 2013;64(10):2058–2068.

23. Bhat V, Gokhale A, Jadhav R, Pudipeddi J, Akoglu L. Min (e) d your tags: Analysis of question response time in stackoverflow. In: Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on. IEEE; 2014. p. 328–335.

24. Goderie J, Georgsson BM, van Graafeiland B, Bacchelli A. Eta: Estimated time of answer predicting response time in Stack Overflow. In: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on. IEEE; 2015. p. 414–417.

25. Linares-Vásquez M, Bavota G, Di Penta M, Oliveto R, Poshyvanyk D. How do api changes trigger stack overflow discussions? a study on the android sdk. In: proceedings of the 22nd International Conference on Program Comprehension. ACM; 2014. p. 83–94.

26. Anderson A, Huttenlocher D, Kleinberg J, Leskovec J. Discovering value from community activity on focused question answering sites: a case study of stack overflow. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2012. p. 850–858.

27. Berger P, Hennig P, Bocklisch T, Herold T, Meinel C. A Journey of Bounty Hunters: Analyzing the Influence of Reward Systems on StackOverflow Question Response Times. In: Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on. IEEE; 2016. p. 644–649.

28. APKTool, a tool for reverse engineering Android APK files. 2017. Available from: https://ibotpeaches.github.io/Apktool/.

29. Doclet Overview. 2017. Available from: http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/doclet/overview.html.

30. van Dijk D, Tsagkias M, de Rijke M. Early detection of topical expertise in community question answering. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM; 2015. p. 995–998.

31. Yang L, Qiu M, Gottipati S, Zhu F, Jiang J, Sun H, et al. Cqarank: jointly model topics and expertise in community question answering. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. ACM; 2013. p. 99–108.

32. Thongtanunam P, Kula RG, Cruz AE, Yoshida N, Ichikawa K, Iida H. Mining history of gamification towards finding expertise in question and answering communities: experience and practice with Stack Exchange. The Review of Socionetwork Strategies. 2013;7(2):115–130.

33. Zhou G, Zhao J, He T, Wu W. An empirical study of topic-sensitive probabilistic model for expert finding in question answer communities. Knowledge-Based Systems. 2014;66:136–145.

34. Yang J, Tao K, Bozzon A, Houben GJ. Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In: User Modeling, Adaptation, and Personalization. Springer; 2014. p. 266–277.

35. Manning CD, Surdeanu M, Bauer J, Finkel JR, Bethard S, McClosky D. The Stanford CoreNLP Natural Language Processing Toolkit. In: ACL (System Demonstrations); 2014. p. 55–60.

36. Nasehi SM, Sillito J, Maurer F, Burns C. What makes a good code example?: A study of programming Q&A in StackOverflow. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE; 2012. p. 25–34.

37. Baltadzhieva A, Chrupała G. Question Quality in Community Question Answering Forums: a survey. ACM SIGKDD Explorations Newsletter. 2015;17(1):8–13.

38. Tian Q, Zhang P, Li B. Towards Predicting the Best Answers in Community-based Question-Answering Services. In: ICWSM; 2013.

39. Shah C, Pomerantz J. Evaluating and predicting answer quality in community QA. In: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. ACM; 2010. p. 411–418.

40. Gantayat N, Dhoolia P, Padhye R, Mani S, Sinha VS. The synergy between voting and acceptance of answers on stackoverflow, or the lack thereof. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press; 2015. p. 406–409.

41. Ravi S, Pang B, Rastogi V, Kumar R. Great Question! Question Quality in Community Q&A. ICWSM. 2014;14:426–435.

42. Stack Overflow. Enable automatic deletion of old, unanswered zero-score questions after a year. 2015. Available from: `http://meta.stackexchange.com/questions/78048`.

43. Cameron AC, Trivedi PK. Regression analysis of count data. vol. 53. Cambridge university press; 2013.

44. Bates DM. lme4: Mixed-effects modeling with R. URL http://lme4 r-forge r-project org/book. 2010;.

45. Vandekerckhove J, Matzke D, Wagenmakers EJ. Model comparison and the principle of parsimony. 2014;.

46. Cohen J, Cohen P, West SG, Aiken LS. Applied multiple regression/correlation analysis for the behavioral sciences. Routledge; 2013.

47. Vuong QH. Likelihood ratio tests for model selection and non-nested hypotheses. Econometrica: Journal of the Econometric Society. 1989; p. 307–333.

48. Yao Y, Tong H, Xie T, Akoglu L, Xu F, Lu J. Want a good answer? ask a good question first! arXiv preprint arXiv:13116876. 2013;.

49. Hanrahan BV, Convertino G, Nelson L. Modeling problem difficulty and expertise in stackoverflow. In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion. ACM; 2012. p. 91–94.

50. Schmidt FL, Hunter JE. Methods of meta-analysis: Correcting error and bias in research findings. Sage publications; 2014.

51. Hu M. What does it mean to have a low R-squared? A warning about misleading interpretation. 2014. Available from: http://humanvarieties.org/2014/03/31/what-does-it-mean-to-have-a-low-r-squared-a-warning-about-misleading-inte #more-3185.

52. Birnbaum P. On correlation, r, and r-squared. 2006. Available from: http://blog.philbirnbaum.com/2006/08/on-correlation-r-and-r-squared.html.

53. Birnbaum P. r-squared abuse. 2007. Available from: http://blog.philbirnbaum.com/2007/10/r-squared-abuse.html.

Figure1

Click here to download Figure Fig1.tif

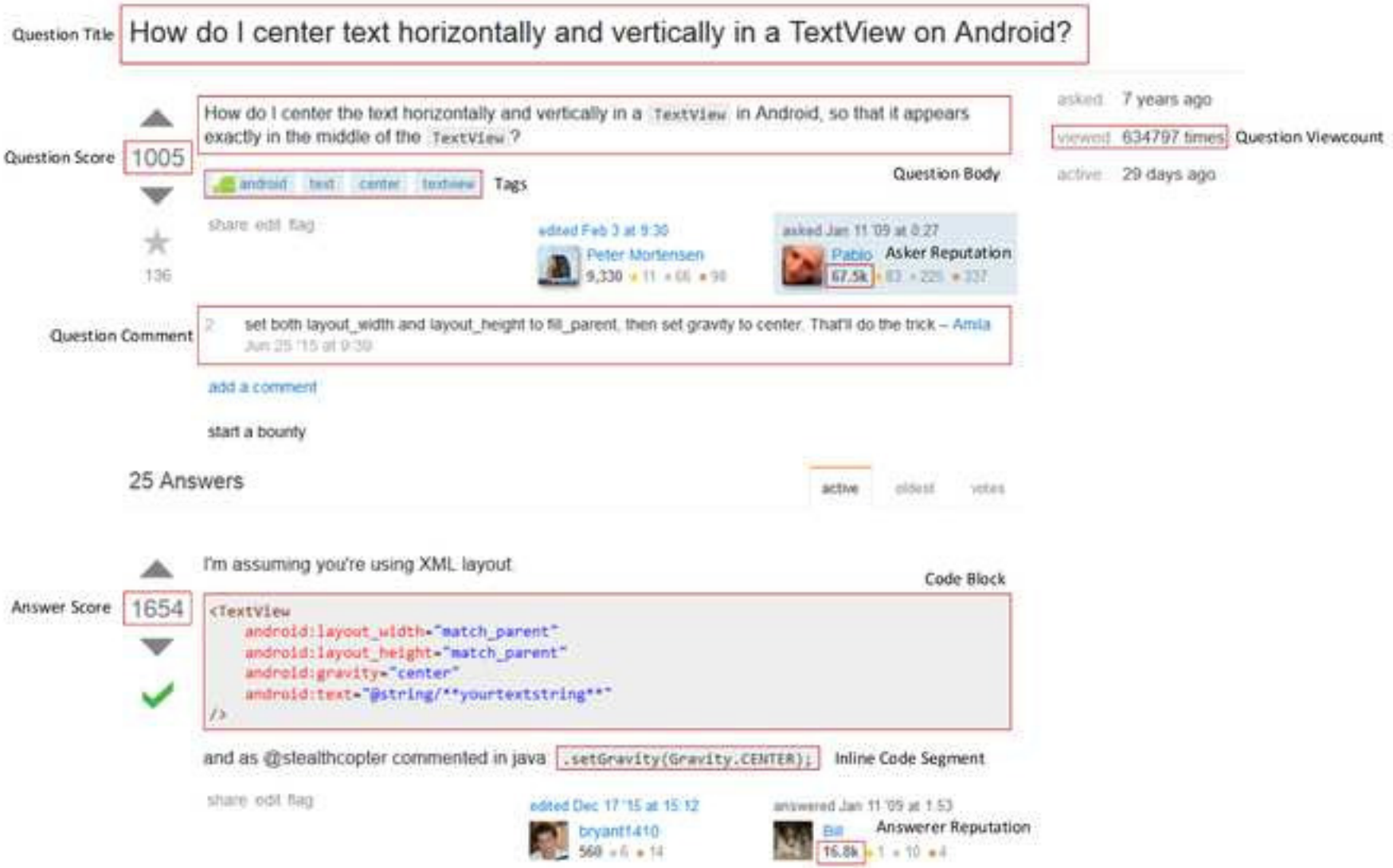

Figure2

Click here to download Figure Fig2.tif

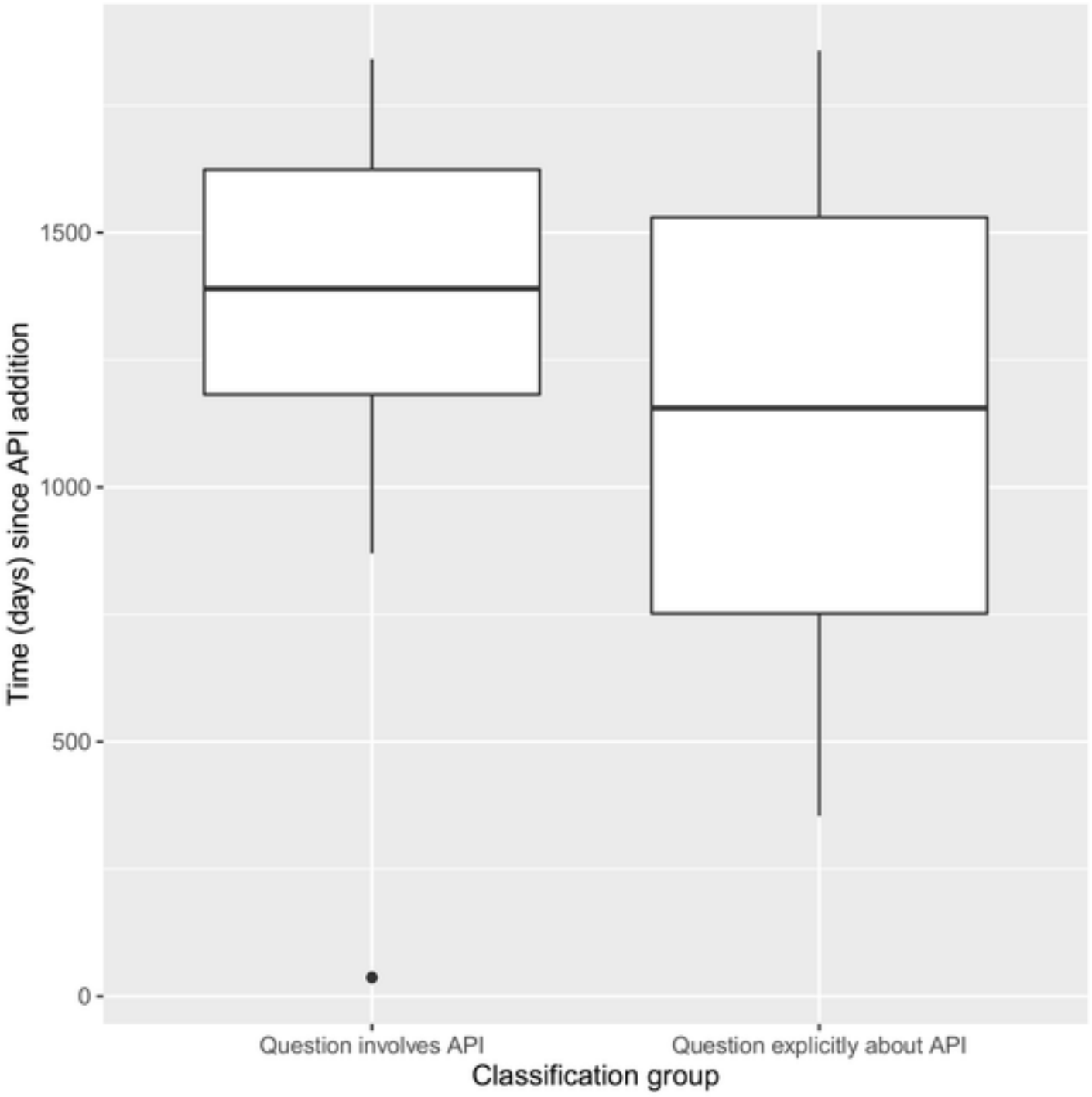

Figure3

Click here to download Figure Fig3.tif