Recommending Posts Concerning API Issues in Developer Q&A Sites

Wei Wang, Haroon Malik, and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
Email: {w65wang, hmalik, migod}@uwaterloo.ca

***Abstract*—API design is known to be a challenging craft, as API designers must balance their elegant ideals against "real-world" concerns, such as utility, performance, backwards compatibility, and unforeseen emergent uses. However, to date, there is no principled method to collect or analyze API usability information that incorporates input from typical developers. In practice, developers often turn to Q&A websites such as stackoverflow.com (SO) when seeking expert advice on API use; the popularity of such sites has thus led to a very large volume of unstructured information that can be searched with diligence for answers to specific questions. The collected wisdom within such sites could, in principle, be of great help to API designers to better support developer needs, if only it could be collected, analyzed, and distilled for practical use.**

In this paper, we present a methodology that combines several techniques, including social network analysis and topic mining, to recommend SO posts that are likely to concern API design-related issues. To establish a comparison baseline, we introduce two more recommendation approaches: a reputation-based recommender and a random recommender. We have found that when applied to Q&A discussion of two popular mobile platforms, Android and iOS, our methodology achieves up to 93% accuracy and is more stable with its recommendations when compared to the two baseline techniques.

I. Introduction

Modern software developers rely heavily on pre-defined libraries, frameworks, and services to implement much of the infrastructure of their systems. The application programming interface, or API, not only defines the services that are provided, it also specifies the protocols for their use. An effective API allows service providers, such as Google, Apple, Amazon, and Facebook, to give their clients controlled access to vast amounts of data and powerful building blocks in creating their own third-party systems; the explosion in this style of development have given rise to what is called the "API economy" [24].

Since the business success of an API provider ties closely to the existence of a large and enthusiastic user base, it is key that such APIs be carefully maintained to be responsive to the needs of the developers. As "issues" relating to bugs, performance, or usability are identified by the user community, the API provider/designers must strive to fix bugs quickly, accommodate changing requirements, support backward compatibility, help to overcome "learning obstacles", improve documentation, and support unexpected usages [34], [16], [31], [12]. For example, Linares et al. have studied how bug-free, stable APIs can affect successful application development within the Android technical ecosphere [19].

A starting point in addressing API-related issues is to quickly identify their context and underlying causes. API-related issues can be inferred by mining bug repositories [42], newsgroups [15], programmer posts [33], and e-mails [14]. Q&A websites, such as Stack Overflow (SO), hold particular promise as they contain discussions of the real-world issues encountered by millions of developers.

While the information contained within Q&A sites may be invaluable to API designers, these sites are designed to support the discussion and subsequent browsing of very detailed design questions rather than summarizing the knowledge contained within; additionally, much of the volume of API-related discussion concern repetitive "newbie" questions that are generally less informative to API designers. Consequently, if API designers want to infer API-related issues from posts in Q&A sites, posts must be analyzed, organized, and filtered. Recent empirical work on exploring the contents of SO has concentrated on the developer's role: identifying the contributions of individual developers [26], crowd-sourced documentation [36], knowledge dissemination [11], and the evolution of the software system [14]. Our work here aims to assist API designers to identify SO posts that are relevant to them, to better support their goals.

In this paper, we present a methodology that distills and ranks posts of Q&A sites that are likely to concern API-related issues, so that these posts can be studied in detail by API designers. More specifically, we make the following contributions:

1) **We present an approach to assist API designers finding API issues faced by the SO community.** Currently, API designers may spend considerable time skimming through all SO posts to identify API issues. Our approach uses textual data (questions and answers) and meta-data (such as "posted-by", "posted-time", and "#-of-views") as input. The output is a set of SO posts that are ranked by the perceived severity of API issues.
2) **We present a technique to identify experts on Q&A websites.** The majority of the important questions and accepted answers (pointing to bugs and API usability issues) are provided by expert members [6]. We automatically identify experts by modeling the Q&A forum as a directed graph, where each user is represented by a

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.28

TABLE I: Data corpus under study

	Date	Distinct Users	Total Posts	Accepted Answers	Total Answers
iOS	2011	17,808	51,192	35,846	93,333
Android	2011	28,330	90,132	54,764	157,864

node; and for a user B who answers a question posted by user A, a direct edge is linked from A to B. By leveraging the connectivity of the graph, we identify users who actively participate in SO discussions and related moderation activities, but do ask few questions themselves. We found the proposed model to be effective in removing up to 39% of non-technical posts, most of which are "how-to" questions, usually asked by novice developers.

3) **We show how generative models can be applied to reduce the dimensionality of SO data.** We use Latent Dirichlet Allocation (LDA) to reduce the dimensionality of SO posts, thereby allowing API designers to organize the posts along topics that capture the interests/concerns of a large developer base. Analyzing SO posts relative to the topic of interest allows designers to quickly uncover latent issues within the discussions.
4) **We present a simple statistical technique to expedite SO post selection.** SO posts that are answered late or quickly relative to the time of their original posting can be indicative of issues such as a known bug, confirmation for API usability issues, and a workaround for a buggy API. We use a Control Chart (CC) to automatically define what constitutes as 'late' or 'quickly' with respect to a topic's temporal trend.
5) **We propose a metric to rank SO posts.** To assess the difficulty of a question posed in SO, we have derived a formula that aggregates a number of different measures (i.e., the extent of experts' participation on a question, their resolution rates, and meta-data related to posts) that are likely to be indicative of the importance of the underlying issues. Furthermore, we rank the SO questions based on the perceived difficulty. A question with multiple answers, resolution pending, heavy expert participation, and high number of comments from SO community, is likely to contain a difficult API related issue and is of more interest to API designers in contrast to a question posted at the same time with a few answers.
6) **We compare our approach with two baseline recommendation techniques.** We performed a case study on posts related to the APIs of two popular mobile platforms: Apple's iOS and Google's Android. Manual validation of 1,200 posts threads (1,200 questions along with their answers) reveals that the proposed methodology has 34% less noise, (i.e., how-to and implementation-specific questions) and is more stable with its recommendations compared to the baseline recommendation techniques.

II. Background

A. Stackoverflow

Stackoverfow.com is a popular Q&A website dedicated to discussing questions related to programming, including general purpose algorithms, platform and language-specific features, and commonly used APIs. SO is actively used by millions of programmers to ask questions and discuss possible answers. Its contents are openly accessible to public, and posting a question generally requires only that the user have successfully created a free account on the SO website. Since users are registered, one can track the questions/answers on a per-user basis.

For each posted question, a user may include a title and textual description of the problem including code snippets in the body. Tags are used to organize the questions. Users must attach at least one tag and may attach up to five tags when asking a question. For each question, multiple answers can be given by different users. The user who asked the original question can then either post a summative comment or indicate one of the answers as correct. Other users can also rate whether they like either the questions and/or the answers.

The SO community itself is responsible for assuring the quality of the questions and answers: if a question or an answer is considered relevant, thorough, or correct, users "up-vote" it; if not, they "down-vote" it. Users who posted those up-voted questions/answers receive "reputation points". Consequently, building reputation within the community is a key motivator for individual contribution to SO.

At time of writing, the data and community of the SO are continuing to grow at an exponential rate; a total of 4,125,638 questions in 2009 [6] to 7,249,728 questions posted to date, 4.4 million answers, and 3.2 million comments. The site served more than 8.7 million visitors. After its first year, SO had around 53 thousand members. The membership grew rapidly and reached close to 1.3 million by 2012 [5] and 2.9 million by year 2013. The success of StackOverflow is fueled by a number of factors:

1) *High answer rate*: 92.6% of questions receive at least one answer [21]. This rate exceeds those reported from users of Yahoo! Answers (88.2% [13]) and KiN (66% [25]).
2) *Fast response*: The median time to the first answer for a question is 11 minutes; the median time to an answer accepted by the questioner is 21 minutes [21].
3) *Expert participation*: There is a heavy expert participation both from developers, designers, and architects.

B. Scope of Research

In this paper, we investigate methods to distill and rank Q&A posts with API-related issues. API-related issues span a spectrum of severity, from API bugs as the most severe issues to minor problems such as lack of proper API documentation. In this paper, we investigate only the five kinds of API-related issues that were investigated by an API usability model by Zibran et al.[42]. These API-related issues are: *1)* API correctness (for example, unexpected behavior of API[1]); *2)* memory management issues (some issues have been officially confirmed[2]); *3)* missing error handling (such as "errors in GPS lookup"[3]); *4)* backward compatibility issues (especially when platforms are upgrading quickly[4]); *5)* lack of proper API documentation (such as outdated documentation[5]). This scope of API-related issues is used both in establishing the ground truth and in assessing the performance of our proposed methodology. The details of deciding whether a Q&A post discusses any of the API-related issues can be found in Section VI-A.

C. Data corpus

For the purpose of assessing the performance of our proposed methodology for distilling and ranking Q&A posts, we conduct a case study on SO posts. This case study choose all Q&A posts with Android or iOS-related topics in 2011, as listed in Table I. We choose Android and iOS posts since both platforms have established a large developer community. Moreover, unlike other technical issues, we think posts with Android- or iOS-related topics are more likely to concern APIs compared to posts for a specific programming language. We perform our case study only on the posts logged in the year 2011 since both Android and iOS were in a relatively nascent stage where many API-related issues were yet to be fixed.

III. Analyzing StackOverflow posts: Current practice

API designers from major software platform providers, including Facebook [3] and Google [2], [1], already use SO posts to support API maintenance. API maintainers may participate in discussions to support platform developers, while API designers are continuously monitoring SO to capture posts containing issues that are likely candidates for corrective and perfective maintenance. Such issues include developer requests for new feature, APIs performance enhancement, usability issues and bugs.

When API designers wish to browse the SO content for issues perceived as problematic by the SO community, they may enter into what we call the "dark side" of the SO knowledge base, where there is no or little agreement on issues under discussion. For example, it makes sense that API designers may wish to first examine posts that are unanswered so far. Of course, an unanswered post may simply be new, poorly worded, or a duplicate of a common issue that has not yet been flagged as such. On the other hand, an unanswered question may also be indicative of a particularly subtle or complex question, or an issue that is rarely encountered.

In the second tier of importance might be questions with only a pending resolution, i.e., those with no generally accepted answer. This may be indicative of a well-considered question for which there is no clear and generally accepted answer. And in the third tier, API maintainers may look for late-answered questions, where the eventual answer appears long after the original question; this may also be indicative of issues that took thought and discussion to resolve on the part of the SO community.

[1]http://stackoverflow.com/questions/8044476/
[2]http://stackoverflow.com/questions/5358219/
[3]http://stackoverflow.com/questions/6487727/
[4]http://stackoverflow.com/questions/4658021/
[5]http://stackoverflow.com/questions/4922750/

IV. Analyzing StackOverflow posts: Challenges

We believe that the current practice of analyzing SO posts to capture maintainability issues is manual, error prone, and faces many challenges:

Uncertainty: Analyzing unanswered questions — those for which the SO community has so far failed to provide a consensus solution — seems at first glance to be a straightforward proxy for capturing API-related issues. However, using unanswered questions cause a few problems. Most importantly, an empirical study [5] indicates that most of these unanswered questions are how-to questions, largely from novice or low-reputation members. Questions at SO often remain unanswered due to the fact they are poorly phrased, the problem is too generic [21], [25], the issue is too new or not strongly relevant to most developers [5], or lacks sufficient visibility [10], [38]. These unanswered questions are often later deleted by moderators and experienced users after a timeout period of several months [11].

Similarly, many of the questions on SO ($\sim$31% [4], for instance, Android (241,987; 33%) and iOS (299,201; 45%) questions do not receive satisfactory answers because according to SO conventions, declaring an answer as "accepted" is a voluntary act. In practice, many SO members do not bother to formally endorse good answers as accepted; these users tend to be "information seekers", who have a single question but do not remain in the SO community once they acquire a workable answer. Also, since response turnaround in SO is very fast [21], information seekers and novice members often expect answers within a short time frame; if an answer is longer in forthcoming, they may simply abandon the posting [5].

Thus, we can see that a question that falls into the "dark side" may not capture a legitimate API-related issue that requires the attention of an API designer; however, the API designer must still wade through these posts. Filtering such questions — ideally, automatically — would thus be of clear benefit.

Large volume: The average number of Android related questions posted monthly on SO was 12,515 during 2012, and this grew to 19,767 by April 2014 [20]. This amounts to around 700 new questions (i.e., not including answers posts)

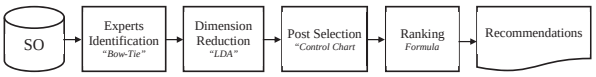

Fig. 1: Workflow of Proposed Methodology.

per day. The average ratio between question and answer posts of questions to answers is 1:3, resulting into 2,100 posts. The minimum average length of answer post is 3 lines of text [35]. Thus, an analyst must skim through more than 6,300 lines of textual contents to identify APIs that require perfective or corrective maintenance.

Limited time:

A. Internal pressure — Highly competitive markets and the consumerization of mobile apps have put pressure on API designers and maintainers to proactively provide updates and reduce release cycle times. For example, Android APIs are evolving at the rate 115 updates per month on average [23]; also, a major version of Android is released as often as every one to six months [18]. Identifying post-release API issues from crowd-sourcing is always the first step in an already tight maintenance schedule. Hence, managers are usually eager to reduce the time allocated for performing analytics of crowdsource information, i.e., Q&A website and allocate more time in fixing and testing API issues.

B. Slow process — In SO, the task of ensuring the quality of discussions is left to the crowd. Poor-quality posts are identified by community moderators, who have the power to close or delete questions. The process of deleting low-quality posts is determined by a voting mechanism and the review process to decide upon its quality is slow [29]. Around 80% of the questions takes at least a month to receive their first delete vote [11].

Risk of error: The process of analyzing SO posts is error-prone for several reasons. First, an API designer who is monitoring the posts may have only limited domain knowledge. Second, the underlying process for selecting posts of interest is largely manual, tedious, and so subject to human misjudgment. For example, the SO community is capable of generating rich source of both questions and answers about the Android APIs, covering 87% of the all APIs [10]. However, there is often no single person in a given evolving large software system who has complete and authoritative knowledge about the entire working of the system and its encompassing technical ecosphere. Consequently, for example, an API designer with good knowledge about the Android Location API may quickly uncover valid problematic issues related to maps, GPS, motion, position or environment sensors in SO posts. However, the API designer may fail to envision the criticality of many of the underlying issues in posts that relate to the telephony API. Further, SO provides only a limited set of tags for marking questions. Therefore, API designers may select a post for a more in-depth analysis by doing low-level textual searching for specific keywords like "fail", "error", "crash" and "usable". Of course, not all posts containing the "fail" and "error" are worth investigating; false positives are likely to be common. API designers may choose to follow domain trends, i.e., to look into only posts that are relevant to APIs with a history of heavy use, or to APIs that are known to be error prone.

Due to these challenges, we believe that the current practices for analyzing SO posts are grossly inadequate for current needs. API designers need an approach that is better designed to assist them to find posts concerning API-related issues in an authoritative, scalable, timely, accurate, and systematic manner.

V. Methodology

The main objective of our proposed methodology is to distill and rank SO posts so that those concerning API-related issues can be efficiently processed by API designers. Our approach takes SO posts and related meta-data as an input, and filters out most of the noise, i.e., how-to questions, closed questions, ill-posed, generic, and low-quality posts that fail to provide designers an insight to API-related issues. The goal is to produce a reduced set of posts that are more likely concern API issues, and that are ranked by the level of perceived difficulty by the SO community. Analyzing reduced, relevant, and ranked set of posts allows API designers to spent less time reading through raw text to find issues of interest to them, and thus allowing more time to work on solutions to API-related issues.

Figure 1 shows the high-level overview of the steps involved in our proposed methodology. The *first step* is to automatically identify the experts. Retaining only the question posts of experts eliminates how-to question from novice users.

The *second step* further reduces the dimensionality of SO posts by organizing them along topics that capture the interest/concern of a large developer base. This provides API designers with the grounds to delegate SO posts to relevant designers. Analyzing posts relative to the topic of interest allow designers and corresponding developers quickly grasp fine-grained issues of SO crowd.

The *third step* automatically identifies questions that are answered quickly or late with respect to temporal trends — such as the popularity of an API and available corresponding expertise at SO — within an observed time span. Questions answered quickly relative to the time of their original posting can be indicative of issues such as known bugs or confirmation of API usability issues. Identifying late questions with respect to temporal trends enables API designer to disseminate timely support to the developers.

The *last step* of our methodology ranks posts, by the level of difficulty-to-resolve as perceived by the SO community, based on a set of metrics derived from the SO meta-data. We now detail each step of our methodology.

A. Experts Identification

Identifying experts out of novice and beginner users eliminates most how-to questions. An empirical study [21] shows that beginner SO users are often "information-seekers", who

	Extracted	*Derived*
Posts	Post ID of Stack Overflow (PID)	No. of answers received
	Questions revised (Y/N)	Time it took to get a first answer
	No. of up votes received	Time it took to get an accepted answer
	No. of down votes received	Number of experts
	No. of user who viewed the post	
	Favorite	
Members	Member ID	Member's Avg. time to post answers
	Member reputation	Member's Avg. time to accept answers
	Total answers posted by a member	Resolution rate of a member
	Total accepted answers of a member	Reputation score of a member
		Member's dual expertise (Y/N)

TABLE II: Meta-data Variables

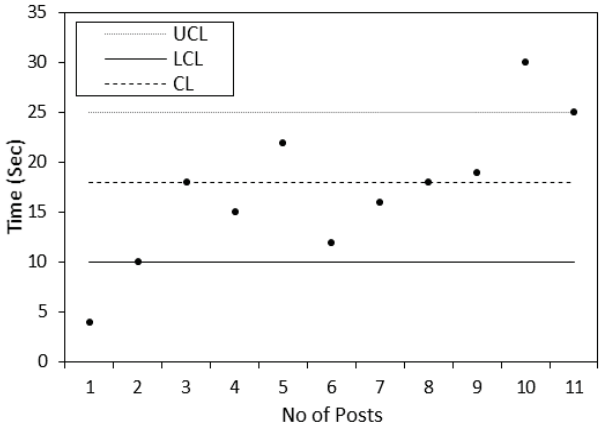

Fig. 2: A Sample Control Chart for Analyzing Stackoverflow Posts.

ask more how-to questions but provide few answers themselves. Novice SO users often ignore explicit guidelines on both posting and answering questions [5]. The majority of the important questions and accepted answers (pointing to bugs and API usability issues) are provided by expert users [6]. For Q&A sites, a straightforward proxy for expertise — the reputation of user accounts — reflects only in part the expertise since a reputation score is often associated with promptness of providing answers [8]. To overcome this limitation, we identify experts by "bow-tie" analysis proposed by Broder et al. [9]. The "bow-tie" analysis leverages connectivity of a question-answer graph for all users by characterizing a Q&A forum as a directed graph, where each user is represented by a node; and for a user B who answers a question posted by user A, a directed edge is constructed from A to B. By analyzing the connectivity of this graph G, we identify the largest strongly connected component C within G, which we call the "core" component; the core represents users who actively ask and answer questions for each other. After determining C, we can then quickly determine the set of users who only ask questions (i.e., all nodes that link to C, the "in" component in bow-tie structure), and the set of users who only answer questions (i.e., all nodes that are linked to by nodes in C, the "out" component in bow-tie structure).

We propose that the "out" component can be considered as experts, since these users answer questions of the active users ("core"), yet they do not ask questions to "core" users. A similar approach [40] was used to identify expertise in other types of social network relationship. For iOS-related posts, we identify 3,027 distinctive accounts belonging to the "out" component. For Android related posts, there are 5,098 distinctive account belonging to the "out" component. The posts related to "out" component only are retained in this step.

B. Dimension Reduction

This step of our methodology uses a topic modeling technique called Latent Dirichlet Allocation (LDA). We use LDA to further reduce the volume of SO post in a systematic manner, i.e., by finding dominant SO discussion topics and associated posts. LDA is a statistical topic modeling technique suitable for natural language documents such as SO posts. LDA represents topics as probability distributions over the words in the corpus. Moreover, it also represents topics as probability distributions over the discovered topics. LDA creates topic when it finds a set of words that tend to co-occur frequently in the documents of the corpus. Often, the words in a discovered topics are semantically related, which gives meaning to topic as whole, For example, the words with the highest probability in a topic might be "server", "client", "request", "host" and "network" (because these words tend to occur together in documents), indicating that this topic is related to database.

We use implementation of the LDA model provided by MALLET [22]. The quality of topics generated by LDA greatly depends upon its configuration parameters. Panichella et al. [28] provide an optimal configuration for LDA to work with text in software artifacts, which has different properties as compared to the natural language text in SO posts. In this paper, we use the following parameters based on our experimentation with LDA: 40 topics, 40 iterations (for convergence), and for the hyper parameters, we choose standard values used in the information retrieval community for natural language corpora: $\alpha = 0.01$ and $\beta = 0.01$. The low value of α enables us to discover dominant topics, i.e., high variability among topics. Each topic contains around 60 words, as we are able to label the topics effectively under this setting.

Due to the probabilistic nature of LDA, some topics are assigned small but non-zero (e.g., 0.01) probabilistic value. Therefore, we use δ threshold as a topic cutoff. We keep only the main topics which have a topic probability of 0.10 or higher. Each post is assigned probabilistic membership values for topics, as shown in Table III. We assigned each post to the topic for which it has the highest probabilistic membership value.

C. SO Post Selection

With a set of SO posts relative to topic of interest in hand, customarily, an analyst (i.e., an API designer) starts by analyzing unanswered posts first, followed by late-answered posts. The analyst uses his/her personal judgment to determine

Topic		*Post*	
Num	*Probability*	*ID*	*Membership Probability*
38	0. 2991	272584	0.765
20	0. 2821	308081	0.835
18	0. 2820	264140	0.698
	0. 1522	255846	0.532
21	0.1697	300048	0.788
	0.1216	300793	0.865
1	0.1092	313018	0.900
⋮	⋮	⋮	⋮
0	0.01221	327082	0.213

TABLE III: An example of topics with probabilistic membership values.

what constitutes as a late answer and filters out the respective SO posts. On the other hand, it is difficult to define what 'late' means in absolute terms and relatively depends upon the popularity of the APIs and a number of corresponding expertise available on SO.

An effective way to identify late answered question is to use 3-sigma (3σ) rule on SO meta-data. However, the temporal data about SO posts is not normally distributed (according to results of a Shapiro-Wilk test [32]). Therefore, we use a statistical quality control technique called *control charts* to expedite and automate the identification and selection of questions that are answered late. We use a control chart due to its previous success in analyzing software engineering related data [27]. In a control chart, the *control limits* represent the limits of variation that should be expected from answering questions at SO, within a given time span. Violation of control limits is considered as an anomaly (i.e., posts with late answers). Control limits include a Central Limit (CL), which is the median of all the value of a variable, *Upper Control Limit* (UCL) and *Lower Control Limit* (LCL) are the upper/lower limits of the range of a variable. A common choice is to use 10^{th} and 90^{th} percentiles to identify LCL and UCL [27].

We explain the use of a control chart to identifying late answers at SO with an example. Figure 2 shows the number of days it took to get an accepted answer <4,10,18,15,22,12,16,18,19,30,25> for eleven SO posts. Thus CL, LCL, and UCL are 18, 10, and 25, respectively. The violations (i.e., late answered posts) are with values that are greater than UCL (25) or smaller than its LCL (10). Hence, in this example, posts (1 and 10) are recommended for manual inspection since they violate the control limits.

One of the advantages of using a control chart is that not only it identifies the late answer post relative to the time span, it also pinpoints the posts that are answered very quickly. Ten minutes appear to be the minimum time for knowledgeable programmers to reply [21]. Thus, posts violating LCL can reveal issues such as a known bug confirmation of API usability issue, and a workaround for a missing API feature.

TABLE IV: Variables used in constructing Question Score (QS) metric

No.	Variables	Motivation and Rationale	Reputation
1	#Up Votes (UV)	A higher number of people liking the question implies that the topic of discussion is important to the community	15
2	#Down Votes (DV)	A higher number of people who dislike the question implies incorrectness. Down votes help to remove the noisy questions from SO.	125
3	#Comments Received (C)	A High number of comments implies popularity of a topic. Comments are mostly used to request clarification from the author of the questions, to offer suggestions to improve the post, and to add relevant but minor or transient information to a post.	50
4	#Answers (A)	Number of answers serve as a proxy to the importance of a question. A question with many answers yet no accepted answer implies that there may not exist a direct solution.	0
5	#Views (V)	A high view count implies popularity of a post.	0
6	Favorite count (F)	A high favorite count implies that the topic is of interest to SO members and the question invites continued attention from these members.	0
7	#Experts (E)	Experts tend to answer questions that require advanced knowledge. Heavy expert participation in post indicates the importance of the question and the severity of the problem (if exists). We did not use the reputation member score provided by stack overflow since the member score can be earned by answering simple questions or community contribution on Stack Overflow. Instead we derived the following metrics intuitive to describe the expert level.	Nil
8	Resolution Rate (R)	This metric reflects the ratio between the number of answers provided by expert and number of answers that get accepted.	Nil
9	Dual Expert (M)	This metric reflects whether a member has knowledge about more than one platform (in this paper: both Android and iOS). The dual expertise provides an edge to a member to leverage expertise for cross-platform questions.	Nil

D. Ranking

To further improve the cost-effectiveness for API designers in terms of maximizing their analytic potential, we also rank the posts based on the level of perceived difficulty by SO community. We consider that a higher level of difficulty implies that underlying issue is likely to be relatively serious. We propose a metric called Questions Score (QS) to model the question difficulty along the two dimensions listed in Table II, whereas Table IV lists the motivation and rationale behind selecting the variables used to construct the QS.

$$QS = P_s + M_s \quad \text{(i)}$$

Question Score (QS) is the sum of *Post Score* (Ps) and *Member Score* (Ms). We calculate the *Post Score* using the meta-variables shown in the following equation:

$$P_s = 15UV + 125DV + 50C + A + F + V + E \text{ (ii)}$$

We normalize the meta-data variables related to posts in equation (ii) to avoid any single variable from dominating the outcome of the post score. Take view count (V) as an example; it is normalized by the average of the number of view count of all the posts, as a result of applying the "bow-tie" model. To give credibility to our Ps metric, the meta-data variables in equation (ii) are assigned weights based on the value of reputation require to perform corresponding activities on SO [6]. The member score (Ms) is calculated using the equation below:

$$M_s = \sum_{\substack{\theta_i \in P \\ i=1}}^{n} R_{\theta_i}. \text{ (iii)}$$

In this equation, θ denotes an expert belonging to a post (P) and *R* is the resolution rate for an expert (which on average is 36% for Android and 34% for iOS). The value of *M* indicates the number of systems in which expertise is held. For example, if an expert has provided answers for both Android and iOS platforms on SO, the expert's *M* value is two.

VI. Case Study

To evaluate our methodology, we have performed a case study using all iOS and Android-related posts from 2011. For the purpose of evaluation, we establish a ground truth concerning API-related posts by manual examination. We also introduce two approaches to establish a baseline for evaluating the performance of our methodology.

- *A reputation-based approach* that recommends SO posts to API designers based on the reputation score of the member posting a question. This approach is inspired by a study [25] that found a correlation between members' reputation and the difficulty level of questions.
- *A random approach* that randomly selects SO posts and recommends them to the API Designer for inspection. The motivation here is to use the most basic recommendation approach for comparing the performance of our proposed recommendation methodology.

A. Establishing Ground Truth

To validate whether each post in the recommendation sets of each approach contains API-related issues, we randomly selected 600 SO posts for Android and 600 posts iOS (along with their answers) for manual categorization before running our case study. To decide whether a post contains an API issue, we create a taxonomy based on the API usability model proposed by Zibran et al.[42]. Specifically, we consider the following issues to be API related:

- **API correctness** — The discussion of the posts appears to reach consensus that a bug of a specific API is being discussed; usually, these threads also offer a work-around solution.
- **Memory management** — Developers report that the API uses excessive memory or has memory-related errors.
- **Missing error handling** — Developers agree that an API lacks appropriate error handling or diagnostics mechanisms.
- **Backward compatibility** — Users report that an updated API fails to deliver previously expected functionality.
- **Lack of proper API documentation** — Developers agree that the official API documentation is insufficient or misleading.

[6] http://stackoverflow.com/help/privileges?tab=creation

Season	System	Total Questions	Bow-Tie	LDA
S-1	iOS	9,132	885	858
	Android	17,380	1,192	1,114
S-2	iOS	12,283	885	858
	Android	22,497	1,432	1,360
S-3	iOS	14,641	902	883
	Android	24,752	1,843	1,750
S-4	iOS	16,136	995	965
	Android	25,503	1,380	1,324

TABLE V: Number of posts remained after each step.

We consider posts that do not fall into any of these categories to be unrelated to API issues, and thus are "noise". In practice, we found that most noisy posts are either "how-to" questions or implementation-specific questions.

Two of the authors performed a manual categorization of all posts. In cases where they did not agree, the categorization decision was made by a third party from outside the lab who was an experienced mobile application developer.

Finally, to gain a better granularity of the dataset, we account for data seasonality by splitting the Android and iOS SO posts from 2011 into four-month periods. This allows our approach to discover seasonal topics, to construct a recommendation set that can provide API designers with insights into the developers' interests and issues at a finer grain [37], and to provide grounds to evaluate the consistency of the approaches with their recommendations across seasons.

VII. Case Study Findings

Table V shows the number of posts retained after applying the expert identification and LDA. We report the viability of our proposed approach, and contrast its performance with that of the reputation-based and the random approaches, using top-200 posts in their corresponding recommendation sets, across three dimensions: *accuracy*, *stability*, and *overhead*.

A. Accuracy

The accuracy of a recommendation approach is the ratio of posts concerning API-related issues to the total number of posts in the recommendation set. The accuracy of the recommendation approaches, across seasons, is shown in the Figure 5. We found that our proposed approach achieved up to 93% accuracy. By contrast, the recommendation set produced

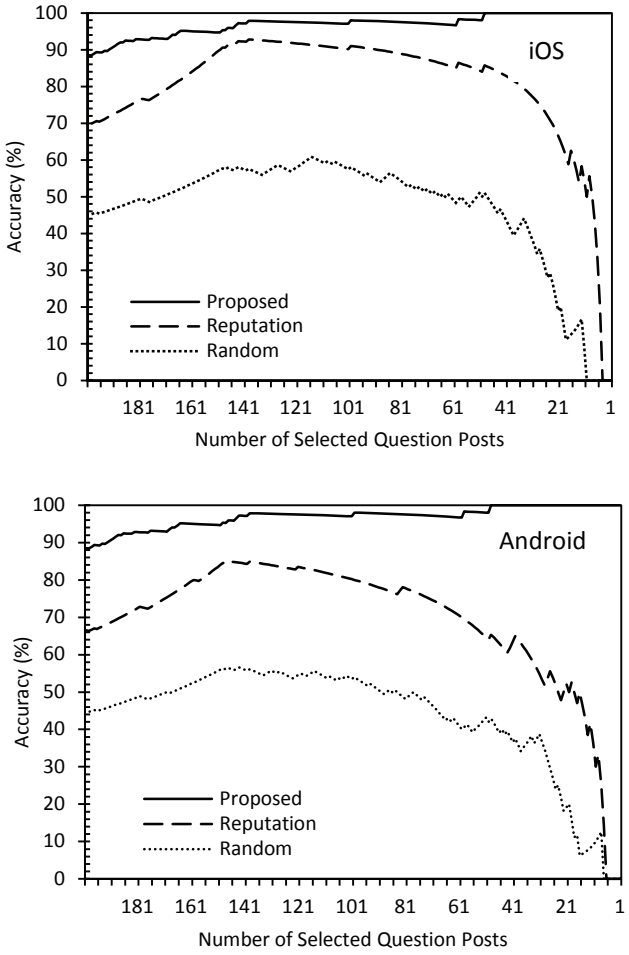

Fig. 3: Stability of three approaches for iOS and Android

by reputation-based approach had 17% noise rate for iOS and 21% for the Android. As expected, the random approach was the least accurate, with up to 36% noise rate for iOS and 41% noise rate for Android.

In our investigation, we found that the noisy posts identified in the recommendation sets by the proposed methodology were primarily implementation-specific questions having a high number of views, QS score, and member reputation. By comparison, on average across the four seasons 31% posts for iOS and 39% posts for Android, identified in the recommendations set of the random approach were noise and had the low QS score, and member reputation. Upon manual inspection, we found that most members responsible for noisy posts are information seekers, who confessed to be a novice or beginner (see Table VI).

The recommendation set by the proposed approach has 34% less noise compared to the random approach. We found that authors of noisy posts are often information seekers, who confessed to be novices (see Table VI.) The proposed approach successfully eliminated most of the SO posts of the novice members.

B. Stability

Stability is defined as the ability of a methodology to maintain its accuracy when reducing the size of its recommendation set. A stable method is preferred as it guarantees accuracy while shrinking the size of a recommendation set.

To investigate the stability of our proposed approaches, we plotted the ratios for all three approaches while decreasing the size of their recommendation sets. For each approach, we found the results to be consistent across the seasons. Due to space constraints, we show only the stability graph for all the approaches when applied to the iOS and Android for the first season in Figure 3. This figure shows that the proposed approach is more stable than the 'reputation-based' and the 'random' approach. This means a change in the size of its recommendation set does not drastically affect the proportion of the posts of interest.

Figure 3 also shows that for both Android and iOS, the line trajectory of the proposed methodology exhibits a smooth increase when shrinking recommendation set. By comparison, the line trajectory of the reputation-based approach and that of random approach both exhibit a sharp increase in robustness over the removal of last sixty posts and then and then gradually taper off. For both techniques, there are also drastically sharp trajectory drops for the stability curve while reducing the recommendation set size beyond the first thirty posts.

Such instability of the reputation-based and the random approaches with respect to reducing the recommendations set of posts affects their practicality. As a first step to explore the stability differences between the approaches, we looked into the text and the attribute values (Table II) associated with the posts. We find that the proposed approach is stable due mainly to its post selection mechanism, which filters out noisy posts, while retaining relevant posts at the top. We examined the last sixty posts — i.e., the tail — in the recommendation sets of the proposed methodology, along all the seasons and for both the Android and iOS. We found that tail contains around 5% noise for iOS (4% implementations specific questions and 1% how-to questions 1%) and about 9% for Android (8 % implementations specific questions and 1% how-to questions).

In case of the reputation-based approach, the tail consists of 18% implementation-specific questions and 5% "how-to" questions for both iOS and Android. For the random approach, the tail contains about 32% "how-to" questions and about 11% implementation questions. In Figure 3, the removal of noisy posts in the tails resulted in a sharp increase in stability curves for the reputation-based and random approach. To better understand the rationale behind a drastic drop of the stability curves for both reputation-based and random approaches, we analyzed the first thirty posts in their corresponding recommendation sets. For both iOS and Android, we found that

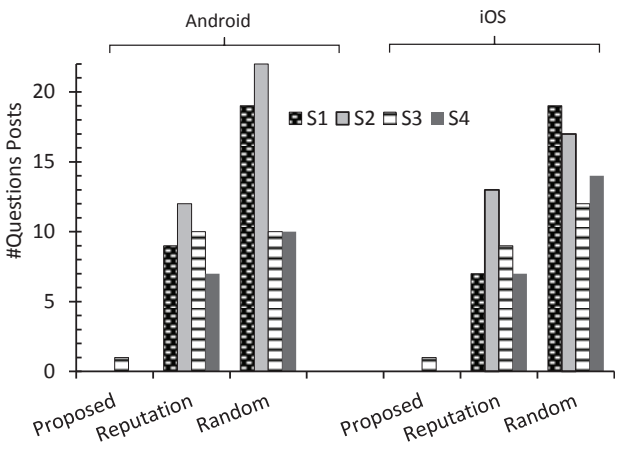

Fig. 4: Number of overhead posts

noise accounted for 41% of the posts recommended by the reputation-based approach, and 59% for the random approach. More specifically, for the reputation-based methodology, we found 37% of the first thirty post were implementation-specific questions from high-reputation members; often, these members would later provide the eventual answer also, which was also highly viewed.[7] We noted this to be a recurring behavioral pattern: An expert developer finds an interesting problem, and then later answers it authoritatively; they tag their own answer as acceptable, thereby also increasing their own reputation.[8] We found this behavior was not only acceptable, but officially encouraged; there are even SO badges for it.[9].

We observe that the random approach contains 52% "how-to" questions among the first thirty posts in its recommendation set. Since there was no robust filtering and ranking mechanism attached with these two baseline approaches, the arbitrary removal of the post attributed to the drastic decline of robustness trajectory.

> The proposed methodology is highly stable due to its post selection mechanism, which filters out noisy posts and ranks them in an order that is likely useful to API designers.

C. Overhead

Overhead is defined as the number of posts that are ranked above the *first* post with API-related issue by our ranking method. Measuring overhead helps to validate the proposed ranking approach of the recommendation set. Figure 4 shows the overhead incurred for the four seasons for each approach.

[7]http://stackoverflow.com/help/self-answer

[8]http://meta.stackexchange.com/questions/17463/can-i-answer-my-own-questions-even-if-i-knew-the-answer-before-asking

[9]http://blog.stackoverflow.com/2011/07/its-ok-to-ask-and-answer-your-own-questions/

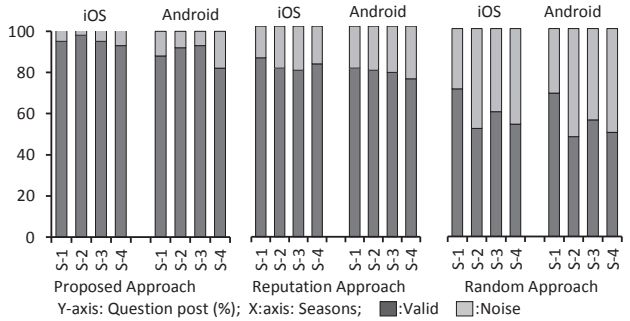

Fig. 5: Accuracy of three distilling approaches for four seasons

TABLE VI: Self-admitted novices confess in "how-to" posts

Post ID	Question Post
7457029	*I'm a beginner Xcode developer and using the Xcode 4 trying follow a Xcode 3.2 tutorial the IDE show me the error.*
5210535	*I'm new to iOS and Objective-C and the whole MVC paradigm and I'm stuck...*
5331697	*I'm a beginner Objective-c programmer and stuck at this point. I created an app from the View Template, then edited the view.*
4602739	*I am new to android, when I create a project with 1.5 to 2.1, I gives me the same error that proguard is missing from there path.*

According to Figure 4, the proposed approach incurred up to 1% of overhead for either Android or iOS. For both projects, the first two questions in the recommendation set of the third season (S3) were implementation-specific questions. The random technique incurred the most overhead: 11% (22 noise posts) followed by the reputation-based approach with up to 6% (12 noise posts).

> There is little overhead associated with the use of the proposed recommendation methodology to support API designers quickly find posts that are likely to concern API-related issues.

VIII. Related Work

Stack Overflow has been extensively studied and analyzed for a wide variety of empirical studies. Treude et al. [38] are among the pioneers who analyzed SO posts. In their study, they manually investigate a few hundred posts and explore the factors to well-answered questions. Nesehi et al. [26] analyze questions on SO to understand the quality of code examples. They find nine attributes of a good questions like concise code, link to extra resources and online documentation. Similarly, Ravi et al. [30] use LDA to automatically predict the quality of questions based on their contents at SO. Both Nesehi et al. [26] and Ravi et al. [30] focus on helping developers construct a good question that can attract large number of answers from SO community. In contrast, our work focuses on helping API designers pinpoint posts that are likely to be indicative of API-related issues instead of improving presentation quality. Barua

et al. [7] use LDA to automatically infer the main topics of discussion by developers at SO. Similarly, Wang and Godfrey [39] analyze iOS and Android developer questions on SO to detect API usage obstacles. Similar to our work, they also used topic models to summarize possible scenarios from all the posts related to previously identified API classes.

Kintab et al.[17] propose a recommendation system which identifies expert developers by mining both source code repositories and social networks. Zhou et al. [41] propose methods that use machine learning techniques to provide content categorization for SO. Such a categorization is important for SO moderators, since they need to assign pre-defined semantic categorizes to API discussion. Moreover, such content categorization of API discussion assists developers to search for useful information. Asaduzzaman et al. [5] conducted a quantitative study to categorize unanswered questions at SO. In their study, they used a fixed threshold (a month) to qualify a posted question as unanswered. Contrarily, our methodology dynamically constructs the threshold for identifying a late answered question taking into account the temporal trends. Work of Panichella et al. [29] is the closest to the work of this paper in a sense that they propose a classification approach to reduce the size of review queues by identifying posts that are misclassified as low quality posts by SO members with a high reputation. Similar to our work, they also rely on post metrics and related metadata to construct their classification model. However, their research goal is to provide relief to SO moderators and expert users from manually responsible for identifying and reviewing lower quality posts.

IX. Threats to Validity

External Validity: An external threat to the validity of our results is that we focus on a single website, Stack Overflow. Nevertheless, SO is currently one of the most popular and largest Q&A websites for software developers. Moreover, the generalizability of our findings could be regarded as one of the limitation. Nevertheless, the techniques used in our methodology have some overlaps with some other studies using SO data [3], [5], [38]. This suggests that our proposed methodology could be generalized to some extent.

Internal Validity:

- **Error and bias** — This study requires various sets of data analysis (data handling and graph analysis to construct "bow-tie" structure) and implementations (LDA for topic modeling and control chart). Therefore, we have tried to minimize the source of error and bias by the use of existing and mature tools (e.g., Mallet for implementing LDA) and packages (e.g., R statistic package for control chart implementation, and "bow-tie" structure analysis). Likewise, the majority of the steps of our methodology — such as expert identification, and post selection — was automated.
- **Human Intervention** — in this study, we rely on the manual effort to read the SO posts for the iOS and Android to identify API-related issues. Due to the subjectivity of manual effort, bias may be introduced into the study. To avoid such impact, we cross-validated the findings of the two developers, manually analyzing the SO posts. A third programmer settled disagreements on a few posts.

Conclusion Validity:

- **Imbalanced construction** — We randomly constructed and inspected small samples of SO posts to identify the underlying issues. However, when the sample size is small, randomization may lead to Simpson Paradox. We tried to mitigate the construct validity by: **i)** selecting random posts from SO dump without replacement to construct five samples, and **ii)** establishing an identical time horizon (four months) for all the samples. Despite, the immense difference between the approaches might be by chance due to the random nature of the sample. We plan to extend the study by: **i)** increasing the sample size of SO posts to confiscate Simpson paradox, and **ii)** use suitable statistical techniques to lower the risk of constructing a non-representative sample.
- **Irrelevancies in the dataset** — Treude et al. [38] report that nearly 5% of all SO posts are non-technical questions. We believe that the threat introduced by these non-technical posts are in large diminished since we filter out posts that do not contain API-related tags.
- **Poor tagging** — Another internal threat is that many SO users may use obscure or generic tags (e.g. "iOS") when posting questions relating only to one API class.
- **Optimal setting** — Choosing the optimal parameter setting for LDA has no known general solutions. Although, we have experimented with different values from the number of topics (K) and found ($K = 40$) to be the best for us, we still cannot be sure that our chosen value is optimal for the analysis we performed. Additionally, our cut-off threshold for document membership (δ) affects the result of our study.

Construct Validity: We have designed a new metric "QS" to rank the questions by order of their difficulty. Ranked questions provide a head start to API designers so that API designers can analyze difficult questions first. However, our proposed metric is based upon variables provided solely by SO. For these variables, their data quality or data integrity are therefore determined by SO. Any data management mistakes of SO may cause threats to this study.

X. Conclusions

Programmer Q&A websites such as StackOverflow serve as convenient and vast repositories of broad development experience and expertise. In particular, API designers who wish to understand how their work is perceived by the community at large would do well to mine these repositories for useful feedback on usability, bugs, and unexpected usage. Consequently, automated — or even semi-automated — techniques that can analyze the vast data and provide timely and high-quality recommendation sets of posts that discuss API-related issues would likely be of great value to API developers. In this paper,

we propose such an automated methodology, and evaluate its effectiveness against two baseline techniques. We find that the proposed approach is more accurate with its recommendation set with 41% less noise in comparison to baseline approaches. Moreover, due its post-selection mechanism, the proposed approach is more stable in the sense that the proportion of the post of interest is indepentant of the size of its recommendation set. Additionally, the proposed approach incurred a 1% of overhead while the overhead for the reputation-based and random approach are 11% and 6% respectively.

REFERENCES

[1] Ask a question about the google drive sdk. https://developers.google.com/drive/support, accessed in Jan. 2015.

[2] Google taps stackoverflow as offical andriod dev support for noobs. http://readwrite.com/2009/12/20/stackoverflow-android-support\#awesm=~oCRlpsKottvxVi, accessed in Jan. 2015.

[3] Supporting developers on stack overflow. https://developers.facebook.com/blog/post/545/, accessed in Jan. 2015.

[4] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec. Discovering value from community activity on focused question answering sites: A case study of stack overflow. KDD '12, pages 850–858, 2012.

[5] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider. Answering questions about unanswered questions of stack overflow. MSR '13, pages 97–100.

[6] K. Bajaj, K. Pattabiraman, and A. Mesbah. Mining questions asked by web developers. MSR '14, pages 112–121.

[7] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? An analysis of topics and trends in stack overflow. 19(3):619–654, 2014.

[8] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: An empirical investigation. MSR '13, pages 89–92, Piscataway, NJ, USA, 2013. IEEE Press.

[9] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Netowrking, pages 309–320, 2000.

[10] L. G. C. Parnin, C. Treude and M. Storey. Crowd documentation: Exploring the coverage and the dynamics of API discussions on stack overflow. 2012.

[11] D. Correa and A. Sureka. Chaff from the wheat: Characterization and modeling of deleted questions on stack overflow. WWW '14, pages 631–642, 2014.

[12] B. E. Cossette and R. J. Walker. Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. FSE '12, pages 55:1–55:11, New York, NY, USA, 2012. ACM.

[13] F. M. Harper, D. Moy, and J. A. Konstan. Facts or friends?: Distinguishing informational and conversational questions in social q & a sites. CHI '09, pages 759–768.

[14] A. Hindle, M. Godfrey, and R. Holt. What's hot and what's not: Windowed developer topic analysis. ICSM '09, pages 339–348, Sept 2009.

[15] D. Hou and L. Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. ICPC, pages 91–100, June 2011.

[16] M. Kechagia. Improvement of applications' stability through robust APIs. ASE '14, pages 907–910, New York, NY, USA, 2014. ACM.

[17] G. A. Kintab, C. Roy, and G. McCalla. Recommending software experts using code similarity and social heuristics. CASCON 2014, pages 4–18. IBM Corp.

[18] M. Linares-Vásquez. Supporting evolution and maintenance of android apps. ICSE Companion '14, pages 714–717, 2014.

[19] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. API change and fault proneness: A threat to the success of android apps. ESEC/FSE 2013, pages 477–487, New York, NY, USA, 2013. ACM.

[20] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk. How do API changes trigger stack overflow discussions? a study on the android sdk. ICPC '14, pages 83–94, 2014.

[21] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest Q & A site in the west. CHI '11, pages 2857–2866, 2011.

[22] A. K. McCallum. Mallet: A machine learning for language toolkit. 2002.

[23] T. McDonnell, B. Ray, and M. Kim. An empirical study of API stability and adoption in the android ecosystem. ICSM '13, pages 70–79, 2013.

[24] R. Medrano. Welcome to the API economy. http://www.forbes.com/sites/ciocentral/2012/08/29/welcome-to-the-api-economy/, Forbes, accessed in Jan. 2015.

[25] K. K. Nam, M. S. Ackerman, and L. A. Adamic. Questions in, knowledge in?: A study of naver's question answering community. CHI '09, pages 779–788.

[26] S. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming Q & A in stackoverflow. ICSM '12, pages 25–34, Sept. 2012.

[27] T. Nguyen, B. Adams, Z. M. Jiang, A. Hassan, M. Nasser, and P. Flora. Automated verification of load tests using control charts. Software Engineering Conference (APSEC), 2011 18th Asia Pacific, pages 282–289, Dec 2011.

[28] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. ICSE '13, pages 522–531, 2013.

[29] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving low quality stack overflow post detection. (ICSME), 2014, pages 541–544. IEEE, 2014.

[30] S. Ravi, B. Pang, V. Rastogi, and R. Kumar. Great question! question quality in community q&a. Eighth International AAAI Conference on Weblogs and Social Media, 2014.

[31] M. Robillard and R. DeLine. A field study of API learning obstacles. 16(6):703–732, 2011.

[32] P. Royston. Approximating the shapiro-wilk w-test for non-normality. 2(3):117–119, 1992.

[33] C. R. Rupakheti and D. Hou. Evaluating forum discussions to inform the design of an API critic. ICPC, pages 53–62, 2012.

[34] J. Stylos, B. A. Myers, and Z. Yang. Jadeite: Improving API documentation using usage information. CHI '09 Extended Abstracts on Human Factors in Computing Systems, pages 4429–4434, 2009.

[35] S. Subramanian and R. Holmes. Making sense of online code snippets. MSR '13, pages 85–88, Piscataway, NJ, USA, 2013. IEEE Press.

[36] S. Subramanian, L. Inozemtseva, and R. Holmes. Live API documentation. Proceedings of the 36th International Conference on Software Engineering, pages 643–652. ACM, 2014.

[37] S. Thomas. Mining software repositories using topic models. ICSE '2011, pages 1138–1139, May 2011.

[38] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? (NIER Track). ICSE '11, pages 804–807, 2011.

[39] W. Wang and M. W. Godfrey. Detecting API usage obstacles: A study of ios and android developer questions. MSR '13, pages 61–64, Piscataway, NJ, USA, 2013. IEEE Press.

[40] J. Zhang, M. S. Ackerman, and L. Adamic. Expertise networks in online communities: Structure and algorithms. WWW '07, pages 221–230, New York, NY, USA, 2007. ACM.

[41] B. Zhou, X. Xia, D. Lo, C. Tian, and X. Wang. Towards more accurate content categorization of api discussions. Proceedings of the 22nd International Conference on Program Comprehension, pages 95–105. ACM, 2014.

[42] M. Zibran, F. Eishita, and C. Roy. Useful, but usable? Factors affecting the usability of APIs. WCRE '11, pages 151–155, Oct 2011.