Mining API Aspects in API Reviews

Gias Uddin
School of Computer Science
McGill University, Montréal, QC, Canada
gias@cs.mcgill.ca

Foutse Khomh
SWAT lab
Polytechnique Montréal, QC, Canada
foutse.khomh@polymtl.ca

ABSTRACT

With the proliferation of online developer forums as informal documentation, developers share their opinions about the APIs they use. While many developers refer to and rely on opinion-rich information about APIs, we found little research that investigates the use and benefits of public opinions as well as the feasibility of automatically harnessing such valuable knowledge into quick and digestable insights. To understand the potential benefits of API reviews, we conducted a case study of opinions in Stack Overflow. We observed that opinions about diverse API aspects (e.g., usability) are prevalent and offer insights that can shape developer perception and decisions related to software development. Motivated by the finding, we built a suite of techniques to automatically mine opinionated sentences about APIs from forum posts. First, we detected API mentions in the forum posts with up to 95% precision. Second, we associated each opinionated sentence in a forum post to the API mention about which the opinion was provided with 72-95% precision. Third, we build a suite of machine learning classifiers that can detect the presence of an API aspect in a sentence with up to 80% precision.

1. INTRODUCTION

APIs (Application Programming Interfaces) offer interfaces to reusable software components. Modern-day rapid software development is often facilitated by the plethora of open-source APIs available for any given development task. The online development portal GitHub [20] now hosts more than 38 million public repositories, a radical increase from the 2.2 million active repositories hosted in GitHub in 2014. Developers share their projects and APIs via other portals as well (e.g., Ohloh [35], Sourceforge [45], Freecode [18]).

With a myriad of APIs being available, developers now face a new challenge — how to choose the right API. To overcome this challenge, many developers seek help and insights from other developers. Figure 1 presents the screenshot of one Stack Overflow conversation containing one answer and two comments. These posts express developers' opinions about two Java APIs (Jackson [17] and Gson [21]) offering JSON parsing features for Java. None of the posts contain any code snippet. The answer contains discussion about Gson with negative opinions on its usability (hard to use) and positive opinions on the performance (faster) of Jackson. The first comment as a reply to the answer contains another positive opinion about Jackson, but this time it is about the usability of the API (easy to use). Later, the developer 'Daniel Winterstein' develops a new version of Gson fixing existing issues and shares his API (C2). This example illustrates how developers share their experiences and insights about different aspects of an API, as well as how

Figure 1: Example of opinions about two APIs in StackOverflow

they influence and are influenced by the opinions of others. A developer looking for only code examples for Gson would have missed the important insights about the API's limitations, which may have affected her development activities. Indeed, the choice of an API or how to reuse the functionality the API offers, to a considerable degree, can be conditioned upon what other developers think about the API.

However, there is very little research that focuses on the analysis of developers' perception about API opinions and how such opinions affect their decisions related to the APIs. As illustrated in Figure 1, while developer forums serve as communication channels for discussing the implementation of the API features, they also enable the exchange of opinions or sentiments expressed on numerous APIs, their features and aspects. In fact, we observed that more than 66% of Stack Overflow posts that are tagged "Java" and "Json" contain at least one positive or negative sentiment. Most of these (46%) posts also do not contain any code examples.

The sheer volume of such opinions about any given API scattered across many different posts though pose a significant challenge to produce quick and digestible insights. For example, we are aware of no tool that can help a developer who wants to learn about the most reviewed API for JSON parsing in Java in terms of performance or usability. As a first step towards facilitating such analyses, we sought to answer the following three research questions:

RQ1: How do the opinions in the forum post relate to the developers?

To understand the potential value of the opinions and API aspects in the opinions, we conducted a case study of opinions about APIs in Stack Overflow posts. First, we created a benchmark based on labeling from two participants. The benchmark contains 4594 sentences each labeled based on the aspects that are discussed in the sentence. An aspect can be an attribute (e.g., usability) or a contributing factors

towards the usage or development of an API (e.g., licensing or community support). We analyzed the labels and the opinions about APIs in the benchmark and report the findings by answering the following research questions:

• ***Do developers display emotions while discussing about APIs?***

We observed that more than 53% of the posts in the benchmark contained opinions and such opinions are observed consistently across different API domains (e.g., Database, Security, Framework, etc.). While most of the domains contained more positivity than negativity, the posts in some domains (e.g., Framework) contained negativity more than the positivity.

• ***How do the API aspects relate to the opinions provided by the developers?***

We observed that developers expressed varying degrees of emotions towards different API aspects. The most positively reviewed API aspects were 'Security' and 'Legal', while Performance of APIs was among the least positively reviewed.

• ***How do the various stakeholders of an API relate to the provided opinions?***

We observed that both API authors and users leverage forum posts to promote and suggest their API of interest and that such promotions/suggestions were reviewed both favorably and unfavorably across different API aspects.

RQ2: Can we automatically detect API aspects discussed in the opinions?

Motivated by the findings of the case study, we sought to automate the detection of API aspects in forum posts. We present a suite of unsupervised and supervised machine learning classifiers that can detect the presence of an API aspect in a sentence with up to 80% precision.

RQ3: Can we automatically mine opinions about APIs from forum posts?

To leverage opinion-rich information about APIs to support software development activities, we present techniques to automatically detect API mentions in forum posts and trace the mentions to an actual API in our database with almost 95% precision and 70% recall. We detect opinionated sentences. A sentence was considered as opinionated if it had positive or negative sentiments. When such an opinion was provided towards an API, we heuristically associated the opinionated sentence to the API with 72-95% precision.

We make the following contributions: (1) A **case Study** to demonstrate the potential cause and impact of the API aspects in the opinions, (2) A **benchmark dataset** with the labeling of API aspects in 4594 sentences from Stack Overflow posts. The dataset is available in our online appendix [1], (3) Development and evaluation of a suite of **API aspect detection** and **opinion mining** techniques.

Paper Organization. The rest of the paper is organized as follows. We describe the benchmark in Section 2 and then then report the results of our case study in Section 3. We leverage the benchmark to investigate the feasibility of automatic detection of aspects in sentences in Section 4. We explain the overall opinion mining process for APIs in Section 5. We discuss the potential impact of our findings and the threats to validity in Sections 6 and 7. We summarize the related work in Section 8 and conclude in Section 9.

2. A BENCHMARK FOR API ASPECTS

In a previous study, Uddin et al [50] surveyed software developers and found that developers prefer to see opinions about the following API aspects in the forum posts: **(1) Performance**: How well does the API perform? **(2) Usability:** How usable is the API? **(3) Security:** How secure is the API? **(4) Documentation:** How good is the documentation? **(5) Compatibility:** Does the usage depends on other API? **(6) Portability:** Can the API be used in different platforms? **(7) Community:** How is the support around the community? **(8) Legal:** What are licensing requirements? **(9) Bug:** Is the API buggy? **(10) Only Sentiment:** Opinions without specifying any aspect. To investigate the potential values of opinions about an API and its aspects, we needed opinionated sentences from forum posts with information about the API aspects that are are discussed in the opinions. In the absence of any such dataset available, we created a benchmark with sentences from Stack Overflow posts, each manually labeled based on the presence of the above aspects in the sentence.

The benchmark consisted of 4,594 manually labeled sentences from 1,338 Stack Overflow posts (question/answer/-comment) (see Table 1). The threads were selected from 18 tags representing nine distinct domains (two tags for each domain). We selected the tags from domains as diverse as possible, by adopting a purposeful maximal sampling approach (discussed below) [55]. In Table 1, the second column shows the tags for each domain. Overall, the tags came from nine domains. For each domain, we selected the most popular two tags. Popularity was determined based on the total number of threads under each tag. Each tag was also accompanied by another tag 'java'.

Sampling strategy. For each tag, we selected four threads based on a stratified random sampling process as follows: **(1)** Each thread has a score (= #upvotes - #downvotes). We collect the scores, remove the duplicates, then sort the distinct scores. For example, there were 30 distinct scores (min -7, max 141) in the 778 threads for the tag 'cryptography'. **(2)** We divide the distinct scores into four quartiles. In Table 1, we show the total number of threads under each tag for each quartile, e.g., the first quartile had a range $[-7, 1]$ for the above tag. **(3)** For each quartile, we randomly select one thread **(4)** Thus, for each tag, we have four threads. One thread was overlapped between two tags. In Table 1, the 'Tags' column lists the tags under each domain, the posts column shows the total number of posts (question+answer+comment) from the eight threads under each domain. The columns A and C break down the number of posts into answers and comments. The S column shows the total number of sentences under each domain. The domain 'utility' had the maximum number of sentences followed by domains 'widgets' and 'serialize'. The domain 'text' had the minimum number of sentences, followed by the domains 'database' and 'framework'. The last column S/T shows the average number of sentences per thread for each domain.

Thread Preprocessing. We preprocessed the contents in each thread as follows. We formatted a hyperlink by removing the HTML formats and then prepending the hyperlink with a placeholder `URL_`. We kept the hyperlinks because they are often used to refer to an API or its resources. We replaced code examples by placeholders (`CODESNIPPET`) and the code terms by placeholders (`CODETERM`).[1] We did this

[1]A code snippet was detected if there were more than one code line

Table 1: The Benchmark (P = post, A = answer, C = comment)

Domain	Tags	P	A	C	S	S/T
serialize	xml, json	189	68	113	539	67.38
security	auth, crypto	105	34	63	407	50.88
utility	io, file	297	85	204	1008	126
protocol	http, rest	126	49	69	434	54.25
debug	log, debug	138	41	89	479	59.88
database	sql, nosql	128	38	82	370	46.25
widgets	awt, swing	179	67	104	682	85.25
text	nlp, string	83	40	36	273	34.13
framework	spring, eclipse	93	36	49	402	50.25
Total/Average		1338	458	809	4594	64.70

S = sentences, S/T = sentences/thread

Table 2: Progression of agreements between the two coders

Iteration	1	2	3	4	5
Sentence	83	117	52	24	116
Kappa κ	0.28	0.34	0.62	0.71	0.72
% Final	35/59/6*	52/48	50/50	20/80	19/77

to ensure that the labeling was focused only on the textual contents of a sentence. We then detected individual sentences in the thread.[2] We arranged the posts in a thread in a chronological order. For each thread, we created a flat list of sentences by preserving the order of the posts.

Labeling Process

The benchmark was created based on inputs from two coders. The first author coded all of the threads. A post-doc from Ecole polytechnique coded 14 threads. The coding approach is closely adapted from [28]. The approach was as follows:

1. The first two coders jointly labeled one thread by discussing each sentence. The purpose was to develop a shared understanding of the underlying problem and to remove individual bias as much as possible. To assist in the labeling process in the most seamless way possible, we created a web-based survey application. In Figure 2, we show screen shots of the user interface of the application.
2. The two coders separately labeled a number of threads. A cohen kappa value was calculated and the two coders discussed the sentences where disagreements occurred over Skype and numerous emails. The purpose was to find any problems in the labels and to be able to converge to a common ground, if possible.
3. The second coder stopped the labeling once the agreement between the two coders reached the *substantial* level of Cohen Kappa value (0.61 – 0.80) [52] and the change in agreements between subsequent iterations were below 5%, i.e., further improvements might not be possible.
4. The first coder then labeled all of the remaining threads.

We repeated step 2 above five times. In Table 2 we show the number of sentences labeled in each iteration by the two coders and the level of agreement (Cohen Kappa value) for the iteration. The last row in Table 2 shows how the two coders finally agreed on the initially disagreed labels. For example, for iteration 1, the value 35/59/6* is interpreted as follows: 35% of the total disagreed (46) sentences were finally labeled based on the labels of the first coder, 59% from the second coder, and for the remaining 6% the two coders mutually agreed to come up with a different label.

inside the <code> tag or there was at least one assignment operator.

[2]We used the OpenNLP sentence detector [14].

Table 3: Distribution of labels per coder per aspect

	S	B	D	T	U	C	M	P	N	O
Coder1	1	1	15	4	42	3	4	15	6	60
Coder2	9	5	4	3	43	6	10	13	11	45
Final	1	4	4	5	41	4	8	19	11	51

S = Security, B = Bug, D = Document, P = Performance, U = Usability, T = Portability, C = Community M = Compatibility, N = OnlySentiment O = Others

The agreement reached the substantial level at iteration 3 and remained there in subsequent iterations.

Analysis of the Disagreements. In Table 3, we show the number of total labels for per coder each aspect. The two coders labeled each aspect at least once. Other than 'Others' both coders labeled sentences as 'Usability' the most, and 'Portability' as the least.

The major source of disagreement occurred for three aspects: security, documentation, and compatibility. For example, the second coder labeled the following sentence as 'Security' initially: "The J2ME version is not thread safe". However, he agreed with the first coder that it should be labeled as 'Performance', because 'thread safety' is normally associated with the performance-based features of an API. The first coder initially labeled the following sentence as 'Documentation', assuming that the URLs referred to API documentation: "URL[Atmosphere] and URL[DWR] are both open source frameworks that can make Comet easy in Java". The first coder agreed with the second coder that it should be labeled as 'Usability', because it shows how the frameworks can make the usage the API Comet easy in Java.

The manual labels and the discussions were extremely helpful to understand the diverse ways API aspects can be interpreted even when a clear guideline was provided in the coding guide. One clear message was that a sentence can be labeled as an aspect if it contains clear indicators (e.g., specific vocabularies) of the aspects and the focus of the message contain in the sentence is indeed on the aspect. Both of these two constraints are solvable, but clearly can be challenging for any automated machine learning classifier that aim to classify the sentences automatically.

Benchmark overview. Figure 3 shows the overall distribution of the aspects in the benchmark. The 'Others' category is not an aspect, so we exclude that in the chart. Almost half of the labels were 'Usability'. This aspect captured themes both from the usage and design attributes of an API. In our future work, we will investigate of ways to divide this label into the more focused sub-aspects, e.g., design, usage, etc. In total, 95.2% of the sentences were labeled as only one aspect. Among the rest of the sentences, 4.6% were labeled as two aspects and around 0.2% were labeled as three aspects. One major reason was that some of the sentences were not properly formatted and were convoluted with multiple potential sentences. For example, the following sentence was labeled as three aspects (performance, usability, bug): "HTML parsers ... Fast .. Safe .. bug-free ... Relatively simple .....". 37% of the sentences were labeled as others. The investigation of the sentences labeled as 'Others' to determine additional aspects is our future work.

3. OPINION VALUE ANALYSIS (RQ1)

To investigate the potential values of opinions about APIs, we analyzed the opinionated sentences and their relationship with API aspects in our benchmark. We report the findings

Figure 2: Screenshots of the benchmarking app. The circled numbers show the progression of actions in sequential order. The leading page (1) shows the coding guide. Upon clicking the 'Take the Survey', the user is taken to the 'todos' page (2), where the list of threads is provided. By clicking on a given thread in the 'todos' page, the user is taken to the corresponding page of the thread in the app (3), where he can label each sentence in the thread (4). Once the user labels all the sentences of a given thread, he needs to hit a 'Submit' button (not shown) to store his labels. After that, he will be returned to the 'todos' page (i.e., 2).

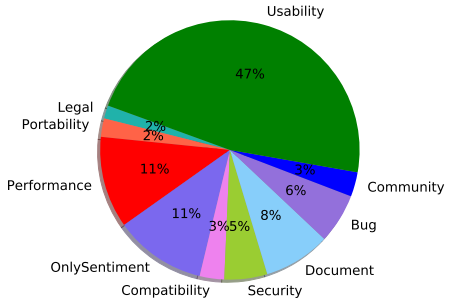

Figure 3: The distribution of aspects in the benchmark.

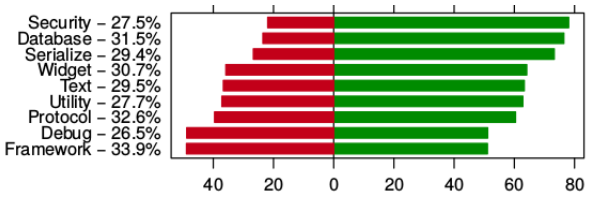

Figure 4: Opinion distribution across domains in benchmark (red and green bars denote negative and positive opinions, rep.

Table 4: Distribution of opinions and code in forum posts

	Code Terms	Code Snippets	Opinions
Percent	5.53	10.99	53.14
Average	0.13	0.18	1.03

by answering three questions:

1. Do developers show emotions while discussing about APIs?
2. How do the API aspects relate to the opinions provided by the developers?
3. How do the various stakeholders of an API relate to the provided opinions?

RQ1.1: Do developers show emotions while discussing about APIs?

Motivation. Previous studies showed the presence of emotions in the issue tracking system [33] and their correlation to the various development activities (e.g., time of fixing a bug, productivity, etc.) [32,36]. We believe that the positive or negative experience of developers while using an API can be related to the opinions they provide in the forum posts while discussing their usage experience of APIs. Within the context of API and the opinions, We sought the answer to two questions: (1) How significant is the presence of opinions in the forum posts? (2) How frequently developers provide opinions within and without the presence of code examples?

Approach. We detected the polarity (positive, negative, neutral) of each sentence in the benchmark. To detect sentiments in the sentences, we used an implementation of the Sentiment Orientation algorithm [25]. The algorithm was used to mine and summarize customer opinions about computer products, such as cameras, cd players, etc., and by Google researchers to detect sentiments in reviews from diverse services (e.g., restaurants) [9]. One of the advantages of this algorithm is its ability to easily use domain-specific sentiment words, which we leveraged by adding selected sentiment words that can be specific to API reviews (e.g., thread-safety). The implementation of the algorithm and the domain specific sentiment words are available in our online appendix. We grouped the sentences per post as well as per the nine domains in our benchmark. We investigated the overall volume and relative proportion of opinions provided per domain. We linked each opinionated sentences to the corresponding forum post where it was provided, and determined the average volume of opinions provided per post.

Findings. In Figure 4, we show the overall presence of opinions per domain. The percentages beside each domain show the overall distribution of opinions (positive and negative) sentences in the domain. The bar for each domain further shows the relative proportion of positivity versus negativity in those opinions. Except two domains (Debug and Framework), developers expressed more positivity than negativity in all domains except two (Framework and Debug). Each domain had at least 26.5% opinionated sentences, i.e., at least one in four sentences contained emotion. In Table 4, we show the distribution of opinions, code terms and code snippets in the forum posts. The first row (Percent) shows the percentage of distinct forum posts that contain at least one code terms or snippets or opinions. The second row (Average) shows the average number of code terms, code snippets and opinionated sentences across the forum posts. More than half of the forum posts (53%) contained at least one opinion, while only 11% contained at least one code example. Therefore, developers while only looking for code examples to learn the usage of APIs may miss important insights shared by other developers in those opinions.

Developers offer significant volume of opinions in the forum posts, much more than they share code examples or discuss code terms. Analysis of such opinions can provide interesting insights into the learning of the APIs discussed.

RQ1.2: How do the API aspects relate to the opinions provided by the developers?

Motivation. Given that the usage of an API can be influenced by factors related to the specific attributes of an API, we believe that by tracing the positive and negative opinions to specific API aspects discussed in the opinions, we can gain deeper insight into the contextual nature of the provided opinions. Tools can be developed to automatically detect those aspects in the opinions to provide actionable insights about the overall usage and improvement of the API.

Approach. We associated the aspects in the benchmark to the opinionated sentences and analyzed the correlation between those based on two distributions: (a) The overall number of positive and negative opinionated sentences associated with each aspect, and (b) For each aspect with opinions, the distribution of those opinions across the different domains. We answer the following questions: (1) What aspects are more dominant in the opinions? (2) What aspects are more disputed across the domains?

Findings. In Table 5, we breakdown the opinions for each aspect into the domains. For each aspect and each domain, we show the overall presence of positive and negative opinions. The percentage beside each aspect name in Table 5 shows how much of the percentage of sentences labeled as that aspect contained opinions with positive or negative sentiment. The bar for each aspect shows the relative proportion of positive and negative opinionated sentences in those opinions. The distribution of opinions across aspects is greater the distribution of opinions across the domains (in Figure 4), i.e., developers are more opinionated while discussing about API aspects. For example, 72% of the sentences labeled as the 'Legal' aspect contained opinions, while 62% of the sentences containing discussion related to API performance contained opinions. Therefore, the analysis of the opinions towards specific aspects can help us understand more about the advantages and problems of using an API based on that aspect (e.g., what are most positively or negatively reviewed performance-based features of an API?).

The two aspects, Performance and Usability, both have much more negative opinions than positive opinions for domains 'Protocol' and 'Debug'. Intuitively, developers can complain about the performance of a program while debugging. However, it is non-trivial any such causes for the domain 'Protocol'. Major theme emerged from the Performance-based negative opinions was related to analysis and deployment of the multi-threaded applications based on the protocols. By cross-linking the opinions both from aspect and domain, we also find that developers did not have any complaints about the Legal aspects of the APIs from the Text domain, although they were not happy with their buggy nature. Such analyses that can help streamline the overall decision making process towards the usage of an API.

Developers are more vocal while discussing specific API aspects (e.g., usability). Correlating the different dimensions (opinions, aspects, and domain) can offer insights into how well reviewed an API aspect is across domains.

RQ1.3: How do the various stakeholders of an API relate to the provided opinions?

Motivation. An API can be associated with different stakeholders. Depending on the usage needs, the interaction between the stakeholders and the analysis of the opinions they provide can offer useful insights into how positively or negatively the suggestions are received among the stakeholders.

Approach. We labeled all the sentences in the benchmark based on three dimensions: (1) **Stakeholder**: Who is providing the opinion? (2) **Signal**: What has caused the stakeholder to provide the opinion? (3) **Intent/Action**: What is the intention and/or action of the stakeholder? We identify the dimensions in the benchmark as follows.

• **Stakeholder:** We manually examined each sentence in the benchmark and labeled it based on the type of person who originally wrote the sentence. For example, we labeled the sentence as came from the author of an API, if the sentence contained a disclosure ("I am the author").

• **Signal:** We identified the following signals in the opinions: (1) **Suggestion** about an API, e.g., "check out gson". We used the following keywords as cues: 'check out', 'recommend', 'suggest', 'look for'. (2) **Promotion:** a form of suggestion but from the author/developer of an API. To qualify as an authors, the opinion-provider had to disclose himself as an author in the same sentence or same post.

• **Intent/Action:** We identified the following intents in the opinions: (1) **Liked**: the opinion showed support towards a provided suggestion or promotion but the opinion provider was different from the suggestion or promotion provider. For cues, we considered all of the positive comments to an answer (when the suggestion/promotion was provided in an answer), or all the positive comments following a comment where the suggestion/promotion was given. (2) **Unliked**: the opinion-provider did agree with the provided suggestion or promotion. For cues, we considered all of the negative comments to an answer (when the suggestion/promotion was provided in an answer), or all the negative comments following a comment where the suggestion/promotion was given. (3) **Used**: the opinion was from a user who explicitly mentioned that he had used the API as suggested. We report the findings using the following metrics:

$$\text{Suggestion Rate} = \frac{\#\text{Suggestions (S)}}{\#\text{Posts (T)}},\ \text{Liked Rate} = \frac{\#\text{Liked}}{\#\text{S} + \#\text{P}}$$

$$\text{Promotion Rate} = \frac{\#\text{Promotion (P)}}{\#\text{Posts (T)}},\ \text{Unliked Rate} = \frac{\#\text{Unliked}}{\#\text{S} + \#\text{P}}$$

Findings. We discuss the findings across the dimensions:
• **Stakeholders:** We observed three types of stakeholders: (1) User (1327): who planned to use the features offered by an already available API or who responded to queries asked by another user in the post. (2) Author (6): who developed or authored an API. (3) User turned author (5): who developed a wrapper around one or more already available APIs and offered the wrapper to others in the post.

• **Signals:** We observed 11 distinct promotions (.8%) from the authors,e.g.,"Terracotta might be a good fit here (disclosure: I am a developer for Terracotta).". We observed 122 distinct suggestions (9.1%),e.g., "Give boon a try. It is wicked fast". Besides

Table 5: Distribution of opinions across aspects and domains (Red and Green bars represent negative and positive opinions, resp.)

↓Aspect|Domain→	Security	Database	Serialize	Widget	Text	Utility	Protocol	Debug	Framework
Performance - 62.1%									
Usability - 41.8%									
Security									
Bug - 49.7%									
Community - 46.2%									
Compatibility - 43%									
Documentation - 37.9%									
Legal - 72%									
Portability - 42.9%									
Others - 9.6%									

promotion, we observed the following types of interactions between users and authors that did not provide any specific promotions and were not included in the signals:(a) Bug fix: from an author, "DarkSquid's method is vulnerable to password attacks and also doesn't work". From a user, "erickson I'm new here, not sure how this works but would you be interested in bounty points to fix up Dougs code?" (b) Support: authors responded to claims that their API was superior to another competing API, e.g., "Performant? ... While GSON has reasonable feature set, I thought performance was sort of weak spot ..."

• **Intents/Actions:** We observed 2887 distinct likes and 1285 unlikes around the provided suggestions and promotions. On average each suggestion or promotion was liked 21.5 times and unliked 9.7 times. Therefore, users were more than two times positive towards the provided suggestions. Two of the users explicitly mentioned that they used the API in the provided suggestions. We believe the usage rate per suggestion can be much more higher, if we considered more subtle notions of usage (e.g., "I will have a try on it"). There are also some other signals in the post that highlight the preference of users towards the selection of an API based on author reputation. Consider the following comment:"Jackson sounds promising. The main reason I mention it is that the author, Tatu Saloranta, has done some really great stuff (including Woodstox, the StAX implementation that I use)."

API authors and users provide opinions to promote and suggest APIs. Users show more positivity than negativity towards an API suggestion in the forum post.

4. API ASPECT DETECTION (RQ2)

While the labeling of aspects in the benchmark offers insights into the various API aspects developers discuss, such a manual approach is not suitable if we want to explore large amount of developer discussions. In this section, we discuss aspect detection classifiers that we developed by leveraging the benchmark described in Section 2. These classifiers take as input a sentence forum post and automatically detect the presence of aspects (e.g., performance, documentation) in the sentence. Due to the absence of any guidelines on how API aspects can be detected in the sentences, we investigated two types of classifiers:

- **Topic-based**: We produced topics representing each aspect and labeled a sentence as an aspect if it contained keywords from its corresponding aspect.
- **Supervised**: We developed one supervised classifier for each aspect based on the bag of word feature model.

Table 6: Performance of the aspect detectors (N = Ngram)

Aspect	Topic-Based			Supervised			
	P	R	F1	N	P	R	F1
Performance	0.27	0.63	0.38	U	0.72	0.49	0.57
Usability	0.52	0.41	0.46	B	0.58	0.58	0.58
Security	0.17	0.73	0.27	U	0.80	0.55	0.59
Bug	0.20	0.57	0.3	U	0.64	0.42	0.45
Community	0.10	0.69	0.18	B	0.19	0.29	0.19
Compatibility	0.08	0.60	0.14	T	0.16	0.23	0.17
Documentation	0.13	0.55	0.22	U	0.53	0.49	0.51
Legal	0.09	0.83	0.17	U	0.72	0.34	0.41
Portability	0.05	0.86	0.10	B	0.68	0.31	0.40
OnlySentiment	0.22	0.42	0.29	B	0.35	0.49	0.41
Others	0.39	0.24	0.3	B	0.48	0.88	0.62

We used three performance measures to assess the classifiers: precision (P), recall(R), F-measure ($F1$) (Equations 1 - 3).

$$P = \frac{TP}{TP + FP} \quad (1) \qquad R = \frac{TP}{TP + FN} \quad (2) \qquad F1 = 2 * \frac{P * R}{P + R} \quad (3)$$

TP = Nb. of true positives, and FN = Nb. false negatives. In Table 6, we show the performance of the two detectors. The supervised classifiers outperformed the topic-based detectors for each aspect. We discuss the algorithms below.

4.1 Supervised Aspect Detection

Because more than one aspect can be discussed in a sentence, we developed a classifier for each aspect. In total, we have developed 11 supervised classifiers (10 for the 10 aspects and one to detect others). To train and test the performance of the classifiers, we used 10-fold cross-validation.

Candidate Supervised Classifiers. Because the detection of the aspects require the analysis of textual contents, we selected two supervised algorithm that have shown better performance for text labeling in both software engineering and other domains: SVM and Logistic Regression. We used the Stochastic Gradient Descent (SGD) discriminative learner approach for the two algorithms, which is better suited for large-scale learning[3]. We trained the algorithms using the bag of words model. An advantage of SGD is that it offers more hyper parameters to tune the perfor-

[3]We used the SGDClassifier of Scikit [43]

mance of the classifier for a given domain. To achieve an optimal performance with a supervised classifer, for a given domain, it is recommended to tune the parameters of the classifier for this domain [7]. Intuitively, the opinions about API performance issues can be very different from the opinions about legal aspects (e.g., licensing) of APIs. Due to the diversity in such representation of the aspects, we hypothesized each as denoting a sub-domain within the general domain of API usage and tuned the hyper parameters for the SGD classifier for each aspect. We computed the hyper parameters using the exhaustive grid search algorithm[4]. As recommended by Chawla [13], to train and test classifiers on imbalanced dataset, we set lower weight to classes with over-representation. In our SGD classifier, we set the class weight for each aspect depending on the relative size of the target values - we used the setting as 'balanced' which automatically adjusts the weights of each class as inversely proportional to class frequencies. Our online appendix contains details of the optimal hyper-parameters.

Picking The Best Classifiers. For each aspect we trained and tested the classifiers on the dataset using 10-fold cross validation as follows: (1) We tokenized and vectorized the dataset into ngrams. We used n = 1,2,3 for ngrams, i.e., unigrams (one word as a feature), bigrams (two consecutive words as a feature) and n = 3 (three consecutive words). (2) We applied the TF-IDF algorithm on the ngrams to normalize the impact of frequent and as well as specific ngrams across the sentences. (3) For each ngram-vectorized dataset, we then did a 10-fold cross-validation of the classifier using the optimal parameter. (4) We took the average of the precision, recall, and F1-score of the 10-folds. (5) Thus for each aspect, we ran our cross-validation three times (one each of the ngrams) (6) We picked the best performing classifier as the one with the best F1-score among the three runs.

While each classifier was tested using three types of ngrams (1,2,3), five aspects in the imbalaned dataset were better suited for unigram-based features (Security, Performance, Bug, Legal, OnlySentiment), five for bigram-based features (Usability, Portability, Community, Documentation, Bug, Others) and only one for trigram-based features (Compatibility). The diversity in ngram selection can be attributed to the underlying composition of words that denote the presence of the corresponding aspect. For example, Performance-based aspects can be recognized through the use of keywords like thread safe, memory footprint, etc. Similarity, Legal aspects can also be recognized through singular words, e.g., free, commercial, etc. In contrast, usability-based features require sequences of words or phrases, e.g., used easily, etc.

4.2 Topic-Based Aspect Detection

For each aspect, we produced the topics using the following steps:(1) We tokenized each sentence labeled as the aspect and removed the stopwords (2) We applied LDA (Latent Dirichlet Allocation) [10] on the sentences labeled as the aspect repeatedly until the coherence value[5] of the LDA model no longer increased. The coherence measure of a given model quantifies how well the topics represent the underlying theme in the data. In Table 7, we show the topic keywords for each aspect. The third column (C) shows the final coherence value. The higher the value is, the more coherent

[4]We used the GridSearchCV algorithm of Scikit-Learn

[5]We used gensim [40] with c_v coherence to produce topics.

Table 7: Examples topic for aspects (C = Coherence, JT, JF = Jaccard index, T for positives, F negatives.)

Aspect	Topic	C	JT	JF
Performance	fast, memory,thread	0.52	0.045	0.007
Usability	easy, works, uses	0.33	0.026	0.011
Security	encrypted,crypto,security	0.54	0.054	0.007
Bug	exception,throw,getters	0.55	0.038	0.005
Community	downvote,community	0.61	0.045	0.005
Compatibility	equivalent, compatible	0.53	0.04	0.008
Documentation	article, tutorial,post	0.53	0.044	0.012
Legal	free, commercial,license	0.5	0.08	0.004
Portability	windows,platform,unix	0.53	0.09	0.007
OnlySentiment	thanks,good,correct	0.61	0.028	0.005
Others	array, write, server	0.49	0.014	0.012

Table 8: Statistics of API database used in the study

API	Java	Python	Javascript	Total
Name	62,549	32,156	56,225	150,930
Hyperlink	74,635	32,156	56,225	163,016

the topics in the model are. For each aspect, we produced two topics and took the top 10 words from each topic as a representation of the aspect. Therefore, for each aspect, we had 20 words describing the aspect. For a given topic, the words at the bottom are typically more generic and less indicative of the topic in general. Even with 10 words, we experienced noises in the topics. For example, for the aspect Bug, one of the words was 'getters' which does not indicate anything related to the buggy nature of an API.

We detected the aspects in a given sentence using the topics as follows: (1) We computed the Jaccard index [31] between the topic-words (T) of a given aspect and the words in the sentence (S) using the following equation:$J = \frac{Common(T,S)}{All(T,S)}$ (2) If the Jaccard index is greater than 0, i.e., there is at least one topic word present in the sentence for the aspect and we label the sentence as the aspect. If not, we do not label the sentence as the aspect. In Table 7, the last three columns show the performance in aspect detection for each aspect. The supervised classifiers for each aspect outperformed the topic-based algorithm for two primary reasons: (a) the topics were based on words (unigrams) and as we observed bigrams and trigrams are necessary to detect six of the aspects. (b) The topics between aspects have one or more overlapping words, which then labeled a sentence erroneously as corresponding to both aspects while it may be representing only one of them. For example, the word 'example' is both in Usability and Documentation and thus can label the following sentence as both aspects: "Look at the example in the documentation", whereas it should only have been about documentation. Determining the labeling based on a custom threshold from the Jaccard index values can be a non-trivial task. For example, in Table 7, the two columns JT and JF show the average Jaccard index for all the sentences labeled as the aspect (T) and not (F) in the benchmark. While for most of the aspects, the values of JF are smaller than the values of JT, it is higher for Others. Moreover, the values of JT are diverse (range $[0.014 - .09]$), i.e., finding a common threshold may not be possible.

5. API OPINION MINING (RQ3)

We mine opinions about APIs using the following steps:

1. **Loading and Preprocessing.** We load Stack Overflow posts and preprocess the contents of the posts as follows: (1) We identify the stopwords. (2) We categorize the post content into four types: a) *code terms*; b) *code snippets*;

c) *hyperlinks*[6]; and d) *natural language text* representing the rest of the content. (3) We tokenize the text and tag each token with its part of speech.[7] (4) We detect individual sentences in the *natural language text*.

2. **Opinionated Sentence Detection.** We detect sentiments (positive and negative) for each the sentence of a given Stack Overflow thread. An opinionated sentence contains at least one positive or negative sentiment.
3. **API Mention Detection and Resolution.** We detect API mentions (name and url) in the textual contents of the forum posts. We link each API mention to an API in our API database. Our API database consists of all of the Java, Python and Javascript APIs listed in two software portals Ohloh [35] and Maven central [44] (see Table 8).
4. **API Mention to Opinion Association.** We associate each opinionated sentence to its corresponding API.
5. **API Aspect Detection.** We then apply the best aspect detection classifier from Section 4 to detect the API aspects in such an opinionated sentence.

In this section, we explain the detection and resolution of API mentions and the association of opinions to the APIs.

5.1 API Mention Detection

We detect potential API mentions in the *textual contents* of the forum posts. First, we identify API names and urls in the posts. Second, we trace each name and url to the API listed in our API database. The detection process is composed of four major steps:(1) Portal operation, (2) Name and url preprocessing, (3) API name detection, and (4) API hyperlink detection. We describe the steps below.

• **Step 1. Portal Operation.** We crawled the the official Javadocs 1.6 SE and EE to get a list of all Java packages and the hyperlink to the package. Each package is considered as an API. We crawled Ohloh to collect all of the Python, Java and Javascript APIs. A Java API is considered if it has the main programming language listed as Java in the portal. We applied similar filters for Python and JavaScript APIs.[8]

• **Step 2. Name and url preprocessing.** We preprocess each API name to collect representative token as follows: (a) **Domain names.** For an API name, we take notes of all the domain names, e.g., for `org.apache.lucene`, we identify that `org` is the internet domain and thus developers just mention it as `apache.lucene` in the post. (b) Provider names: we identify the provider names, e.g., for `apache.lucene` above, we identify that `apache` is the provider and that developers may simply refer to it as `lucene` in the posts. (c) Fuzzy combinations: We create fuzzy combinations for a name multiple tokens. For example for `org.apache.lucene`, we create the following combinations: apache lucene, apache.lucene, lucene apache, and apache-lucene. (d) Stopwords: we consider the following tokens as stopwords and remove those from the name, if any: test, sample, example, demo, and code. (e) Country codes: we remove all two and three digit country codes from the name, e.g., cn, ca, etc. For each such API name, we create two representative tokens for the name, one the full name, and another the name without the code. We preprocess each url as follows: (a) Generic links: We remove hyperlinks that are most likely not pointing to the repository of the API. For example, we removed this hyperlinks http://code.google.com/p, because it just points to the code hosting repository of Google code instead of specifying which project it refers to. (b) Protocol: For an API with hyperlink using the 'HTTP' protocol, we also created another hyperlink for the API using the 'HTTPS' protocol. This is because the API can be mentioned using any of the hyperlink in the post. (c) Base url: For a given hyperlink, we automatically created a base hyperlink by removing the *irrelevant* parts, e.g., we created a base as http://code.google.com/p/gson/ from this hyperlink http://code.google.com/p/gson/downloads/list.

• **Step 3. API name detection.** We match each token in the textual contents of a forum post against each API name in our database. We do two types of matching: exact and fuzzy. If we find a match using exact matching, we do not proceed with the fuzzing matching. We only consider a token in a forum post eligible for matching if its length is more than three characters long. For an API name, we start with its longest token (e.g., between code.gson.com and gson, code.gson.com is the longest token, between 'google gson' and 'gson', 'google gson' is the longest), and see if we can match that. If we can, we do not proceed with the shorter token entities of the API. If for a given mention in the post with more than one exact matches from our database, we randomly pick the one of them. Using contextual information to improve resolution in such cases is our future work. If we don't have any exact match, we do fuzzy match for the token in forum post against all the API names in our database. We do this as follows: (1) we remove all of the non-alpha characters from the forum token (2) we then make it lowercase and do a levenshtein distance. (3) we pick the matches that are above 90% threshold between the token and an API and whose first character match (i.e., both token and API name starts with the same character) (4) If there are more than one match, we pick the one with the highest confidence. If there is a tie, we sort the matches alphabetically and pick the top one.

• **Step 4. API hyperlink detection.** We do not consider a hyperlink in a forum post if the url contains the following keywords: stackoverflow, wikipedia, blog, localhost, pastebin, and blogspot. Intuitively, such urls should not match to any API hyperlink in our database. For other urls, we only consider urls that start with http or https. This also means that if a hyperlink in the post is not properly provided (i.e., broken link), we are unable to process it. For all other hyperlinks in the forum post, we match each of them against the list of hyperlinks in our database. Similar to name matching, we first do an exact match. If no exact match is found, we do a fuzzy match as follows. (a) We match how many of the hyperlinks in our database start with the exact same substring as the hyperlink in the post. We collect all of those. For example, for a hyperlink in forum post http://abc.d.com/p/d, if we have two hyperlinks in our database http://abc.com and http.abc.d.com, we pick both as the fuzzy match. From these matches, we pick the one with the longest length, i.e., http.abc.d.com.

Accuracy analysis. We analyzed the performance of our mention resolution in a benchmark of 18 Java threads. Each thread was selected randomly from each tag we used in aspect creation benchmark (see Table 1). We created the benchmark as follows: 1. For each thread, we tokenized

[6]We detect hyperlinks using regular expressions.

[7]We used the Stanford POS tagger [49].

[8]We crawled Maven in March 2014 and Ohloh in Dec 2013. Ohloh was renamed to Black Duck Open Hub in July 2014.

Table 9: Accuracy analysis of the API mention resolver

	TP	FP	TN	FN	P	R	F1
Name	39	2	472	17	0.95	0.70	0.80
Link	15	0	32	1	1.00	0.94	0.97

the textual contents of the thread and matched each token against the API names in our Java database and the urls in the texts against the Java hyperlinks in our database. This process detected 530 distinct named mentions and 48 distinct linked mentions from all of the threads. 2. The first author manually analyzed each mention and labeled those as 'FALSE' (if it does not refer to an API) or identified the correct API in the database that it referred to. The first author used four sources of information to in the manual validation: (a) Our API database (b) The Stack Overflow thread where the mention was detected, (c) The Google search engine, and (d) The homepages of the candidate APIs for a given mention. 3. If an API was not present in our database, but was mentioned in the forum post, we should considered that to be a missed mention and added that in the benchmark. All such missed mentions were considered as false negatives in the accuracy analysis. 4. A second coder (a graduated PhD student at McGill) was then provided the list, who manually checked each mention. The second coder found three additional mentions missed by the first coder. The second coder disagreed with the resolution of the first coder for 12 mentions. Out of the 12, The final benchmark contained 11 resolutions of the second coder. The other mention 'tomcat' which was denoting a server and not an API.

We ran our API name and link resolver on all of the threads in the benchmark and compared the results of the tool against the benchmark. In Table 9, we present the performance of the mention resolution. The name mention resolver failed to resolve the mention ch.qos.logback.core.recovery to the correct API logback, because our database did not have the class names of the API and exact match had very low confidence between the API name and the mention. Similarity, the mention 'lang' in thread 837703 was erroneously considered as 'FALSE' because the word 'lang' is a common English word and thus was considered as noise. While the performance of the resolver is promising, we note that the tool currently does not take into account the context surrounding a mention and thus may not work well for false mentions, such as 'Jackson' as an API name versus as a person name, etc. We have been investigating the feasibility of incorporating contextual information into the resolution process as our immediate future work.

5.2 API Mention to Opinion Association

We associate an opinionated sentence in a post to an API about which the opinion is provided using three filters: **(1)** API mentioned in the **same sentence. (2)** API in **same post. (3)** API in **same conversation.** We apply the filters in the same order they are discussed.

• **F1. Same sentence association.** If the API was mentioned in the same sentence, we associate the opinion to the API. If more than one API is mentioned in the same sentence, we associate the *closest* opinion to an API as follows. Consider the example, 'I like the object conversation in Gson, but I like the performance of Jackson'. In this sentence, we associate Gson to the opinion 'I like the object conversion in Gson' and Jackson to 'but I like the performance of Jackson'. We do this by splitting the sentence into parts based on the presence of sentiment words (e.g., like) and then associating it to its nearest API that was mentioned after the sentiment word.

Table 10: Accuracy analysis of the API to opinion association

Filter	TP	FP	TN	FN	P	R	F1
Sentence	19	1	0	0	0.95	1	0.98
Post	16	2	0	1	0.89	0.95	0.91
Conversation	8	3	0	1	0.72	0.89	0.8

• **F2. Same post association.** If an API was not mentioned in the same opinionated sentence, we attempted to associate the sentence to an API mentioned in the same post. First, we look for an API mention in the following sentence of a given opinionated sentence. If one mention is found there, we associate the opinion to the API. If not, we associate it to an API mentioned in the previous post (if any). If not, we further attempt to associate it to an API mentioned in the following two sentences. If none of the above attempts are successful, we apply the next filter.

• **F3. Same conversation association.** We define a conversation as an answer/question post and the collection of all comments in reply to the answer/question post. First, we arrange all the posts in a conversation based on the time they were provided. Thus, the answer/question is the oldest in a conversation. If an opinionated sentence is found in a comment post, we associate it to its nearest API mention in the same conversation as follows: an API mentioned in the immediately preceding comment or the nearest API mentioned in the answer/question.

Accuracy. In Table 10, we show the performance of the association filters on the 18 threads that we used to analyze the performance of the mention resolver. The SameSentence filter performed the best, followed by the SamePost filter. The SameConversation filter suffered from the ambiguity in the conversation when the users did not refer to the immediately preceding API mention in their opinion.

6. DISCUSSIONS

Motivated by our findings in the case study on the impact of opinions towards software developers, we developed a prototype opinion search engine where developers can search for an API by its name to explore the positive and negative opinions provided for the API by the developers in the forum posts (see Figure 5 for screenshots). We are currently designing experiments to evaluate its usefulness for software developers. The tool has two components:

• **Offline processor** that crawls Stack Overflow posts to mine opinions using the techniques discussed in Section 5.

• **Online search engine** where users can search for an API for opinion (① in Figure 5). Upon clicking on a search result, the tool will provide the following info based on the mined opinion about the API:

– A **sentiment aggregator** that provides a visual overview of the overall sentiments towards the API (② in Figure 5) and an overview of sentiment trend changes (month by month) towards the API (③ in Figure 5).

– An **opinion viewer** to show each of the positive and negative opinions ranked based on time. The most recent opinion is placed at the top.

– An **opinion summarizer** that groups opinions by aspects (④ in Figure 5). Cicking on each aspect shows each of

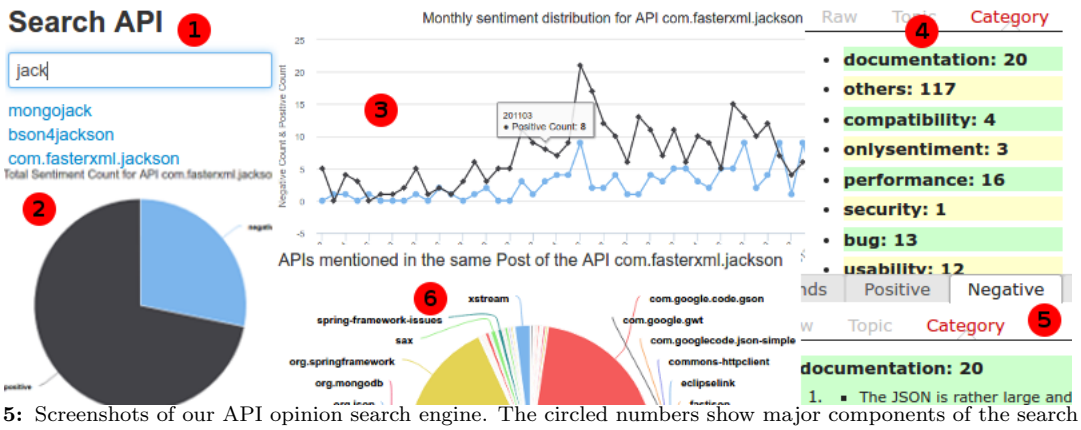

Figure 5: Screenshots of our API opinion search engine. The circled numbers show major components of the search engine.

the opinionated sentences labeled as the aspect for the API.

– A **comentioned viewer** that shows for each API mention which other APIs were positively or negative reviewed in the same forum post (see ⑥ in Figure 5). This analysis can reveal other similar APIs to developers if they are not satisfied with their initially selected API.

7. THREATS TO VALIDITY

The accuracy of the evaluation of API aspects and mentions is subject to our ability to correctly detect and label each such entities in the forum posts we investigated. The inherent ambiguity in such entities introduces a threat to investigator bias. To mitigate the bias, we conducted reliability analysis on both the evaluation corpus and during the creation of the benchmark. While we observed a high level of agreement in both cases, we, nevertheless, share the complete evaluation corpus and the annotations, as well as the results in our online appendix [1].

Due to the diversity of the domains where APIs can be used and developed, the generalizability of the approach requires careful assessment. While the current implementation of the approach was only evaluated for Java APIs and Stack Overflow posts discussing Java APIs, the benchmark was purposely sampled to include different Java APIs from domains as diverse as possible (18 different tags from 9 domains). The domains themselves can be generally understood for APIs from any given programming languages. In particular, the detection of API aspects is an ambitious goal. While our assessment of the techniques show promising signs in this new research direction, the results will not carry the automatic implication that the same results can be expected in general. Transposing the results to other domains requires an in-depth analysis of the diverse nature of ambiguities each domain can present, namely reasoning about the similarities and contrasts between the ambiguities.

8. RELATED WORK

Related work can be categorized into the analysis of (1) Developer forums; (2) Sentiment in software engineering; and (3) APIs in informal documentation.

Analysis of Developer Forums. Online developer forums have been studied to find dominant discussion topics [5,42], to analyze the quality of posts and their roles in the Q&A process [4,11,15,27,29,51], to analyze developer profiles (e.g., personality traits of the most and low reputed users) [6,19], or to determine the influence of badges in Stack Overflow [2]. Several tools have been developed utilizing the knowledge from the forums, such as autocomment assistance [54], collaborative problem solving [12,48], and tag prediction [46].

Sentiment Analysis in Software Artifacts. Ortu et al. [36] observed that bullies are not more productive than others in a software development team. Mika et al. [32] correlated VAD (Valence, Arousal, Dominance) scores [53] in Jira issues with the loss of productivity and burn-out in software engineering teams. Pletea et al. [39] observed that security-related discussions in GitHub contained more negative comments. Guzman et al. [23] found that GitHub projects written in Java have more negative comments as well as the comments posted on Monday, while the developers in a distributed team are more positive. Guzman and Bruegge [24] summarized emotions expressed across collaboration artifacts in a software team (bug reports, etc.) using LDA [10] and sentiment analysis. Murgia et al. [33] labelled comments from Jira issues using Parrot's framework [38]. Jongeling et al. [26] compared four sentiment tools on comments posted in Jira while Novielli et al. [34] analyzed the sentiment scores from the StentiStrength in Stack Overflow posts. They observed that developers express emotions towards the technology, not towards other developers. They found that the tool was unable to detect domain-dependent sentiment words (e.g., bug).

APIs in Informal Documentation. Parnin et al. [37] have investigated API classes discussed in Stack Overflow using heuristics based on exact matching of classes names with words in posts (title, body, snippets, etc.). Using a similar approach, Kavaler et al. [27] analyzed the relationship between API usage and their related Stack Overflow discussions. Both studies found a positive relationship between API class usage and the volume of Stack Overflow discussions. More recent research [22,30] investigated the relationship between API changes and developer discussions. Our findings contribute to the existing body of knowledge on the topic and suggest future extension of the existing research on API traceability, such as, tracing API code terms in documentation [8,16,41,47], and in emails [3].

9. SUMMARY

With the proliferation of online developer forums as informal documentation, developers share their opinions about the APIs they use. With the myriad of forum contents containing opinions about thousands of APIs, it can be challenging for a developer to gain informed decision on what API can be the right choice for his development task. We conducted a case study of the values of opinions about APIs

and found that opinions about APIs are diverse and contain topics from many different aspects. We further observed that developer forums are important avenues for both API users and authors to share and promote APIs. As a first step towards providing actionable insights from opinions about APIs, we presented suites of techniques to automatically mine and summarize opinions by aspects. Our future work involves the effectiveness analysis of our opinion search engine and to identify improvement opportunities to the underlying mining techniques across programming languages.

10. REFERENCES

[1] *Opinion value analysis in API reviews.* https://github.com/anonymous-fse/opinionvalue2, 27 Feb 2017 (last accessed).

[2] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec. Steering user behavior with badges. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 95–106, 2013.

[3] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *32nd International Conference on Software Engineering*, pages 375–384, 2010.

[4] K. Bajaj, K. Pattabiraman, and A. Mesbah. Mining questions asked by web developers. In *In Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 112–121, 2014.

[5] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, pages 1–31, 2012.

[6] B. Bazelli, A. Hindle, and E. Stroulia. On the personality traits of stackoverflow users. In *In Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM)*, pages 460–463, 2013.

[7] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 2546–2554, 2011.

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

[9] S. Blair-Goldensohn, K. Hannan, R. McDonald, T. Neylon, G. A. Reis, and J. Reyner. Building a sentiment summarizer for local search reviews. In *WWW Workshop on NLP in the Information Explosion Era*, page 10, 2008.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

[11] F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli. Mining successful answers in stack overflow. In *In Proceedings of the 12th Working Conference on Mining Software Repositories*, page 4, 2014.

[12] S. Chang and A. Pal. Routing questions for collaborative answering in community question answering. In *In Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining ACM*, pages 494–501, 2013.

[13] N. V. Chawla. *Data Mining for Imbalanced Datasets: An Overview*, pages 875–886. Springer US, Boston, MA, 2010.

[14] CoreNLP. *The Stanford Natural Language Processing Group.* http://nlp.stanford.edu/software/corenlp.shtml, 1999.

[15] D. Correa and A. Sureka. Chaff from the wheat: Characterization and modeling of deleted questions on stack overflow. In *In Proceedings of the 23rd international conference on World wide web*, pages 631–642, 2014.

[16] B. Dagenais and M. P. Robillard. Recovering traceability links between an API and its learning resources. In *Proc. 34th IEEE/ACM Intl. Conf. on Software Engineering*, pages 45–57, 2012.

[17] FasterXML. *Jackson.* https://github.com/FasterXML/jackson, 2016.

[18] freecode.com. http://freecode.com/, 2013.

[19] A. L. Ginsca and A. Popescu. User profiling for answer quality assessment in q&a communities. In *In Proceedings of the 2013 workshop on Data-driven user behavioral modelling and mining from social media*, pages 25–28, 2013.

[20] github.com. https://github.com/, 2013.

[21] Google. *Gson.* https://github.com/google/gson, 2016.

[22] L. Guerrouj, S. Azad, and P. C. Rigby. The influence of app churn on app success and stackoverflow discussions. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 321–330.

[23] E. Guzman, D. Azócar, and Y. Li. Sentiment analysis of commit comments in github: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 352–355, 2014.

[24] E. Guzman and B. Bruegge. Towards emotional awareness in software development teams. In *Proceedings of the 7th Joint Meeting on Foundations of Software Engineering*, pages 671–674, 2013.

[25] M. Hu and B. Liu. Mining and summarizing customer reviews. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177, 2004.

[26] R. Jongeling, S. Datta, and A. Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *Proceedings of the 31st International Confernece on Software Maintenance and Evolution*, 2015.

[27] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov. Using and asking: Apis used in the android market and asked about in stackoverflow. In *In Proceedings of the INTERNATIONAL CONFERENCE ON SOCIAL INFORMATICS*, pages 405–418, 2013.

[28] O. Kononenko, O. Baysal, and M. W. Godfrey. Code review quality: How developers see it. In *Proc. 38th International Conference on Software Engineering*, pages 1028–1038, 2016.

[29] S. Lal, D. Correa, and A. Sureka. Miqs: Characterization and prediction of migrated questions on stackexchange. In *In Proceedings of the 21st Asia-Pacific Software Engineering Conference*, page 9, 2014.

[30] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 83–94, New York, NY, USA, 2014. ACM.

[31] C. D. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval.* Cambridge Uni Press, 2009.

[32] M. Mántylà, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu. Mining valence, arousal, and dominance – possibilities for detecting burnout and productivity? In *Proceedings of the 13th Working Conference on Mining Software Repositories*, pages 247–258, 2016.

[33] A. Murgia, P. Tourani, B. Adams, and M. Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

[34] N. Novielli, F. Calefato, and F. Lanubile. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering*, pages 33–40, 2015.

[35] Ohloh.net. www.ohloh.net, 2013.

[36] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. Are bullies more productive?

empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015.

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

[38] W. G. Parrott. *Emotions in Social Psychology*. Psychology Press, 2001.

[39] D. Pletea, B. Vasilescu, and A. Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 348–351, 2014.

[40] R. Řehůřek and P. Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, 2010.

[41] P. C. Rigby and M. P. Robillard. Dicovering essential code elements in informal documentation. In *Proc. 35th IEEE/ACM International Conference on Software Engineering*, pages 832–841, 2013.

[42] C. Rosen and E. Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, page 33, 2015.

[43] scikit learn. *Machine Learning in Python*. http://scikit-learn.org/stable/index.html#, 2017.

[44] Sonatype. *The Maven Central Repository*. http://central.sonatype.org/, 22 Sep 2014 (last accessed).

[45] Sourceforge.net. http://sourceforge.net/, 2013.

[46] C. Stanley and M. D. Byrne. Predicting tags for stackoverflow posts. In *In Proceedings of the 12th International Conference on Cognitive Modelling*, pages 414–419, 2013.

[47] S. Subramanian, L. Inozemtseva, and R. Holmes. Live api documentation. In *Proc. 36th International Conference on Software Engineering*, page 10, 2014.

[48] Y. Tausczik, A. Kittur, and R. Kraut. Collaborative problem solving: A study of math overflow. In *In Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 355–367, 2014.

[49] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, pages 173–180, 2013.

[50] G. Uddin, O. Baysal, and L. Guerrouj. Understanding how and why developers seek and analyze api-related opinions. *IEEE Transactions on Software Engineering*, page 13, 2017.

[51] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov. 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*, pages 342–354, 2014.

[52] A. J. Viera and J. M. Garrett. Understanding interobserver agreement: The kappa statistic. *Family medicine*, 37(4):360–363, 2005.

[53] A. B. Warriner, V. Kuperman, and M. Brysbaert. Norms of valence, arousal, and dominance for 13,915 english lemmas. *Behavior Research Methods*, 45(4):1191–1207, 2013.

[54] E. Wong, J. Yang, and L. Tan. Autocomment: Mining question and answer sites for automatic comment generation. In *In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 562–567, 2013.

[55] R. K. Yin. *Case study Research: Design and Methods*. Sage, 4th edition, 2009.