Opiner: An Opinion Search and Summarization Engine for APIs

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*—Opinions are key determinants to many of the activities related to software development. The perceptions of developers about an API, and the choices they make about *whether* and *how* they should use it, may, to a considerable degree, be conditioned upon how other developers *see* and *evaluate* the API. Given the plethora of APIs available for a given development task and the advent of developer forums as the media to share opinions about those APIs, it can be challenging for a developer to make informed decisions about an API to support the task. We introduce Opiner, our opinion search and summarization engine for API reviews. The server side of Opiner collects and summarizes opinions about APIs by crawling online developer forums and by associating the opinions found in the forum posts to the APIs discussed in the posts. The client side of Opiner is a Website that presents different summarized viewpoints of the opinions about the APIs in an online search engine. We evaluated Opiner by asking Industrial developers to select APIs for two development tasks. We found that developers were interested to use our proposed summaries of API reviews and that while combined with Stack Overflow, Opiner helped developers to make the right decision with more accuracy and confidence. The Opiner online search engine is available at: http://opiner.polymtl.ca. A video demo is available at: https://youtu.be/XAXpfmg5Lqs.**

***Index Terms*—Opinion mining, API informal documentation, opinion summaries, study, summary quality.**

I. Introduction

APIs (Application Programming Interfaces) offer interfaces to reusable software components and are an integral part of the modern day rapid software development. The online development portal GitHub now hosts more than 38 million public repositories, which is a radical increase from the 2.2 million active repositories hosted in GitHub in 2014. The plethora of APIs available for virtually any given development task offers tremendous benefit to the developers to support their development task, but such richness also offers an interesting problem to the developers - how to select the right API?

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 (i.e., 46%) posts also do not contain any code examples. The number of numerous APIs available and the sheer volume of opinions about any given API scattered across many different posts though pose a significant challenge to produce quick and digestible insights.

Fig. 1. Example of a Stack Overflow discussion about two APIs.

Figure 1 presents the screenshot of seven Stack Overflow posts (four answers, three comments). The oldest post (at the top) is dated November 06, 2009 and the most recent one (at the bottom) is February 10, 2016. These posts express developers' opinions about two Java APIs (Jackson [1] and Gson [2]) offering JSON parsing features for Java. None of the posts contain any code snippets. The first answer (A1) representing a positive opinion about the Gson API motivates the developer 'binaryrespawn' to use it (C1). In the next answer (A2), the user 'StaxMan' compares Gson with Jackson, favouring Jackson for offering better support, and

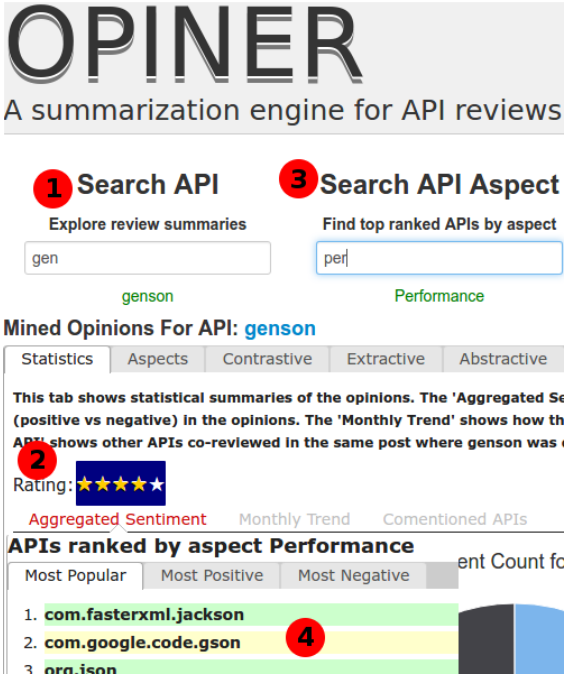

Fig. 2. Screenshots of Opiner API review search engine

based on this feedback, 'mickthomson' (A3) decides to use Jackson instead of Gson. Most of the posts (A2–A4) imply a positive sentiment towards Jackson but a negative one about Gson. Later, the developer 'Daniel Winterstein' develops a new version of Gson fixing existing issues and shares his API (C3). This example illustrates how developers share their experiences and insights, as well as how they influence and are influenced by the opinions. 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. Thus, opinions extracted from the informal discussions can drive developers' decision making.

We developed Opiner, an online API review search and summarization engine. Given as input all the opinionated sentences about an API, Opiner produces summaries of the reviews. In Figure 2, we present a screenshot of Opiner. Opiner is developed as a search engine, where developers can search opinions for an API ①. Upon clicking on API, developers can see all the reviews collected about the API both in summarized and original formats ②. A developer can also search for an API aspect (e.g., performance) ③ in Opiner, to find the most popular APIs based on the aspect ④.

We conducted a study where we provided access to our tool to professional software engineers to help them in their selection of an API for two development tasks. The developers attempted to select an API first by using Stack Overflow only and then by using both Opiner and Stack Overflow. We observed that developers correctly picked the right API with 20-66% accuracy while just using Stack Overflow. However, they had 100% accuracy while they used Opiner and Stack Overflow together. A video demo of Opiner is available at: https://youtu.be/XAXpfmg5Lqs. The Opiner online search engine is available at: http://opiner.polymtl.ca

II. Challenges in API Review Analysis

We addressed three major technical challenges during the mining and summarization of opinions about APIs from developer forums:

- **API Mention Detection:** The collection of opinions about APIs requires the detection of API mentions in the forum posts. The detection is challenging due to two major ambiguities in how APIs are mentioned in the textual contents of the forum posts: 1) An API name can be false. For example, Jackson as an API name vs a person name. 2) An API mention can be associated with more than one API name. For example, in the online Java API hosting portal, Maven Central, a search for Jackson returns 258 APIs containing the name 'jackson'.
- **API Opinion Association:** The association of opinions to an API is challenging when more than one API were mentioned around an opinion.
- **API Opinion Summarization:** Opinions are summarized in other domains mainly around aspects. The summarization of opinions is challenging due to the implicit nature of aspects discussed in the opinions. For example, the opinion 'GSON is fast' is about the 'Performance' aspect of the API GSON.

We briefly discuss the techniques we developed to addressed the above challenges and provide reference to the technical report explaining the detailed techniques and their evaluation.

A. API Mention Detection

An API mention in a post is a reference to an API. A mention can be one of the following types:

- *Name:* A name as a token (e.g., "Jackson") or a series of tokens (e.g., "Jackson JSON parser").
- *Link:* A link to an API resource (e.g., homepage).
- *Code:* Code (or code like) term/snippet using packages and code elements from the API.

In Opiner, we focused on the resolution of mentions referenced by names and links. There are two mention detectors: (1) named-entity (for Natural language texts), and (2) linked-entity detector (for hyperlinks). For each detected mention, the output is a Mention-Candidate List (MCL). In an MCL, there is one mention, but there can be more than one candidate. Each candidate is an API in our API database. Our API database consisted of all the Java APIs listed in online software portals, Maven Central and Ohloh. In Figure 3, we show a partial mention candidate list for the mention 'Jackson' shown in Figure 1. In Figure 3, each ellipsis contains a distinct candidate name. The top part of an ellipsis contains the API name, while the bottom part contains the module name (if the module name is matched). For each mention in a forum post, the resolver takes as input the MCL of the mention and associates the mention to an API in the candidate list if mention is true API or label is 'FALSE', if the mention was not referring to an API. The resolution engine correlates the information found about an API in the database against those

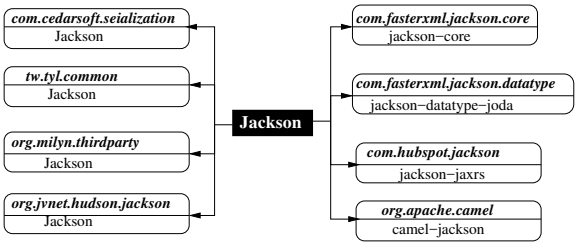

Fig. 3. A partial mention candidate list (MCL) for "Jackson".

found in the textual and code examples found in the forum post where the API mention was found. The correlation is analyzed using heuristics, such as, similarity in the API name with the mention, similarity in the discussion of API features in the forum post against the description of the API in our API database. A detailed explanation of the technique is described in [3] and [4]. We observed a precision of more than 90% for [3] and are currently evaluating [4]. The Opiner current website is deployed using [3].

B. Opinion Mining

We collected the opinionated sentences about APIs using our technique described in [3]. The technique takes cues from discourse resolution by attempting to associate an opinion to its *nearest* API mention. A *nearest* API is determined based on the presence of APIs in the same sentence, post or conversation. A conversation was defined as having all the replies to a given post. A reply to post is an immediate comment to an answer or a response to a comment in a Stack Overflow thread. We observed a precision of more than 90% for the association based on the same sentence heuristics and above 80% for the other heuristics.

C. Opinion Summarization

In a previous study [3], we 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. (11) Others: Opinions about other aspects. We developed a supervised classifier to detect each aspect. To train and test the classifiers, we produced a benchmark of more than 4000 sentences as Stack Overflow, each labeled as one or more aspect. For the opinions labeled as 'Others', we further categorized those using techniques adapted Hu and Liu [5]. The discussion and evaluation of the opinion summarization techniques are described in our technical papers [3], [6].

III. System Architecture

The Opiner architecture supports the efficient implementation of the algorithms described in Section II, and a web-based

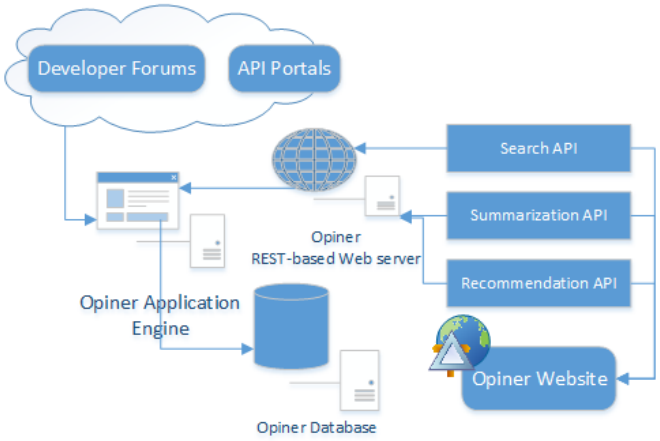

Fig. 4. The client-server architecture of Opiner

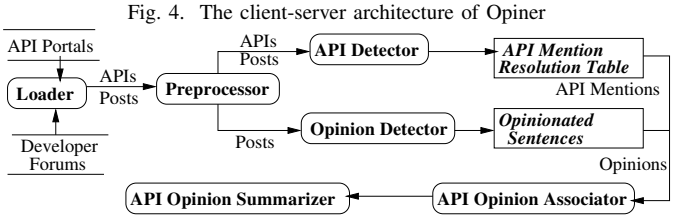

Fig. 5. The architecture of the Opiner Application Engine

search engine with visual representation of the insights obtained from the collection and summarization of API reviews. There are four major components in Opiner (see Figure 4):

- **Application Engine:** Handles the loading and processing of the data from forum posts and API portals and hosts all the algorithms and techniques used to collect and summarize opinions about APIs (Section III-A).
- **Database:** A hybrid data storage component with support for data manipulation in diverse formats, such as, relational, graph, tree, etc. (Section III-B).
- **REST Web Server:** The middleware between the Application Engine and the Website (Section III-C).
- **Website:** The web-based user interface of Opiner that hosts the search, summarization and recommendation results based on API reviews (Section III-D).

A. Opiner Application Engine

The application engine has six major modules (Figure 5):

1) **Loader:** crawls the developer forums and API portals.
2) **Preprocessor:** processes the data into different formats to be consumed for subsequent analyses.
3) **API Mention Resolver:** detects API mentions (name and url) in the textual contents of forum posts.
4) **Opinion Detector:** detects positive and negative opinionated sentences in the forum posts.
5) **Opinion to API Associator:** associates opinions to the API about which opinion was provided in the forum post.
6) **API Opinion Summarizer:** summarizes opinions about an API in different formats.

• **Loader:** There are two types of crawlers: Forum and Portal. For each forum and portal to crawl, the component contains a parser. We have two separate databases: one for the database (and the portals) and another for the forums. For example, we have one parser for Reddit and another for StackOverflow. In this paper, we used an offline version of the StackOverflow dump (January 2014) to ensure reproducibility of our technique and evaluation. It can happen that a post we may have chosen in our experiment can be deleted in the online forum, but we can still have it the offline dump. For example, the StackOverflow thread 338586 was deleted during the middle of 2014, but it was included in the January dump. We have one crawler for the API portal, Maven central and another for Ohloh. In addition, we have a javadoc crawler to collect information about the official Java packages.

• **Preprocessor:** We preprocess the forum contents and the API descriptions in our API database as follows: 1) We identify the stopwords in the forum contents and in the description of APIs in the database. We used three types of stopwords: (1) *Regular*: The default stopwords in English (e.g., a, the, etc.) (2) *Country*: The country and institutional codes (e.g., com, edu, ca, eu, etc.), and (3) *Domain*: API-specific (e.g., API, library, etc.) 2) We categorize the post content into four types: a) *code terms*; b) *code snippets*; c) *hyperlinks*[1]; 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.[2]

• **API Mention Resolver:** The API mention resolver component consists of three major modules (Figure 6): API database creator, Mention Detector and Mention Resolver.

The API database creator has two modules: spurious API detector, API schema generator. **(a) *Spurious API detector:*** We discarded three types of APIs: (i) Demo or example software (e.g., projects with name 'demo' or 'example' in their names), and (ii) APIs that do not have links to their online source code repository, and (iii) APIs that do not have their source code archived (e.g., in a jar file). **(b) *Schema generator:*** We store each API entry by collecting all the information as required by the schema template of the API database. Ideally, all such information should be available from a portal. However, information can be missing. For example, many APIs in Maven do not have any overview description. We crawl the resource pages (e.g., homepage) of those APIs to store their overview description.

The API mention detector in Opiner supports both exact and fuzzy matching of API names and urls from the API database in the textual contents of the forum posts. Fuzzy matching was necessary, because of 1) possible typos in the mention in forum posts, 2) usage of shortened or longer names in forum posts with regards to an API To support the name matching, we produced a *trie* representation of our entire API database, which was both time and space efficient.

[1]We detect hyperlinks using regular expressions.

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

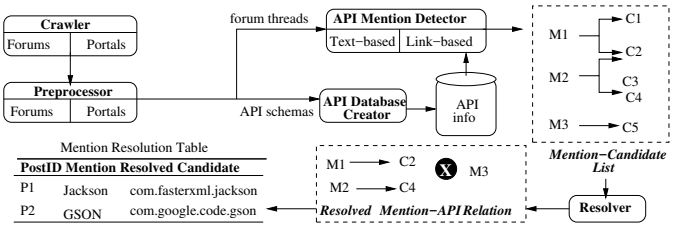

Fig. 6. The architecture of the API mention resolver

The Mention Candidate Lists used in our API mention resolution were developed as an in-memory graph structure to explore the relationships among the candidates, e.g., their dependencies on each other.

• **Opinion Mining.** Given as input a thread from Stack Overflow, we mined opinions about APIs mentioned in the thread as follows: 1) We detected individual sentences in the thread 2) We arranged the sentences in the same sequence they were provided in the thread. For example, the oldest post was placed at the top, and its first sentence as the topmost. 3) We labeled each sentence as either positive, negative, neutral. To detect sentiments in a sentence, we used a rule-based algorithm based on a combination of Sentistrength [8] and the Sentiment Orientation (SO) algorithm [9]. 4) We associated each opinionated sentence to the API about which the opinion was provided using the technique described in Section II.

• **Opinion Summarization.** Given as input all the opinionated sentences of an API, we investigated opinion summarization algorithms from the following major categories [10], [11]:

1) **Aspect-based**: positive and negative opinions are grouped around aspects related to the entity (e.g., picture quality). Aspects can be pre-defined or dynamically inferred using algorithms such as topic modeling.
2) **Contrastive**: Contrastive viewpoints are grouped.
3) **Extractive**: A subset of the opinions are extracted.
4) **Abstractive**: An abstraction of the opinions is produced.
5) **Statistical**: Overall polarity is transformed into numerical rating (e.g., star ratings).

We produced aspect-based and statistical summaries of API reviews using our techniques as described in Section II and [6]. We leveraged off-the-shelf algorithms to produce extractive, abstractive, and topic-based summaries and implemented the algorithm [12] to produce contrastive summaries.

B. Opiner Database

We used a hybrid data storage architecture. The bulk of the data is stored in a Postgresql relational database. The nature of the relational data structure is not efficient for the analysis of the textual data. Such unstructured and intermediate analyses are carried on using disk-based IO. The supervised classifiers are stored in binary serialized format.

C. Opiner REST Web Server

The Opiner REST web server offers interfaces to ingest the results of the Opiner Application Engine. The Opiner website leverages the REST web server to produce the user interface

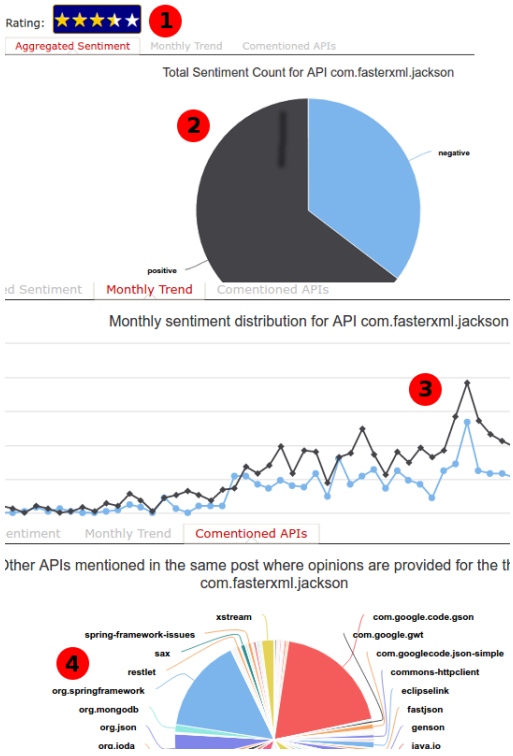

Fig. 7. Statistical summarization of opinions for API `Jackson`.

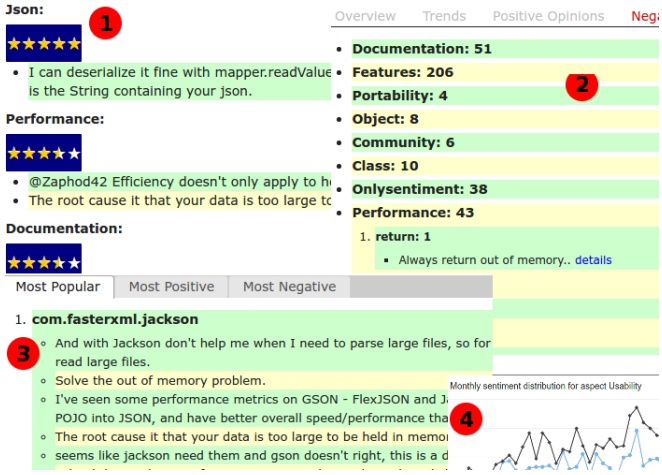

Fig. 8. Screenshot of the aspect-based summarizer

.

of Opiner. This decoupling allows the development of any user interface without preventing the development of the Application Engine. The end points are divided into three categories: 1) Search 2) Summarizations, and 3) Recommendations.

D. Opiner Website

The Opiner website is built based on AJAX principles. Thus, the search capabilities are auto-complete. In Figures 7 and 8, we show the statistical and aspect-based summaries produced by Opiner for the API 'Jackson' based on the processing of all the Stack Overflow posts tagged as 'java+json', respectively.

TABLE I
PROGRESSION OF LEARNING FROM STACK OVERFLOW TO OPINER

T	Tool	Correctness	Conversion	Confidence	Time
1	SO	20.0%	–	4.3	12.3
	SO + Opiner	100%	100%	4.5	7.5
2	SO	66.7%	–	2.7	18.6
	SO + Opiner	100%	100%	4.6	6.8

IV. Evaluation

Because the purpose of developing Opiner was to add benefits over Stack Overflow, we investigated the usefulness of Opiner by conducting a study of professional software developers who completed two tasks using Stack Overflow and Opiner. We detected API mentions using the technique [3] in Opiner for this evaluation.

The two tasks in total involved four different APIs, with two APIs for each task. Each task was described using a development scenario. For each task there was only one correct API (decided based on cues from discussions in Stack Overflow). The participants were asked to select the correct API for each task. Each task was executed in two settings:

1) **SO only**: Decide only using Stack Overflow
2) **SO + Opiner**: Decide using Stack Overflow + Opiner

For a task in a setting, each developer provided the following answers upon task completion: (1) **Selection**: Their choice of API (2) **Confidence**: How confident they were while making the selection. We used a five-point scale: Fully confident (value 5) - Fully unsure (1). (3) **Rationale**: The reason of their selection. In addition, we logged the time it took for each developer to complete a task under each setting.

We invited nine professional developers from a software company and two developers from Freelancer.com to participate in the study. The experience of the developers ranged between 1 year and 34 years. The developers carried on different roles ranging from software developer to architect to team lead. We analyzed the responses along three dimensions:(1) **Correctness**: How precise the participants were while making a selection in the two settings. (2) **Confidence**: How confident they were making the selection? (3) **Time**: How much time did the developers spend per task? In addition, we computed a 'conversion rate' as the ratio of developers who made a wrong selection using Stack Overflow but made the right selection while using both Stack Overflow and Opiner.

Nine developers completed task 1 using the two different settings (10 using Stack Overflow). Five developers completed task 2 using both of the settings (out of the nine using Stack Overflow, four could not complete the task). In Table I, we present the impact of Opiner while completing the tasks. To compute the Conversion rate, we only considered the developers that completed a task in both settings. For task 1, only 20% of the developers in SO setting selected the right API, but all of them picked the right API in SO+Opiner setting (i.e., 100% conversion rate). For task 2, only 66.7% of the developers in SO setting selected the right API, but all of them picked the right API when they used both Opiner and Stack Overflow (conversion rate = 100%). The developers also

reported higher confidence levels when making the selection using Opiner with Stack Overflow. For task 1, the confidence increased from 4.3 to 4.5 and for Task 2, it increased from 2.6 (Partially unsure) to 4.6 (almost fully confident).

V. Related Work

To the best of our knowledge, Opiner is the first tool developed to automatically mine and summarize opinions about APIs from developer forums. However, online developer forums have been studied extensively, e.g., to find dominant discussion topics [13], to analyze the quality of posts [14], developer profiles (e.g., personality traits of the most and low reputed users) [15], [16], or the influence of badges [17]. Tools utilized the knowledge from the forums, e.g., autocomment assistance [18], collaborative problem solving [19], and tag prediction [20]. Natural language summaries are produced for software documentation [21] and source code [22]. Storey et al. [23] analyzed tags and annotations in source code. The summarization of source code has been investigated by combining structural with lexical information of the source code [24], by correlating code contents with the call graphs in Java classes [22], by analyzing the identifier and variable names in methods [25], and by automatically documenting source code [26]. The selection and presentation of source code summaries were investigated through developer interviews [27] and eye tracking experiment [28]. Our findings confirm that developers also require different types of summaries of API reviews posted in developer forums.

VI. Summary

Given the plethora of opinions available for an API in various online developer forums, it can be challenging for a developer to make informed decisions about the API. We developed, Opiner, an online opinion search and summarization engine that presents summaries of opinions. We found that developers were interested to use our proposed summaries of API reviews and that while combined with Stack Overflow, Opiner helped developers to make the right decision with more accuracy and confidence. Our future work on Opiner will focus on the following extensions: 1) A unified framework to support the loading of opinions from disparate sources of developer forums, and 2) A refined recommendation engine to provide insights on APIs with features from different languages.

References

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

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

[3] G. Uddin and F. Khomh, "Mining aspects in API reviews," Polytechnique Montréal, Tech. Rep., May 2017.

[4] G. Uddin and M. P. Robillard, "Resolving API mentions in informal documents," McGill University, Tech. Rep., Sept 2017.

[5] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 168–177.

[6] G. Uddin and F. Khomh, "Automatic summarization of API reviews," in *In Proceedings of the of 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, p. 12.

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

[8] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment in short strength detection informal text," *American Society for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[9] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *ACM SIGKDD Knowledge Discovery and Data Mining*, 2004, pp. 168–177.

[10] H. D. Kim, K. Ganesan, P. Sondhi, and C. Zhai, "Comprehensive review of opinion summarization," University of Illinois at Urbana-Champaign, Tech. Rep., 2011.

[11] B. Liu, *Sentiment Analysis and Subjectivity*, 2nd ed. Boca Raton, FL: CRC Press, Taylor and Francis Group, 2010.

[12] H. D. Kim and C. Zhai, "Generating comparative summaries of contradictory opinions in text," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 385–394.

[13] 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*, pp. 1–31, 2012.

[14] 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*, 2014, pp. 342–354.

[15] 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)*, 2013, pp. 460–463.

[16] A. L. Ginsca and A. Popescu, "User profiling for answer quality assessment in Q&A communities," in *Data-driven user behavioral modelling and mining from social media*, 2013, pp. 25–28.

[17] 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*, 2013, pp. 95–106.

[18] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Automated Software Engineering*, 2013, pp. 562–567.

[19] Y. Tausczik, A. Kittur, and R. Kraut, "Collaborative problem solving: A study of math overflow," in *Computer Supported Cooperative Work and Social Computing*, 2014, pp. 355–367.

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

[21] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Trans. Software Eng*, vol. 40, no. 4, pp. 366–380, 2014.

[22] L. Moreno, J. Aponte, G. Sridhara, M. A., L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *Proceedings of the 21st IEEE International Conference on Program Comprehension (ICPC'13)*, 2013, pp. 23–32.

[23] M.-A. D. Storey, L.-T. Cheng, R. I. Bull, and P. C. Rigby, "Shared waypoints and social tagging to support collaboration in software development," in *CSCW*, pp. 195–198.

[24] A. M. S. Haiduc, J. Aponte, "Supporting program comprehension with source code summarization," in *Proceedings of the 32nd International Conference on Software Engineering*, 2010, pp. 223–226.

[25] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," in *Proc. 25th IEEE/ACM international conference on Automated software engineering*, 2010, pp. 43–52.

[26] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 21st IEEE International Conference on Program Comprehension*, 2014, pp. 279–290.

[27] A. T. T. Ying and M. P. Robillard, "Selection and presentation practices for code example summarization," in *Symposium on Foundations of Software Engineering*, 2014, pp. 460–471.

[28] P. Rodeghero, C. McMillan, C. McMillan, N. Bosch, and S. D'Mello, "Improving automated source code summarization via an eye-tracking study of programmers," in *Proc. 36th International Conference on Software Engineering*, 2014, pp. 390–401.