Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools

Preetha Chatterjee*, Kostadin Damevski†, Lori Pollock*,
Vinay Augustine‡, and Nicholas A. Kraft‡
* University of Delaware, Newark, DE, USA
{preethac, pollock}@udel.edu
† Virginia Commonwealth University, Richmond, VA, USA
kdamevski@vcu.edu
‡ABB Corporate Research, Raleigh, NC, USA
{vinay.augustine, nicholas.a.kraft}@us.abb.com

Abstract—**Modern software development communities are increasingly social. Popular chat platforms such as Slack host public chat communities that focus on specific development topics such as Python or Ruby-on-Rails. Conversations in these public chats often follow a Q&A format, with someone seeking information and others providing answers in chat form. In this paper, we describe an exploratory study into the potential usefulness and challenges of mining developer Q&A conversations for supporting software maintenance and evolution tools. We designed the study to investigate the availability of information that has been successfully mined from other developer communications, particularly Stack Overflow. We also analyze characteristics of chat conversations that might inhibit accurate automated analysis. Our results indicate the prevalence of useful information, including API mentions and code snippets with descriptions, and several hurdles that need to be overcome to automate mining that information.**

I. Introduction

Researchers have demonstrated that various software engineering tasks can be supported by mining information from emails, bug reports, tutorials, and Q&A forums. For example, information mined from emails and bug reports is used to re-document source code [1] or to recommend mentors in software projects [2]. Further, the natural language text in tutorials is analyzed to aid API learning [3], [4]. Over the years, researchers have also mined the knowledge embedded in Q&A forums, such as Stack Overflow, for supporting IDE recommendation [5]–[10], learning and recommendation of APIs [11]–[13], automatic generation of comments for source code [14], [15], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [16], [17]. These successes suggest that other kinds of developer communications may also provide information for mining-based software engineering tools.

Software developers are increasingly having conversations about software development via online chat services. In particular, developers are turning to public chat communities hosted on services such as Slack, IRC, Hipchat, Gitter, Microsoft Teams, and Freenode to discuss specific programming languages or technologies. Developers use these communities to ask and answer specific development questions, with the aim of improving their own skills and helping others. Over eight million active users participate daily on Slack, which is currently the most popular platform for these public chat communities and hosts many active public channels focused on software development technologies [18].

While chat communities share some commonalities with other developer communications, they also differ in intent and use. Overall, Q&A forum content is archival, while chat community content is transient. Q&A forums encourage longer, more in-depth questions that receive one or more well-thought-out answers. The question and its answers are subsequently read by many developers. Developer chats are typically informal conversations, with rapid exchanges of messages between two or more developers, where several clarifying questions and answers are often communicated in short bursts. Chats thus contain shorter, quicker responses, often interleaved with non-information-providing messages. The chat conversations are usually short-lived, often available only for a relatively short period of time[1], and over the life of the community, similar questions may be repeated (and answered) many times.

Most studies of developer chat communities have focused on learning about how they are used by development teams or for analyzing developer behaviors/interactions [19]–[24]. Panichella et al. [25] investigated how developer collaboration links differ across three various kinds of communication channels, including mailing lists, issue trackers, and IRC chat logs. However, limited work has focused on mining and extracting specific types of information from developer chats. For instance, Alkadhi et al. [23], [26] examined the frequency and completeness of available rationale in HipChat and IRC messages, and the potential of automatic techniques for rationale extraction. To the best of our knowledge, no previous work has investigated chat communities from the perspective of understanding what kind of information can be mined from them, how much of that information is available, and how difficult that information is to extract.

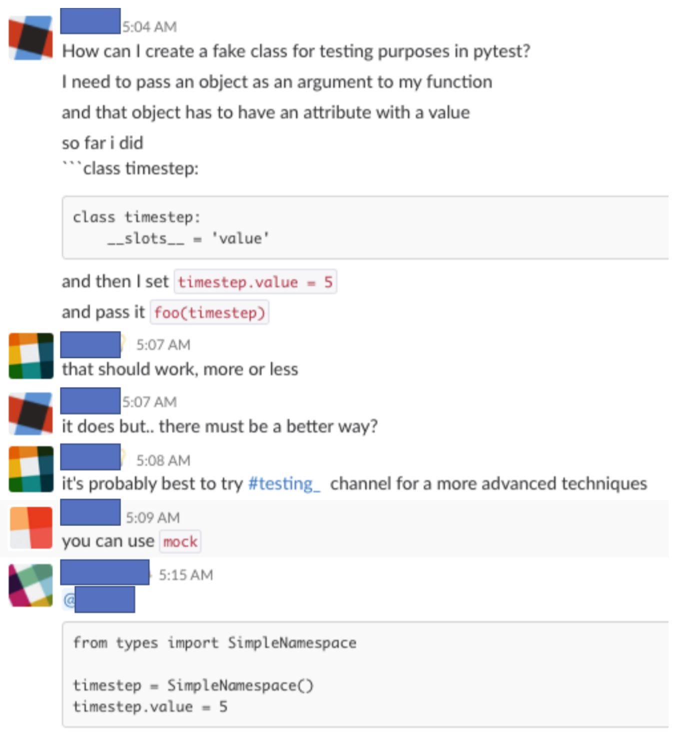
In this paper, we investigate the potential usefulness and challenges of mining developer Q&A chat conversations for

[1]For example, Slack's free tier only provides access to the most recent 10,000 chat messages.

supporting software maintenance and evolution tools in comparison with Q&A forums. We report on an exploratory study to compare the content in Q&A-focused public chat communities (hosted on Slack) to a Q&A-based discussion forum (Stack Overflow), because both resources share the intent of supporting learning and information sharing among developers.

We explore the availability and prevalence of information in developer Q&A chat conversations, which provides us with the first insight into the promise of chat communities as a mining source. We study the characteristics of information mined from the chat communities, e.g., analyzing the characteristics of embedded code snippets and of the natural language text describing the code snippets, and quantify the novel challenges in mining this data source. The overall goal is to gain insight into whether it is worth mining chat communities for information to support software maintenance and evolution tools, and whether there are additional opportunities from mining chat communities that are not offered by Q&A forums.

In total, we automatically analyzed 23,893 developer Q&A chat conversations from five programming communities on Slack and 825,294 posts labeled with the corresponding tags on Stack Overflow. Since some of the measures that we wanted to investigate for chat communities are not easily automated with high accuracy, we created data sets of 400 conversations in Slack and 400 posts in Stack Overflow for manual analysis. Our results indicate the prevalence of useful information in developer Q&A chats, and thus provide insights that will aid future research in using the available information.



[redacted] 5:04 AM
How can I create a fake class for testing purposes in pytest?
I need to pass an object as an argument to my function
and that object has to have an attribute with a value
so far i did
```class timestep:

```
class timestep:
    __slots__ = 'value'
```

and then I set `timestep.value = 5`
and pass it `foo(timestep)`

[redacted] 5:07 AM
that should work, more or less

[redacted] 5:07 AM
it does but.. there must be a better way?

[redacted] 5:08 AM
it's probably best to try #testing_ channel for a more advanced techniques

[redacted] 5:09 AM
you can use `mock`

[redacted] 5:15 AM
@[redacted]

```
from types import SimpleNamespace

timestep = SimpleNamespace()
timestep.value = 5
```

Fig. 1. Sample of a conversation in Slack (Python-dev community)

II. Background and Related Work

A. Chat Communities for Software Engineers

Public chats comprise multiple communities focused on particular topics such as a technology (e.g., Python or Ruby-on-Rails), with specific channels within a given community assigned to general discussion or to particular subtopics [27]. Within each channel, users participate in chat conversations, or chats, by posting messages. Chats in some channels follow a Q&A format, with information seekers posting questions and others providing answers, possibly including code snippets or stack traces, as shown in Figure 1.

Studies on chat communities have typically focused on learning about how they are used by development teams and the usefulness of the conversations for understanding developer behaviors. Shihab et al. [19] analyzed developer Internet Relay Chat (IRC) meeting logs to analyze the content, participants, their contribution and styles of communications. Yu et al. [20] conducted an empirical study to investigate the use of synchronous (IRC) and asynchronous (mailing list) communication mechanisms in global software development projects. Lin et al. [21] conducted an exploratory study to learn how Slack impacts development team dynamics. Lebeuf et al. [22] investigated how chatbots can help reduce the friction points that software developers face when working collaboratively. Paikari et al. [28] characterized and compared chatbots related to software development in six dimensions (type, direction, guidance, predictability, interaction style, and communication channel). Alkadhi et al. [23], [26] conducted exploratory studies to examine the frequency and completeness of available rationale in chat messages, and the potential of automatic techniques for rationale extraction. They also designed a tool for annotating chat messages that contain rationale in Slack [29]. The purpose of all of these analyses is to empirically study specific behavior of developers in chat channels, while ours is to holistically assess public Q&A chat as a mining source for improving software tools.

B. Mining Software Artifacts for SE Tools

For several years, researchers have been developing techniques to mine Stack Overflow and other software artifacts for information to help software developers. A key mechanism for providing the mined information is through recommendations in an Integrated Development Environments (IDE). Common recommendations include programming errors and exceptions [6], [7], [10], linking relevant discussions to any source code based on context [8], [9] or through a query in the IDE [5], [30].

Treude and Robillard proposed an approach for automatically augmenting API documentation with informative sentences from Stack Overflow [31]. Rahman et al. [12] proposed a technique that takes a natural language query as input and uses keyword-API associations mined from Stack Overflow to produce API recommendations as output. Other works for enriching API documentations include augmenting code examples [32], [33], identifying iOS and Android API classes
```

for which developers regularly face usage obstacles [13], and integrating crowdsourced frequently asked questions (FAQs) from the web into API documents [34].

Code snippets and their descriptive text in Stack Overflow have also been analyzed to automatically generate comments for source code in open source projects [14], [15]. Nagy and Cleve [35] mined code blocks that appear in Stack Overflow questions to identify common error patterns in SQL statements. Yang et al. [36] investigated the usability of code snippets on Stack Overflow. Badashian et al. [37] used developers' contributions to Stack Overflow as a proxy for their expertise for consideration during bug triaging.

Stack Overflow has also been analyzed to build thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [16], [17], [38]. Others have applied topic analysis and mining of domain-specific information [39], [40], exploring gender bias [21], [41], [42], and emotions [43], [44].

Stack Overflow includes built-in quality signaling in the form of up & down votes, accepted answers, and user reputation. Some researchers [5], [6], [8], [14], [15], [36] have used these quality signals to filter the posts that they use as input for their mining techniques. The need for such filtering partially motivates our study, as we want to understand, for instance, whether and how developers indicate accepted answers in chat communities.

To our knowledge, no research has focused on mining similar information from chat communities. However, Slack provides easy integration to other frequently used developer tools (e.g., Github, Bitbucket, JIRA, and Jenkins) through a set of conversation-based bots and apps. These bots and apps have been widely adopted by many developers for different software engineering tasks such as maintaining code quality, testing, conducting development operations, supporting customers, and creating documentation [22].

III. Exploratory Study

A. Research Questions

We designed our study to explore the potential of Slack and similar chat communities supporting Q&A conversations to serve as sources for mining information similar to Stack Overflow as mined for software engineering tool use, by seeking to answer the following questions:

RQ1: How prevalent is the information that has been successfully mined from Stack Overflow Q&A forums to support software engineering tools in developer Q&A chats such as Slack?

RQ2: Do Slack Q&A chats have characteristics that might inhibit automatic mining of information to improve software engineering tools?

B. Data Set Creation

1) Community Selection and Extraction: We established several requirements for dataset creation to reduce bias and threats to validity. To conduct an effective comparison between Q&A forums and chat communities, we needed to identify groups that primarily discussed software development topics, had a substantial collection of participants, and had a presence on both kinds of systems.

For this study, we chose Stack Overflow and Slack as modern, popular instantiations of Q&A forums and developer chat communities, respectively. We selected four programming communities with an active presence on both systems: clojure, elm, python and racket. Within those selected communities, we focused on channels that follow a Q&A format.

Because programmatic access to the data in Slack communities is controlled by the administrators of the Slack team, we contacted several public Slack teams and asked for an API token that would allow us to read and store their data. Public Slack teams typically use Slack's free tier, which only stores the most recent 10,000 messages. Thus, for each Slack community that we selected for our study, we downloaded all of the discussion data from each channel every day for 489 days (June 2017- November 2018). Since Stack Overflow regularly publishes all of its data in XML format via the Stack Exchange Data Exchange, we extracted posts for this study from the 27-August-2017 release of the Stack Exchange Data Dump [45].

2) Documents, Posts, Conversations, and Disentanglement: To answer our research questions, we needed to curate data from both developer Q&A chats and Q&A forums. The first decision is what constitutes a "document" for analysis in each of these communities.

In a Q&A forum, we use a question and its corresponding multiple answers, votes, and comments, as the basis for a document. For our analysis, we limit a document to be the question and the two most popular answers (based on votes). We omit further, less relevant, answers to the question which are more likely to be unnoticed by developers and to contain noise. While the two most popular answers often include the *accepted* answer, this is not always the case. In Stack Overflow, the original questioner indicates which answer is the accepted one. If the vote total differs from this, it is because other users have found a different answer more useful or because the context around the question has changed (necessitating a new answer).

The most reasonable equivalent to a Q&A forum post within a chat community is a single conversation that follows a Q&A format with someone seeking information and others providing answers in chat form. However, messages in chats form a stream, with conversations often interleaving such that a single conversation thread is entangled with other conversations, thus requiring preprocessing to separate, or disentangle, the conversations for analysis.

The disentanglement problem has been studied before in the context of IRC and similar chat platforms [46]. We leveraged the technique proposed by Elsner and Charniak [47] that learns a supervised model based on a set of features between pairs of chat messages that occur within a window of time of

TABLE I
DATA SETS AND SIZES

Community	**# Conversations**		**Community**	**# Posts**	
(Slack Channels)	$Slack_{auto}$	$Slack_{manual}$	(StackOverflow Tags)	$StackOverflow_{auto}$	$StackOverflow_{manual}$
clojurians#clojure	5,013	80	clojure	1,3920	80
elmlang#beginners	7,627	80	elm	1,019	160
elmlang#general	5,906	80	-	-	-
pythondev#help	3,768	80	python	806,763	80
racket#general	1,579	80	racket	3,592	80
Total	23,893	400	Total	825,294	400

each other. The features include the elapsed time between the message pair, whether the speaker in the two messages is the same, occurrence of similar words, use of cue words (e.g., hello, hi, yes, no), and the use of technical jargon. For the training set, we manually disentangled a set of 500 messages from each channel and trained the model using the combined set.

After we observed that some Slack channels can become dormant for a few hours at a time and that participants can respond to each other with considerable delay, we modified Elsner and Charniak's algorithm to compute features between the current utterance and every utterance that 1) occurred ¡= 1477s prior to it, or 2) is within the last 5 utterances observed in the channel We also enhanced the set of features used by Elsner and Charniak, introducing several specific to Slack, for instance, the use of emoji or code blocks within a message. We also followed the procedure prescribed by Elsner and Charniak to create a better set of technical words for the model by extracting all of the words occurring in Stack Overflow documents tagged with a particular tag that do not co-occur in the English Wikibooks corpus. To measure the accuracy of the disentangling process, we manually disentangled a separate set of 500 messages from the python-dev channel. The model with our enhancements produced a micro-averaged F-measure of 0.79; a strong improvement over the vanilla Elsner and Charniak approach's micro-averaged F-measure of 0.57. We corrected or removed all poorly disentangled conversations prior to our annotation.

3) Curating a Slack-Stack Overflow Comparison Dataset: To enable comparison study between related communities on Stack Overflow and Slack, we first selected Slack programming communities and then identified related Stack Overflow communities by extracting all of the posts tagged with the language name from the Stack Exchange Data Dump. Table I shows the programming language communities (channels on Slack and tags on Stack Overflow) used as subjects of study. A key concern in comparing Slack and Stack Overflow was that the Slack conversations may not contain the same distribution of subtopics within a particular technology as Stack Overflow (e.g., if most conversations on pythondev#help were on Django but not a significant proportion of 'python' tagged Stack Overflow questions). To address this concern, we curated our comparison dataset by organizing the extracted Slack conversations and Stack Overflow posts by similar subtopics, using the following workflow:

1) Remove URLs and code snippets from the Slack and Stack Overflow datasets. Filter out terms that appear in more than 80% of documents or in less than 15 documents.
2) Train a Latent Dirichlet Allocation (LDA) topic model using the larger Stack Overflow data. By manually examining topics for interpretability and coherence, we chose to extract 20 LDA topics in each community.
3) Remove background topics, if they exist, which are very strongly expressed in the corpus (strongest topic for $>$ 50% of documents) and do not express technical terms.
4) Infer topic distributions for each document (i.e., post in Stack Overflow and conversation in Slack).
5) Select documents that strongly express the same LDA topic in Stack Overflow and Slack.

Throughout this paper, we refer to these comparison datasets as $StackOverflow_{auto}$ and $Slack_{auto}$. As Table I shows, $Slack_{auto}$ consists of 23,893 conversations, while $StackOverflow_{auto}$ contains 825,294 posts.

4) Datasets for Manual Analysis: Some of the measures we wanted to investigate for chat communities are not easily automated with high accuracy. Thus, we created subsets of $Slack_{auto}$ & $StackOverflow_{auto}$ which we call $Slack_{manual}$ & $StackOverflow_{manual}$, respectively. In order to obtain statistical significance with confidence of $95\% \pm 5\%$, we sampled 400 conversations for $Slack_{manual}$ and 400 posts for $StackOverflow_{manual}$, distributing the samples equally across the different communities.

Two of the coauthors computed the measures in section III-C on the $Slack_{manual}$ dataset using the following process. We wanted to insure that the inter-rater agreement is sufficient to allow two researchers to compute measures separately without duplication, to create a larger manual set with confidence. First, both authors computed all the measures on a shared set of 60 Slack conversations on $Slack_{manual}$. Using this initial shared set, we computed the Cohen's Kappa inter-rater agreement metric between the two authors, observing an agreement of more than 0.6 for each measure, which is considered to be sufficient [48]. Choosing a shared sample of 60 conversations also ensured the sample produced high confidence in the computed value of Cohen's Kappa based on the number of categories in our measures [49]. Second, the authors divided and separately computed the measures in the remaining manual samples to reach a total of 400 conversations in $Slack_{manual}$ and 400 questions in $StackOverflow_{manual}$.

Table I includes the number of conversations from each Slack channel and number of posts from each Stack Overflow community.

C. Methodology

Conversations in Slack following different discussion formats might require different mining strategies. In this paper, we focus on Q&A conversations to compare with Stack Overflow Q&A forums as both are knowledge sharing. To gain a sense of the prevalence of Q&A conversations on the Slack channels we monitored, we computed the total number of conversations that follow a Q&A format and contain discussions that are about software-specific issues using $Slack_{manual}$. We found that a high percentage (91.50%) of conversations in our Slack data set were of Q&A format, containing discussions about issues in software.

We also conducted a qualitative analysis using Zagalsky et al.'s knowledge types, which they developed to compare Stack Overflow to the R-help mailing list [50]. Namely, we computed the occurrence of questions, answers, updates, flags, and comments on our channels to determine whether Slack Q&A conversations are similar in structure to Stack Overflow Q&A. In the context of Q&A chat conversations, 'question' represents the primary question which initiates the main topic of discussion; 'answer' provides a solution to the primary question; 'update' requests a modification to a question or answer; 'comment' provides clarification to a specific part of the question or answer; and 'flag' requests moderator attention (e.g., off-topic or repeated questions, spam).

We found the following number of knowledge types in each of these categories in our Slack data set: (#Questions(393), #Answers(447), #Updates(159), #Comments(1947), and #Flags(19)). These counts indicate that in our $Slack_{manual}$ data set of 400 conversations, almost every conversation contains at least one question and more answers than questions, with some updates and flags, and a large number of comments compared to questions and answers.

In the remainder of this section, we describe the measures that we used to investigate each of our research questions. We also explain how we computed each measure either automatically or manually.

RQ1: How prevalent is the information that has been successfully mined from Stack Overflow Q&A forums to support software engineering tools in developer Q&A chats such as Slack?

To answer this question, we focused on information that has been commonly mined in other software artifacts. Specifically, we analyzed code snippets, links to code snippets, API mentions, and bad code snippets. To provide context, we also collected data on document length in Slack and Stack Overflow. Table II shows the measures computed.

The first step in analysis is identifying the particular elements in Slack and Stack Overflow documents, (e.g., code snippet, link, API mention). Since the text of each Slack document is stored in the Markdown text format, we used regular expressions to extract the code snippets and the URLs of links to code snippets from each conversation. Since the text of each Stack Overflow post is stored as HTML, we used the HTML parser, Beautiful Soup [51], to extract the code and link elements from each post. We wrote Python scripts to analyze conversations and posts for all the automatically collected information shown in Table II. Next, we describe how and why we computed each measure for RQ1.

First, we compute the document lengths for context in understanding the prevalence of the measures described below. *Document length* is defined as the number of sentences in a document (i.e., a Slack conversation or a Stack Overflow post). We computed this measure on the natural language text in each document using the sentence tokenizer from NLTK [52].

High occurrence of code snippets within a forum indicates more opportunity to support various software engineering tasks such as IDE recommendation, code clone detection, and bug fixing. *Code snippet count* is computed as the number of code snippets per document, for which we counted all inline and multiline code snippets in a document. The answers that solve the original question on Stack Overflow mostly contain multiple line snippets, so the measure can be an indicator of snippet quality [53]. *Code snippet length* is the number of non-whitespace characters in each code snippet. For multiline snippets, we also counted the number of non-blank lines in each snippet.

URLs that link different software artifacts are useful to establish traceability between documents, which has been shown to be useful for tasks such as automatic comment generation. Specifically, by establishing mappings between code and associated descriptive text, it is possible to automatically generate descriptive comments for similar code segments in open-source projects [54]. We counted all URLs to Gist and Stack Overflow in a document. For instance, for *Gist links*, we counted all links to http://gist.github.com. For *Stack Overflow links*, we counted all links to either stackexchange.com or stackoverflow.com. The $StackOverflow_{auto}$ dataset also includes relative links in this measure.

High frequency of API mentions in code and text show potential to build tools such as API linking and recommendation and augmenting API documentation [55]–[57]. *API mentions* in code snippets and text are the numbers of APIs mentioned in the code snippets and in the natural language text, respectively, for each document.

Bad code snippets could be mined to build tools for debugging and bug fixing, testing and maintenance, e.g., in designing test cases for mutation testing [58], [59]. Percentage of *bad code snippets* is defined to be the number of "bad" code snippets divided by the total number of (inline and multiline) code segments in a document. Bad code snippets are defined as code segments that are described by negative words or phrases such as "ugly," "error", "does not work", or "horrible." These

TABLE II
MEASURES FOR RQ1. THE MEASURES LABELED $auto$ ARE COMPUTED BY APPLYING SCRIPTS TO THE $Slack_{auto}$ AND $StackOverflow_{auto}$ DATASETS, WHEREAS THE MEASURES LABELED $manual$ ARE COMPUTED MANUALLY USING THE $Slack_{manual}$ AND $StackOverflow_{manual}$ DATASETS.

Measure	**Definition**	**Datasets used**
Document length	number of sentences in a chat conversation or Q&A post	$auto$
Code snippet count	number of code snippets in a document	$auto$
Code snippet length	number of characters in a code snippet	$auto$
Gist links	number of links to Gist in a document	$auto$
Stack Overflow links	number of links to Stack Overflow in a document	$auto$
API mentions in code snippets	number of APIs mentioned in code snippets in a document	$manual$
API mentions in text	number of APIs mentioned in non-code text in a document	$manual$
Bad code snippets	percentage of erroneous code snippets in a document	$manual$

descriptors indicate that one or more of the forum or chat participants find that the code snippet does not implement the desired functionality, is inefficient, or has poor readability or syntax errors. We manually searched for the presence of such negative indicators in the natural language text to identify and count bad code snippets in the documents. We did not evaluate the code snippet itself for badness, as we were interested in bad code snippets based on participant interpretation, not our annotators' interpretations.

RQ2: Do Slack Q&A chats have characteristics that might inhibit automatic mining of information to improve software engineering tools?

To answer this question, we focused on measures that could provide some insights into the form of Slack Q&A conversations (participant count, questions with no answer, answer count) and measures that could indicate challenges in automation (how participants indicate accepted answers, questions with no accepted answer, natural language text context pertaining to code snippets, incomplete sentences, noise within a document, and knowledge construction process) that suggest a need to filter. All measures shown in table III were computed manually on the $Slack_{manual}$ dataset, except *Participant count*, which could be computed with high accuracy automatically with scripts. Manual analysis was performed by two of the authors. Since the research question investigates challenges in mining information in developer chat communications to support software engineering tools, we only computed the measures on Slack.

Conversations with high participant counts suggest richness of information since they generally contain different opinions, additional information, and examples. To compute *Participant count*, we counted the number of unique users in each Slack conversation. For example, a conversation with 10 messages posted by 3 users has a participant count of 3. We report the minimum, median, and maximum participant frequencies over all Slack conversations in $Slack_{auto}$.

Questions without answers are not helpful for developers seeking help online on specific programming tasks. Higher percentage of questions with no response also undermines the potential of chat communications as a mining resource for software engineering tools. *Questions with no answer* in $Slack_{manual}$ was computed by counting the number of disentangled conversations that do not have any answer in response to the initial question in the conversation. We report the measure as a percentage of all conversations in $Slack_{manual}$.

Higher answer count indicates variety to the pool of solutions, which gives the developers an option to select the answer most specific to the problem in context. However, identifying multiple answers is non-trivial as it requires identifying the responsive sentences and the question to which those sentences are responding. For computing *Answer count*, we manually determined which sentence mapped to a given question and counted the number of solutions proposed in response to the initial question in the conversation that explained the problem. Specifically, we calculated the number of Zagalsky [50] "Answer" type of artifact in a conversation, and computed the minimum, median, and maximum number of answers per conversation in $Slack_{manual}$.

Accepted answers are an indicator of good quality information. Higher percentage of accepted answers provides confidence to leverage the information for both developers and software engineering tools. We computed the *Questions with no accepted answer* percentage as a ratio of the number of conversations with no accepted answers, to the total number of conversations in $Slack_{manual}$. Questions are deemed not to have an accepted answer if they lack an explicit acceptance indicator.

Unlike Stack Overflow, Slack does not provide an option to accept a suggested solution in the platform. However, the correct solutions are often followed by positive emojis, such as 🙂. In addition, there are prevalent textual clues that indicate that the suggested solution worked, and/or helped in solving the problem being discussed in the conversation. By manually analyzing all the natural language text in $Slack_{manual}$ conversations, we built a list of words/phrases/emojis that participants used as acceptance indicators. The potential reasons for no acceptance of answers could be incorrectness/inefficiency of the suggested solutions, the user's discretion to acknowledge and show acceptance, etc.

Embedded code snippets in Slack conversations are surrounded by natural language (NL) text that describes various code metadata such as the functionality being implemented, data structures, code complexity and effectiveness, errors and

TABLE III
MEASURES FOR RQ2.

Measure	Definition	Datasets used
Participant count	Number of participants in a conversation	$Slack_{auto}$
Questions with no answer	Percentage of conversations that have no answer	$Slack_{manual}$
Answer count	Number of answers for a question in a conversation	$Slack_{manual}$
Indicators of accepted answers	List of emojis and textual clues used in conversations/votes in Q&A posts	$Slack_{manual}$
Questions with no accepted answer	Percentage of conversations with no accepted answer	$Slack_{manual}$
NL text context per code snippet	Number of NL sentences related to a code snippet	$Slack_{manual}$
Incomplete sentences	Percentage of incomplete sentences describing code in a conversation	$Slack_{manual}$
Noise in document	Percentage of conversations containing noise	$Slack_{manual}$
Knowledge construction process	Percentage of conversations in each type of knowledge construction (participatory, crowd)	$Slack_{manual}$

exceptions. *NL text context per code snippet* is defined as the number of sentences used to describe a particular code snippet. Extracting such sentences or phrases that provide additional information about a code segment embedded in a forum is valuable to software engineering tasks such as automatic comment generation and augmentation of documentation [15], [54], [57]. Identifying such sentences is not straightforward due to the informal and intertwined nature of utterances in conversations. Thus, we manually analyzed all the sentences in the disentangled conversations to count the number of NL sentences related to a particular code snippet. We report the minimum, median, and maximum counts of sentences describing code per code segment in conversations of $Slack_{manual}$.

Developer chat communications are informal by nature. Therefore, conversations include incomplete sentences, which potentially makes mining of essential information difficult. We computed the percentage of *Incomplete sentences* as the number of incomplete sentences describing code divided by the total sentences describing code in a conversation. For this measure, we are not considering all the incomplete sentences in a conversation, but only the incomplete sentences that describe code.

Presence of many types of information makes it more challenging to disentangle conversations and automatically classify and extract specific types of information. In addition, noise in conversations makes mining more challenging. *Noise* are sentences that do not provide useful information for mining e.g., 'What do you mean?', 'Yes.', 'Any idea?'. We report the percentage as the ratio of conversations which contain noise to the total number of conversations in $Slack_{manual}$.

We also examined the approach to constructing knowledge as the percentage of conversations belonging to each type of knowledge construction in $Slack_{manual}$ dataset, per Zagalsky's definitions - participatory versus crowd knowledge construction [50]. In participatory knowledge construction, answers are contributed by multiple users in the same thread. Participants enrich each others' solutions by discussing several aspects such as the pros and cons of each answer, different viewpoints, additional information, and examples. In crowd knowledge construction, participants add solutions individually, without discussing other solutions, thereby creating a resource of independent solutions. The *knowledge construction process* is computed as the percentage of conversations belonging to each category of knowledge construction in $Slack_{manual}$ dataset.

D. Threats to Validity

Construct validity: As with any study involving manual human analysis, there might be some cases where the humans may have incorrectly analyzed the documents. To limit this threat, we ensured that the authors who analyzed the manual data set had considerable experience in programming and qualitative analysis, and followed a consistent coding procedure that was piloted in advance. We computed the Cohen's Kappa inter-rater agreement metric between the two sets of results, observing an agreement of more than 0.6 for each measure, which is considered to be sufficient [48].

Internal validity: In examining the automatically disentangled conversations, we observed occasional errors, the most egregious of which resulted in orphaned sequences of one or two messages, which poses a threat to internal validity. We mitigated this problem in the manual dataset by performing a validation step, filtering out conversations that we were able to detect as poorly disentangled. However, the threat may still exist for some metrics and could have resulted in some imprecision.

In addition, since we are considering the question and two most popular answers for Stack Overflow, we may be missing relevant information in the remaining answers, For instance, the *Bad code snippets* measure could be affected by not including unpopular answers, which could contain a higher proportion of bad code snippets (as the reason for their lack of popularity).

External validity: We selected the subjects of our study from Stack Overflow and Slack, which are the most popular systems for Q&A forums and chat communities, respectively. Our study's results may not transfer to other Q&A forums or chat platforms. To mitigate this threat, we selected four active programming language communities for our study. There is a broad array of topics related to a particular programming language; we also used LDA to curate the Slack-Stack Overflow comparison dataset, thus ensuring that we compared Q&A posts and conversations about similar topic areas (in case the two sources were unbalanced in their emphasis on topics).

It is also possible that our smaller datasets for manual analysis are not representative of the full corpus of Slack conversations or Stack Overflow posts for a given community.

TABLE IV
STATISTICAL RESULTS FOR RQ1 (SLACK VS. STACK OVERFLOW).

Measure	U	p-**value**
Document length	162.0	< 0.001
Code snippet count	90.5	< 0.001
Code snippet length	1549.5	< 0.001
Gist links	3551.5	0.04
Stack Overflow links	335.0	< 0.001
API mentions in code snippets	32815.5	< 0.001
API mentions in text	101361.0	< 0.001
Bad code snippets	19235.5	< 0.001

The size of these datasets was chosen to give us a statistically representative sample, feasible for our annotators to analyze. However, scaling to larger datasets for the manual analysis might lead to different results.

E. Findings

In this section, for each research question, we first report the quantitative results of our study and then discuss the implications of those results.

RQ1: How prevalent is the information that has been successfully mined from Stack Overflow Q&A forums to support software engineering tools in developer Q&A chats such as Slack?

For RQ1, we display the results primarily as box plots in Figure 2. The box plots for *Gist links*, *Stack Overflow links*, and *Bad code snippets* would be difficult to read because the median of the metric was 0. Therefore, we present the results for *links* as an aggregate in Figure 2d. For *Bad code snippets*, despite the medians being 0, the mean for Stack Overflow (21.2) was more than double the mean for Slack (9.4).

Table IV presents the results of Mann Whitney U tests comparing our measures using the Slack and Stack Overflow datasets. All tests are two-tailed, checking the null hypothesis that *the metric values for Slack and Stack Overflow do not differ significantly*. The 'U' and 'p' columns list the U statistic and the corrected p-value, respectively. For all measures, we can reject the null hypothesis (at $p < 0.05$) of no difference between the two sources.

Much of the information mined from Stack Overflow is also available on Slack Q&A channels. Approaches for mining of Stack Overflow for software maintenance and evolution tools focus on code snippets and API mentions in code and text, thus we focus on those results here. For all of these measures, we found that Slack conversations indeed contain all of these items — code snippets, and API mentions in code and text. Thus, chat communities provide availability of all of this information for mining.

However, most of this information, with the exception of API mentions, is available in larger quantities on Stack Overflow. For instance, Figure 2b indicates that Stack Overflow posts can have a much larger number of code snippets than Slack conversations. This suggests that code snippets could be mined from chat communities, but more conversations than posts would need to be analyzed to extract similar numbers of code snippets from Slack. We also observed this as we were curating our datasets for this study.

As shown in Figure 2c, the median code snippet lengths from both sources are similar, although the variation of snippet length is much larger for Stack Overflow code snippets than those included in Slack conversations. Thus, if one wants to mine sources where longer snippets might be available, this data suggests to mine Stack Overflow rather than Slack.

Conversations and posts are the most natural unit of granularity for representing a document in our analysis of chats and Q&A forums, respectively. Our data shows that median conversation length is indeed shorter than median post length with statistical significance. Recall that we define a Stack Overflow post as being only the question and the top two most-voted answers. This shows that Slack conversations are likely to require more external context or assumptions when mined.

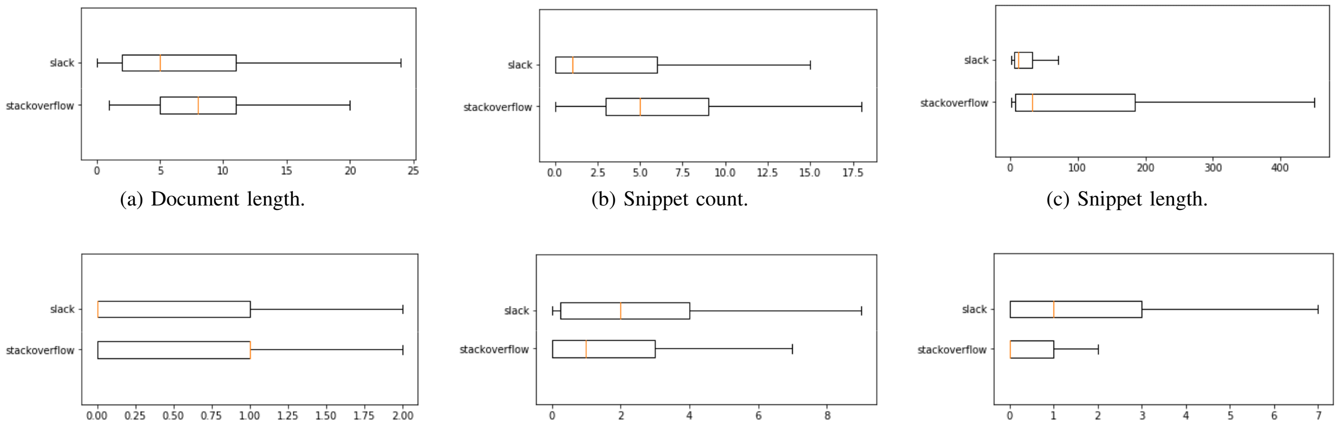
API mentions are available in larger quantities on Slack Q&A channels. We did observe statistically significant evidence that there are more API mentions in Slack conversations in general than in Stack Overflow documents. While both sources had a fairly low median occurrence of API mentions in text, Slack had a higher value and more variance. During our manual analysis, we also observed that APIs are mentioned often in Slack conversations, as many conversations centered around API use. This result suggests that Slack could be used for mining based on API mentions, similar to Stack Overflow, but perhaps providing even a richer and more interesting source.

Links are rarely available on both Slack and Stack Overflow Q&A. Before the study, we anticipated that developers on Slack would often use links to answer questions, saving time by pointing askers to an existing information source, such as Stack Overflow. Alternatively, we expected askers to use Gist to post code prior to asking questions, in order to benefit from the clean formatting that enables the display of a larger block of code. While both of these behaviors did occur, they were fairly infrequent and occur with a higher frequency on Stack Overflow than on Slack.

RQ2: Do Slack Q&A chats have characteristics that might inhibit automatic mining of information to improve software engineering tools?

Table V and Table VI present the results for RQ2. Table V shows all the natural language clues (words, phrases, and emojis) used as accepted answer indicators over all analyzed Slack communities. The most prevalent indicator is "Thanks/thank you", followed by phrases acknowledging the participant's help such as "okay", "got it", and other positive sentiment indicators such as "this worked", "cool", and "great". Accepted answers were also commonly indicated using emojis as listed in the table.

Table VI presents the results for the remaining measures. Results represented as percentages are reported directly, while other results, computed as simple counts, are reported as



(a) Document length. (b) Snippet count. (c) Snippet length.

(d) URL count. (e) API mentions in snippets. (f) API mentions in text.

Fig. 2. Box plots of RQ1 measurements by community.

TABLE V
INDICATORS OF AN ACCEPTED ANSWER IN $Slack_{manual}$ DATASET.

Words/Phrases: good find; Thanks for your help; cool; this works; that's it, thanks a bunch for the swift and adequate pointers; Ah, ya that works; thx for the info; alright, thx; awesome; that would work; your suggestion is what I landed on; will have a look thank you; checking it out now thanks; that what i thought; Ok; okay; kk; maybe this is what i am searching for; handy trick; I see, I'll give it a whirl; thanks for the insight!; thanks for the quick response @user, that was extremely helpful!; That's a good idea! ; gotcha; oh, I see; Ah fair; that really helps; ah, I think this is falling into place; that seems reasonable; Thanks for taking the time to elaborate; Yeah, that did it; why didn't I try that?
Emojis: 🙂; 👍; 💡; 🎉

TABLE VI
FINDINGS FOR RQ2. CELL DATA IS OF THE FORM
$Min < Median < Max$ OR A PERCENTAGE.

Measure	**Datasets Used**	**Results**
Participant frequency	$Slack_{auto}$	$1 < 2 < 34$
Questions with no answer	$Slack_{manual}$	15.75%
Answer frequency	$Slack_{manual}$	$0 < 1 < 5$
Questions with no accepted answer	$Slack_{manual}$	52.25%
NL context (No. sentences) per snippet	$Slack_{manual}$	$0 < 2 < 13$
Incomplete sentences describing code	$Slack_{manual}$	12.63%
Noise in document	$Slack_{manual}$	10.5%
Knowledge construction	$Slack_{manual}$	61.5% crowd; 38.5% participatory

minimum < median < maximum. For example, participant frequency in $Slack_{topic}$ ranged from a single participant in a conversation to 34 participants, with a median of 2 participants. A single participant conversation was one with no answers. Based on the results reported in Table VI, approximately 16% of questions go unanswered, while the maximum number of answers was five for a question. Of the questions with answers, approximately 52% of questions have no accepted answer.

Code snippets have from 0 to 13 sentences that contained some kind of descriptive information about the code snippet. The results also indicate that the number of incomplete sentences describing code is low, 13%, and similarly the noise in a conversation can be as high as 11%. We also observed that 61.5% conversations followed an approach of crowd knowledge construction, while the rest 38.5% were participatory [50], where multiple developers collaborate to settle on an answer to a challenging question. To gain insight into the semantic information provided in Slack conversations, we analyzed the kinds of information provided in the conversations. Using the labels defined by Chatterjee et al. [60], we categorized the information as providing context in terms of design, efficiency, erroneous nature of the code, explanatory description of the functionality of the code, or a detail about the structure of the code. Figure 3 presents the prevalence of each type of information in our $Slack_{manual}$ dataset.

The largest proportion of Slack Q&A conversations discuss software design. We observed that the most prevalent types of information on Slack is "Design". This aligns with the fact that the main purpose of developer Q&A chats is to ask and answer questions about alternatives for a particular task, specific to using a particular language or technology. Often the focal point of conversations are APIs, mentions to which we observed occur often on Slack Q&A channels, where a developer is asking experts on the channel for suggestions on API or proper idioms for API usage. Conversations that were "Explanatory" often involved a developer explaining, in long form, the lineage of a particular decision in the language or technology, or contrasting the capabilities of different languages, while "Structure" conversations often focused around a specific data structure (e.g., representing a dictionary in Elm) or control

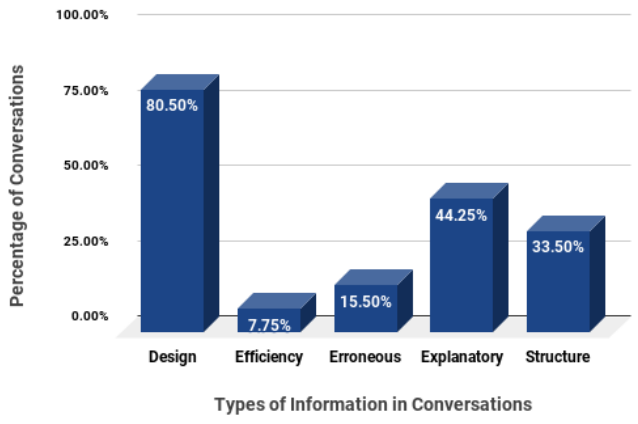


Fig. 3. Percentage conversations of each type in $Slack_{manual}$

structure (e.g., how code can be split in multiple files in Clojure).

Accepted answers are available in chat conversations, but require more effort to discern. The existence of an accepted answer, selected by the question asker using a subsequent response, present the clearest indication of a Slack conversation's quality. There is a significant proportion of accepted answers available in Slack, based on Table VI. The median number of answers is only one, which suggests that there are expert people on the channels, such that only one answer is given and it is accepted. One might think it is then easy to automatically identify the answer to a given question, especially with the use of emojis and the word/phrase indicators of accepted answers. However, an automatic mining tool needs to automatically identify the sentence in a conversation that is an answer to a question and which question it is answering. This implies that NLP techniques and sentiment analysis will most likely be needed to automatically identify and match answers with questions. Due to the speed of Slack conversations, we observed cases where the question asker accepts an answer, only to come back a few minutes later to declare that the answer did not help. Therefore, mining techniques have to be able to track the conversation to the final accepted answer.

Participatory conversations provide additional value but require deeper analysis of conversational context. Nearly 40% of conversations on Slack Q&A channels were participatory, with multiple individuals working together to produce an answer to the initial question. These conversations present an additional mining challenge, as utterances form a complex dependence graph, as answers are contributed and debated concurrently. However, the discussion also holds numerous interesting insights about specific technologies or APIs, as the developers are continually criticizing and improving upon existing suggestions.

IV. Conclusions and Future Work

In this paper, we reported on an exploratory study to investigate the potential of developer Q&A chats in Slack as a mining resource for software maintenance and evolution tools. We found that Q&A chats provide, in lesser quantities, the same information as can be found in Q&A posts on Stack Overflow. However, Q&A chats generally provide more information on API mentions than do Q&A posts.

At first sight, one might believe that the informal nature and interleaved conversations of Slack would make it difficult to mine automatically. There is also no pre-defined notion of conversation; each could span from two messages to hundreds. However, our work to implement this study required us to disentangle conversations. We found that adapting the technique and training sets can indeed achieve high accuracy in disentangling the Slack conversations.

The availability of complete sentences and indicators of accepted answers suggest that it is feasible to apply automated mining approaches to chat conversations from Slack. However, the lack of inbuilt formal Q&A mechanisms on Slack does result in some potential mining challenges. Notably, identifying an accepted answer is non-trivial and requires both identifying the responsive sentence(s) and the question to which those sentences are responsive. Part of this challenge stems from the free-form style of chat conversations, in which a question may be followed by a series of clarification or follow-up questions, and contextual clues must be followed to determine which question is ultimately being answered.

Our study reveals several new opportunities in mining chats. While there were few explicit links to Stack Overflow and GitHub Gists in our dataset, we believe that information is often duplicated on these platforms, and that answers on one platform can be used to complement the other. Future work includes further investigating this linking between public Slack channels to Stack Overflow.

Participatory Q&A conversations are available on Slack on large quantities. These conversations often provide interesting insights about various technologies and their use, incorporating various design choices. While such conversations have also been observed on mailing lists, on Slack, the availability and speed of such conversations is much higher. Because of this, as future work, we intend to investigate mining such conversations for software development insights.

During our study, we also observed that developers use Slack to share opinions on best practices, APIs, or tools (e.g., API X has better design or usability than API Y). Stack Overflow explicitly forbids the use of opinions on its site. However, it is clear that receiving opinions is valuable to software developers. The availability of this information in chat may lead to new mining opportunities for software tools. As our future work, we plan to investigate the mining of opinion statements available in public Slack channels.

Acknowledgment

We acknowledge the support of the National Science Foundation under grant no. 1812968, 1813253 and the DARPA MUSE program under Air Force Research Lab contract no. FA8750-16-2-0288.

REFERENCES

[1] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, June 2012, pp. 63–72.

[2] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?" in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 44:1–44:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393647

[3] H. Jiang, J. Zhang, Z. Ren, and T. Zhang, "An unsupervised approach for discovering relevant tutorial fragments for apis," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 38–48. [Online]. Available: https://doi.org/10.1109/ICSE.2017.12

[4] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining api types using text classification," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 869–879. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818859

[5] L. B. L. de Souza, E. C. Campos, and M. d. A. Maia, "Ranking crowd knowledge to assist software development," in *Proc. 22nd Int'l Conf. on Program Comprehension*, May 2014, pp. 72–82.

[6] M. Rahman, S. Yeasmin, and C. Roy, "Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions," in *Proc. IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering*, Feb. 2014, pp. 194–203.

[7] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proc. 3rd Int'l Wksp. on Recommendation Systems for Software Engineering*, May 2012, pp. 85–89.

[8] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," in *Proc. 11th Working Conf. on Mining Software Repositories*, May 2014, pp. 102–111.

[9] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing Stack Overflow for the IDE," in *Proc. 3rd Int'l Wksp. on Recommendation Systems for Software Engineering*, May 2012, pp. 26–30.

[10] V. Amintabar, A. Heydarnoori, and M. Ghafari, "ExceptionTracer: A solution recommender for exceptions in an integrated development environment," in *Proc. IEEE 23rd Int'l Conf. on Program Comprehension*, May 2015, pp. 299–302.

[11] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions — incorporating relational and categorical knowledge into word embedding," in *Proc. IEEE 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering*, Mar. 2016, pp. 338–348.

[12] M. Rahman, C. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in *Proc. IEEE 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering*, Mar. 2016, pp. 349–359.

[13] W. Wang and M. Godfrey, "Detecting API usage obstacles: A study of iOS and Android developer questions," in *Proc. 10th Working Conf. on Mining Software Repositories*, May 2013, pp. 61–64.

[14] E. Wong, J. Yang, and L. Tan, "AutoComment: Mining question and answer sites for automatic comment generation," in *Proc. 28th IEEE/ACM Int'l Conf. on Automated Software Engineering*, 2013, pp. 562–567.

[15] M. M. Rahman, C. K. Roy, and I. Keivanloo, "Recommending insightful comments for source code using crowdsourced knowledge," in *Proc. IEEE 15th Int'l Working Conf. on Source Code Analysis and Manipulation*, Sep. 2015, pp. 81–90.

[16] Y. Tian, D. Lo, and J. Lawall, "Automated construction of a software-specific word similarity database," in *Proc. IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering*, Feb. 2014, pp. 44–53.

[17] C. Chen, Z. Xing, and X. Wang, "Unsupervised software-specific morphological forms inference from informal discussions," in *Proc. 39th Int'l Conf. on Software Engineering*, 2017, pp. 450–461.

[18] T. S. P. Statista, "https://www.statista.com/statistics/652779/worldwide-slack-users-total-vs-paid/," 2018.

[19] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer irc meetings in open source projects," in *2009 IEEE International Conference on Software Maintenance*, Sept 2009, pp. 147–156.

[20] L. Yu, S. Ramaswamy, A. Mishra, and D. Mishra, *Communications in Global Software Development: An Empirical Study Using GTK+ OSS Repository*, 2011, pp. 218–227.

[21] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik, "Why developers are slacking off: Understanding how software teams use Slack," in *Proc. 19th ACM Conf. on Computer Supported Cooperative Work and Social Computing Companion*, 2016.

[22] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "How software developers mitigate collaboration friction with chatbots," in *Proc. 20th ACM Conf. on Computer-Supported Cooperative Work and Social Computing*, 2017.

[23] R. Alkadhi, T. Lata, E. Guzmany, and B. Bruegge, "Rationale in development chat messages: An exploratory study," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 436–446.

[24] S. A. Chowdhury and A. Hindle, "Mining StackOverflow to filter out off-topic IRC discussion," in *Proc. IEEE/ACM 12th Working Conf. on Mining Software Repositories*, May 2015, pp. 422–425.

[25] S. Panichella, G. Bavota, M. D. Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 251–260.

[26] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How do developers discuss rationale?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, vol. 00, March 2018, pp. 357–369. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SANER.2018.8330223

[27] M.-A. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, 2017.

[28] E. Paikari and A. van der Hoek, "A framework for understanding chatbots and their future," in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '18. New York, NY, USA: ACM, 2018, pp. 13–16. [Online]. Available: http://doi.acm.org.udel.idm.oclc.org/10.1145/3195836.3195859

[29] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge, "React: An approach for capturing rationale in chat messages," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Nov 2017, pp. 175–180.

[30] C. Greco, T. Haden, and K. Damevski, "StackInTheFlow: Behavior-driven recommendation system for Stack Overflow posts," in *Proceedings of the International Conference on Software Engineering*, 2018.

[31] C. Treude and M. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *Proc. IEEE/ACM 38th Int'l Conf. on Software Engineering*, May 2016, pp. 392–403.

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

[33] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Enriching Documents with Examples: A Corpus Mining Approach," *ACM Trans. Inf. Syst.*, vol. 31, no. 1, pp. 1:1–1:27, Jan. 2013.

[34] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced FAQs into API documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 456–459.

[35] C. Nagy and A. Cleve, "Mining Stack Overflow for discovering error patterns in SQL queries," in *Proc. IEEE Int'l Conf. on Software Maintenance and Evolution*, Sep. 2015, pp. 516–520.

[36] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: An analysis of stack overflow code snippets," in *Proc. IEEE/ACM 13th Working Conf. on Mining Software Repositories*, May 2016, pp. 391–401.

[37] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," in *Proc. IEEE Int'l Conf. on Software Maintenance and Evolution*, Sep. 2015, pp. 506–510.

[38] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S. W. Lin, "HDSKG: Harvesting domain specific knowledge graph from content of webpages," in *Proc. IEEE 24th Int'l Conf. on Software Analysis, Evolution and Reengineering*, Feb. 2017, pp. 56–67.

[39] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang, "An empirical study on Stack Overflow using topic analysis," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 446–449.

[40] I. K. Villanes, S. M. Ascate, J. Gomes, and A. C. Dias-Neto, "What are software engineers asking about Android testing on Stack Overflow?" in *Proceedings of the 31st Brazilian Symposium on Software Engineering*, 2017, pp. 104–113.
[41] S. Morgan, "How are programming questions from women received on Stack Overflow? a case study of peer parity," in *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2017, pp. 39–41.
[42] D. Ford, J. Smith, P. Guo, and C. Parnin, "Paradise unplugged: Identifying barriers for female participation on Stack Overflow," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 846–857.
[43] N. Novielli, F. Calefato, and F. Lanubile, "Towards discovering the role of emotions in Stack Overflow," in *Proceedings of the 6th International Workshop on Social Software Engineering*, 2014, pp. 33–36.
[44] 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*, 2015, pp. 33–40.
[45] S. Exchange, "Stack exchange data dump," https://archive.org/details/stackexchange, 2017, [Online; accessed 2017-08-27].
[46] D. C. Uthus and D. W. Aha, "Multiparticipant chat analysis: A survey," *Artificial Intelligence*, vol. 199, pp. 106–121, 2013.
[47] M. Elsner and E. Charniak, "You talking to me? a corpus and algorithm for conversation disentanglement," in *Proc. Association of Computational Linguistics: Human Language Technology*, 2008, pp. 834–842.
[48] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.
[49] M. A. Bujang and N. Baharum, "A simplified guide to determination of sample size requirements for estimating the value of intraclass correlation coefficient: a review." *Archives of Orofacial Science*, vol. 12, no. 1, 2017.
[50] A. Zagalsky, D. M. German, M.-A. Storey, C. G. Teshima, and G. Poo-Caamao, "How the R community creates and curates knowledge: An extended study of Stack Overflow and mailing lists," *Empirical Software Engineering*, 2017.
[51] L. Richardson, "Beautiful soup," https://www.crummy.com/software/BeautifulSoup/, 2017, [Online; accessed 2017-05-07].
[52] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.
[53] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: An analysis of stack overflow code snippets," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 391–402. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901767
[54] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 562–567. [Online]. Available: https://doi.org/10.1109/ASE.2013.6693113
[55] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q a discussions – incorporating relational and categorical knowledge into word embedding," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 338–348.
[56] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 349–359.
[57] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 392–403.
[58] Q. Gao, H. Zhang, J. Wang, Y. Xiong, L. Zhang, and H. Mei, "Fixing recurring crash bugs via analyzing q amp;a sites (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2015, pp. 307–318.
[59] F. Chen and S. Kim, "Crowd debugging," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 320–332. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786819
[60] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? an exploratory study," in *Proc. 24th IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering*, Feb. 2017.