Automatically Identifying the Quality of Developer Chats for Post Hoc Use

PREETHA CHATTERJEE, University of Delaware, USA
KOSTADIN DAMEVSKI, Virginia Commonwealth University, USA
NICHOLAS A. KRAFT, UserVoice, USA
LORI POLLOCK, University of Delaware, USA

Software engineers are crowdsourcing answers to their everyday challenges on Q&A forums (e.g., Stack Overflow) and more recently in public chat communities such as Slack, IRC and Gitter. Many software-related chat conversations contain valuable expert knowledge that is useful for both mining to improve programming support tools and for readers who did not participate in the original chat conversations. However, most chat platforms and communities do not contain built-in quality indicators (e.g., accepted answers, vote counts). Therefore, it is difficult to identify conversations that contain useful information for mining or reading, i.e,. conversations of post hoc quality. In this paper, we investigate automatically detecting developer conversations of post hoc quality from public chat channels. We first describe an analysis of 400 developer conversations that indicate potential characteristics of post hoc quality, followed by a machine learning-based approach for automatically identifying conversations of post hoc quality. Our evaluation of 2000 annotated Slack conversations in four programming communities (python, clojure, elm, and racket) indicates that our approach can achieve precision of 0.82, recall of 0.90, F-measure of 0.86, and MCC of 0.57. To our knowledge, this is the first automated technique for detecting developer conversations of post hoc quality.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; • **Information systems → Collaborative and social computing systems and tools**.

Additional Key Words and Phrases: online software developer chats, quality of social content

ACM Reference Format:
Preetha Chatterjee, Kostadin Damevski, Nicholas A. Kraft, and Lori Pollock. 2020. Automatically Identifying the Quality of Developer Chats for Post Hoc Use. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 26 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Studies show that modern software development communities are increasingly social with developers contributing to and leveraging crowd-sourced knowledge and using new community-based tools, including GitHub, Stack Overflow and Slack [71]. Public chat communities hosted on services such as Slack, IRC, Gitter, and Microsoft Teams are now commonly used for developer Question & Answer (Q&A) discussions. Unlike intra-organizational/small group use of chat services, numerous people participate in public software-related chats to gain knowledge or help others, similar to Q&A forums like Stack Overflow. Communication in public chats often follows a Q&A format, with information seekers posting questions and others providing answers, possibly including code snippets or stack traces [18]. Our earlier study indicates that Q&A chats in Slack provide the same information as can be found in Q&A posts on Stack Overflow [18]. This suggests that chats can also serve as a resource for mining-based software engineering tools.

Q&A content can be leveraged to enhance the result quality of search engines, identify domain experts, and enrich the knowledge bases of Q&A services, chat bots and discussion forums [41]. Thus, as researchers have shown for Q&A forums, Q&A chats may also be mineable for information to support IDE recommendation, [9, 56, 60], learning and recommendation of APIs [20, 59, 76], automatic generation of comments for source code [61, 77], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [21, 73]. Furthermore, while Q&A forums such as Stack Overflow explicitly forbid posting of questions that ask for opinions or recommendations, developers regularly use chat forums to share opinions on best practices, APIs, and tools [18]. Recent research has shown that mined opinions are valuable to software developers [49]. Additionally, developer chats have already served as a mining source for identifying feature requests [64] and extracting developer rationale [5]. Apart from supporting knowledge gathering, chat conversations could be potentially used in dialog research, such as curating a training corpus for virtual assistants that support software development tasks. For this purpose, researchers have released large developer chat datasets [17, 31, 50]. Lowe et al. [50] used several steps to ensure the quality of the chat corpus, such as discarding long conversations and questions that did not generate a response. Ehsan et al. [31] corrected misspelled words that could negatively impact the quality of the text analysis in their corpus.

Understanding the quality of the information in the mining source is essential for building effective data-driven software tools. From our preliminary analyses (Section 2 and 3), we found that conversations in public chats vary significantly in quality. Specific questions and answers may be poorly formed or lacking in important context. In addition, conversations may be personal and lack any relevant software-related information. So outcomes, in terms of quality exchange of information, are highly variant on the channel and moment in time. Relative to other developer communications such as Stack Overflow, where quality feedback is explicitly signaled (in the form of accepted answers, vote counts, or duplicate questions), in Q&A chats, quality feedback is signaled in the flow of the conversation, mostly using textual clues or emojis [13]. There is no formal mechanism for voting or accepting an answer. While researchers have proposed ways to assess the quality of information in Q&A forums beyond built-in mechanisms of the websites, (e.g., conciseness of answers, containing contextual information, or code readability) [10, 29, 67, 80], to our knowledge, there are no known techniques to automatically assess the quality of the content in developer chat conversations.

In this research, our goal is to automatically identify information in public chat channels that would be useful to software engineers beyond the conversation participants, i.e., *conversations of post hoc quality, containing useful information for mining or reading after the conversation has ended.* We contribute a suite of techniques for automatically identifying post hoc quality developer conversations. Our techniques can be applied as a quality filter mechanism to identify chats that are suitable to serve as a learning or mining source for building task-based software applications such as API recommendation systems, virtual assistants for programming/debugging help, and FAQ generation. The mining tools could leverage our techniques to discard lower quality chat conversations, thereby reducing the tool's input data overload and producing faster and effective results. Our work could also help readers in efficient information gathering by saving time and ensuring high-quality information, thus enhancing developer productivity.

Our first step was to understand what contributes to *post hoc quality* of developer chats. We conducted an analysis of 400 Slack developer chats covering five different programming channels. We recruited participants to annotate judgements of the quality of the chats based on the ease of gaining useful software-related knowledge, and then we manually analyzed the characteristics of chats rated as higher quality. The results indicated that only 251 of 400 conversations were deemed to be of high-quality by a majority of raters. The fact that many conversations were not deemed high-quality further motivated us to investigate automatic techniques to determine conversations of *post hoc quality*. We leveraged the observations from our manual analysis to form hypotheses about the characteristics of *post hoc quality* conversations that could be used to build automated techniques for quality identification.

We developed what we believe is the first approach to automatically identify conversations of *post hoc quality* on developer chat platforms. Our approach uses machine learning-based classifiers based on a set of features closely related to the hypothesized characteristics of *post hoc quality* conversations. In this work, we focused on public Slack channels as Slack has over ten million daily active users, and is currently a popular platform for these public chat communities that hosts many active public channels on software development technologies [69]. We evaluated our techniques on a set of 2000 manually annotated Slack developer conversations. Specifically, we focused on answering the following research questions:

- RQ1: How effective are machine learning-based techniques for automatic identification of *post hoc quality* developer chats?
- RQ2: Which classifiers and features result in more effective automatic identification?
- RQ3: What types of conversations are difficult to automatically detect as *post hoc quality* using our techniques?

We report precision, recall, F-measure, AUC, and MCC and conduct qualitative error analysis. Our results indicate that *post hoc quality* conversations can be identified with precision of 0.82, recall of 0.90, F-measure of 0.86, and MCC of 0.57.

2 MOTIVATIONAL EXAMPLES

Developer public chats are becoming increasingly important corpora for several applications including, understanding topic trends of developer discussions [7, 31, 32, 48, 66], extracting feature requests [64] and developer rationale [5], and building virtual assistants for software engineers [50]. In contrast to many other sources of software development-related communication, the information on chat forums is shared in an unstructured, informal, and asynchronous manner. Chat communications typically consist of rapid exchanges of messages between two or more developers, where several clarifying questions and answers are often communicated in short bursts. Hence, developer chats are context-sensitive. Understanding the context of conversations is crucial towards extraction of relevant information. This led us to consider the entire conversation instead of single utterances as the granularity for assessing *post hoc quality*. This section presents two concrete examples of the varying post hoc quality of chat conversations from our manual analysis study of Slack conversations (see Section 3), as potential sources for both problem solving and API-related mining purposes, among others.

Example 1 (Problem-solving conversations): To demonstrate the variance in *post hoc quality* of developer chats, we first show a pair of example conversations related to solving programming problems, in Table 1. For readability, the conversations in Table 1 are shown as already disentangled (more details on disentanglement in Section 4.1). We observe that conversation (1a), related to solving programming problems, is concise and succinct, contains contextual details of the problem and the suggested solution, and shows indication of answer acceptance. Thus, conversation (1a) is easy to read and understand, and indicators of answer acceptance give the readers a sense of validation and confidence in the correctness of the information. In contrast, conversation (1b) contains too many back and forth questions, lacks contextual details in the starting question, and includes no indication of answer acceptance. Thus, conversation (1b)'s characteristics make it difficult to extract specific information for either software engineers or mining tools. While

Table 1. Example Problem-Solving Conversations on *pythondev#help* (For readability, shown as already disentangled)

(a) Post hoc Quality Conversation

<table>
<tr><th>Author</th><th>Utterance</th></tr>
<tr><td>Alexia</td><td>Hi, I have a file with following contents 1234 alphabet /vag/one/arun > 1454 bigdata /home/two/ogra > 5684 apple /vinay/three/dire, but i want the output to be like 1234 alphabet one > 1454 bigdata two > 5684 apple three</td></tr>
<tr><td>Elaina</td><td>tee hee 🙂
sed -r 's|(.+)/[^/]+/([^/]+)/.+|\1\2|g'</td></tr>
<tr><td>Corina</td><td>Even though I dont have anything to do with this question, could you explain the logic behind the answer? The formatting sentence seem so random</td></tr>
<tr><td>Elaina</td><td>'sed -r' is an extended mode, so that + is enabled (matches one or more characters, unlike * that matches zero or more); s///g or s|||g or any symbol instead of | is how a basic replacing expression is constructed. The first field is what to match, the second is what to replace it with.;
sed -r 's|(.+)/[^/]+/([^/]+)/.+|\1\2|g'
(.+)/ matches anything from the start until the first / and puts found characters in the first group (\1); [^/]+/ matches anything that is not a slash, and then a slash ('vag/' or 'home/'); ([^/]+)/ matches the same thing, but puts the stuff found in-between slashes in the second group \2; and then .+ matches whatever comes next to the end of line; and the second field tells sed to replace the line with \1\2, so our saved groups side-by-side: the first group was everything before the first slash, and the second group was the stuff between 2nd and 3rd slashes</td></tr>
<tr><td>Corina</td><td>Ah ok, thanks a lot for the explanation!</td></tr>
</table>

(b) Non Post hoc Quality Conversation

Author	Utterance
Cody	Hello guys I got a huge problem
Holli	Cody: ask away
Cody	We've been ask as assignment the implementation of Dijkstra's and Bellman Ford's algorithm for calculating the shortest path in a given graph
Holi	So what's the issue?; run into a problem?
Cody	I don't really know how to start and that's my problem
Holli	Well were you given code to start off with as far as i will give you this input in order to build the graph you have with the path weights and what not?
Cody	The Input to the algorithms is an oriented graph with weights
Holli	So then you have to build the graph in the code, yes?
Cody	More precisely compute the shortest path from a given source to all the other nodes
Holli	I understand that, I am just asking if you have anything to build off of like code given to you to complete the task or if you have to start from scratch and build the graph in code then calculate it.
Cody	You need to code the algorithm from scratch and supply it with a graph in a text file and it'll calculate the shortest path in it
Holli	can you give an example input
Darrin	Cody: how much experience do you have writing code?; for example, there are quite a number of existing examples of the algorithms you're talking about
Cody	basic i'm just starting
Darrin	ok; can you describe the steps on how you execute the algorithm?; and do you understand why those steps are necessary?; if so, then the next step you take is translating your written description of the process into pseudocode; once you have a reasonable sequence of actions, you then implement the pseudocode in your language of choice; frankly, the first two items are always the most difficult; because it requires you to understand the problem domain; once you understand it, making it work is usually much less effort
Rachel	Cody: oof graph theory for a beginner. do you understand how those algorithms work, ?
Cody	Yes I understand how those work
Darrin	just having trouble translating described steps to code?

conversation (1b) could be useful for researchers studying developer communications (e.g., confusion detection [30]), it might not be suitable for task-based tools such as question and answer (Q&A) extraction.

Example 2 (API-related conversations): Our previous exploratory study shows prevalence of API-related information in developer chats [18]. Conversations containing API mentions (e.g., API X has better design or usability than API Y) or API caveats, could be useful for developers (beyond the original participants in the conversation) reading and learning about APIs. It is also possible to build API recommendation systems by leveraging the information in such conversations. Before building applications that extract API-related information, ensuring completeness and credibility of information in the source is crucial. Table 2 shows a pair of conversations that contain discussions pertaining to APIs. We observe that conversation (2a) contains explanations and examples related to usage of *round()* in different versions of the programming language. Two potential solutions are offered through the conversation, and their credibility,

Table 2. Example API-related Conversations on *pythondev#help* (For readability, shown as already disentangled)

(a) Post hoc Quality Conversation

Author	Utterance
Carylon	Hi, guys is there one option to use round function and rounding the value for example 2.59 – show 2.60
Darrin	https://docs.python.org/2/library/functions.html#round
Carylon	On documents there is a note about that: note The behavior of round() for floats can be surprising: for example, round(2.675, 2) gives 2.67 instead of the expected 2.68. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See Floating Point arithmetic: Issues and Limitations for more information.
Darrin	yup. what's the question?
Carylon	look this function: $qtd\_lata = round(120/(18 * 3), 2)$; returns: 2.0; But if i don't use round function
Darrin	are you using python 2 or 3?
Velva	Related: Division changed in python: https://www.python.org/dev/peps/pep-0238/; You can just make any of those numbers a float
Carylon	Python 2.7.9
Darrin	"'$In[5] : 120/(18 * 3) Out[5] :>>> 120/(18 * 3) 2.222\ldots$"'
Velva	'$120.0/(18 * 3)$'
Rachael	or 'from__future__ import division'
Carylon	>>> from__future__ import division >>> $140/(18 * 3) 2.59259\ldots$; in this case i need 3; understood; If i use round; its show 2.0
Rachael	if you dont want to switch versions of python, use 'from __future__ import division' or change one value to a float; what happens when you called 'round' after using 'from__future__ import division'?
Carylon	try run this:; from __future__ import division round(140 / (18 * 3), 2); the result is 2.59; i need in this case rounding to 3.0
Rachael	the second argument to 'round' is how many significant digits to give, you're asking it for 2; hence it being 2.59; remove the second argument to the round function to get a whole number
Carylon	perfect
Rachael	"' from __future__ import division round(140 / (18 * 3)) "'; should equal 3.0
Carylon	very good man; so easy; ahauuaa

(b) Non Post hoc Quality Conversation

Author	Utterance
Herbert	I have something seemingly complex and puzzling. I'm trying to add dynamic localization to my library which also powers a REST aPI. I was able to add the gettext() function by running this at my root before importing the other modules $< code\_segment >$ That works great. However, I want to add dynamic localization for certain functions. Ex: "' #using system foo() #using given locale foo(lang='es') # Goes back to system foo(). I'm trying to do this by running within a context switcher: $< code\_segment >$ But I'm getting this opaque error and can't figure out how to isolate it $< error\_trace >$. The interesting part is that it works on the first call to foo but not the second. My guess is that something is going on after the 'yield' which screws the system up, but I'm not sure where, why, or how to ask the right question; OH MY GOD. I've been testing this in the REPL. I have a function which returns a boolean that gets 'thrown away', but the REPL assigns it to '_' which overwrites the global '_' set by 'gettext.install()'; Thoughts on how to make gettext REPL-safe?
Desiree	Herbert: don't use 'install'?

relatedness and completeness can be understood through the flow of the discussion. API-related information from this conversation could be potentially extracted or summarized to augment existing API documentations. In conversation (2b), although the question contains sufficient details on the problem at hand, no concrete solution with explanation is offered by the other participant. Therefore, conversation (2b) is not a suitable source of information gathering, i.e., demonstrates *non post hoc quality*.

Both of the above pairs of problem-solving (Table 1) and API-related (Table 2) conversations exhibit the need of a mechanism for assessing quality of content in developer chats. Thus, based on previous research in determining quality of other kinds of developer communications, we investigate a more systematic and software engineering (SE)-specific quality assessment approach for automatically determining *post hoc quality* developer chat conversations.

3 TOWARDS DISCERNING POST HOC QUALITY CONVERSATIONS

This section describes the details of our data-driven approach to determining the characteristics inherent of *post hoc quality* chat conversations. As mentioned earlier, we conducted an analysis of 400 Slack conversations with conversation non-participants judging the informational value of conversations. This analysis helped guide us to answer: *What* *characteristics are inherent in software-related conversations that software engineers beyond the participants find useful (i.e., post hoc quality conversations)? What are the primary characteristics of non post hoc quality conversations?*

Dataset: We established several requirements for dataset creation to reduce bias and threats to validity. To curate an analysis dataset that is representative of the quality of chats that are both public and software-related, we identified chat groups that primarily discussed software development topics and had a substantial number of participants. We chose chat groups that had at least one new participant per week, and had at least 100 participants to avoid analyzing conversations with only a few participants which would have potential to not be representative. Since we chose public software-related chats as the subjects of our study, we avoided channels that focus on personal conversations within a small group. We selected five channels from four programming communities (two channels from elmlang, and one from each of pythondev, clojurians, and racket) with an active presence on Slack, and were willing to provide us API tokens for downloading chats for research purposes. Within those selected communities, we focused on public channels that follow a Q&A format, i.e. a conversation typically starts with a question and is followed by a discussion potentially containing multiple answers or no answers. While the Q&A format is not strictly enforced by a moderator in the selected channels, each channel on Slack is intended for a particular purpose. For example, the pythondev Slack community has a channel 'announcements' intended for users to share information about events (and thus does not follow a Q&A format), while the channel 'help' is used to ask and answer questions. The channels are advertised on the Web and allow anyone to join, with a joining process only requiring the participant to create a username and a password. Once joined, on these channels, participants can ask or answer any question, as long as it pertains to the main topic (e.g., programming in Python). In order to obtain statistical significance with confidence of 95% ± 5%, we sampled 400 conversations from Slack. We used a sample size of 400 corresponding with the statistical measure - confidence level of 95% and margin of error of 5%, since our dataset is sampled from 40k public software-related chat conversations on Slack [17]. We specifically extract a subset of 400 randomly chosen developer conversations [1] from our previously released dataset [17]. Our analysis dataset of 400 developer conversations consists of an equal distribution of conversations across all channels, for generalizing our observations across all conversations in the channels used in the study, i.e., 80 conversations from pythondev#help channel, 80 from clojurians#clojure, 80 from racket#general, 80 from elmlang#beginners, and 80 from elmlang#general.

Procedure: We recruited human judges (students from graduate courses) with prior experience (2+ years) in programming and in using Slack, but no knowledge of our research focus. We designed instructions for the human judges and conducted a pilot study to test the annotation procedure while noting their annotation time. The judges were asked:

(1) *How would you rate the quality of this conversation, based on the ease that you found it to gain useful software-related knowledge? a) Poor b) Average c) Good.*
(2) *Why do you think this conversation is 'Poor' or 'Average' or 'Good'? Justify your rating.*

To account for potential subjectivity, each conversation was analyzed by three judges independently. Each judge took approximately one hour to annotate 20 conversations. Based on the timing results and the need for 1200 (400 x 3) annotations, 60 judges were each assigned 20 randomly selected conversations, with 4 conversations from each programming channel in our dataset. We used specific ratings instead of a Likert scale (e.g., how likely is the conversation to be of post hoc quality), and understand that different users may have different thresholds for what is considered post hoc quality. Therefore, to capture the overall consensus on quality, we used majority voting instead of inter-rater agreement to determine the annotation for each conversation.

[1]https://bit.ly/3dkgA9g

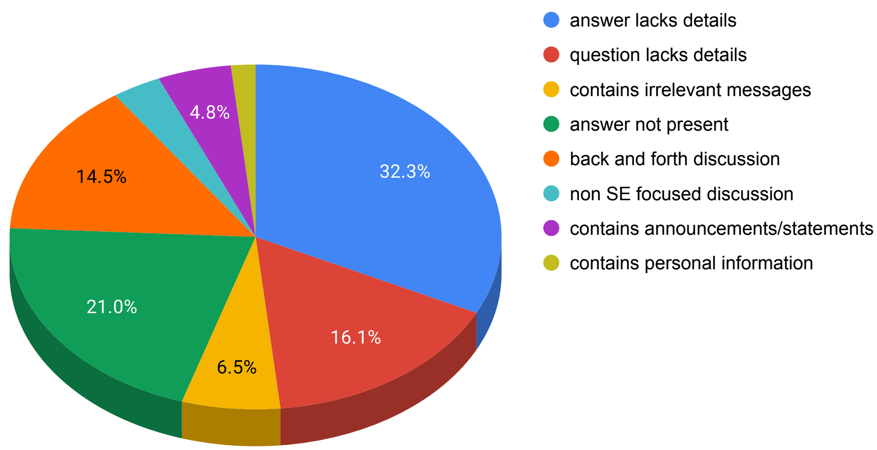


Fig. 1. Study Responses Suggesting Characteristics of Poor Quality Conversations

We followed a quantitative and qualitative analysis procedure [63] to analyze the ratings of the Slack conversations. The analysis procedure consisted of the following steps: (1) First, we used majority voting to segregate good, average, and poor quality conversations. We used percentage occurrence to measure the distribution of conversations in each category. (2) Following a qualitative content analysis procedure, two authors of this paper independently studied the responses to the second question of the survey. More specifically, we used the technique of Explanation Building [63], where patterns were identified based on cause-effect relationships between the ratings and the underlying explanations from the judges' responses. For instance, let us consider two judge's responses to a good-quality conversation in the study: "Good because a solution has been suggested." , "solution was given with instructions on how to do it". From these responses, we deduced that the presence of an answer (or solution) to the initial question in the conversation contributes to post hoc quality information. We wrote memos for our findings from each response to the second question of the survey to facilitate the process of analysis, such as recording observations on characteristics of good and poor content, researcher reflections, and additional information (if applicable). We thematically categorized responses by structure (question and answers) and content (e.g., code, topic of discussion) characteristics. We also manually analyzed the participants' disagreements and conversations to identify the primary characteristics of conversations in each category. The two authors then met to discuss and group common observed characteristics of good and poor quality conversations. The analysis was performed in an iterative approach comprised of multiple sessions, which helped in generalizing both of the annotators' observations from previous sessions to the characteristics of good and poor quality conversations. (3) Next, we examined the conversations in our dataset to study the occurrence of structure and content-related characteristics in the good and poor quality conversations. Specifically, we quantitatively analyzed the presence of (a) question *i.e., if the conversation starts with a software-related question*, (b) answers *i.e., if the conversation contains an answer that is found useful by the questioner, often denoted with phrases such as "thanks", "that worked", etc. [18]*, (c) code snippets *i.e., if the conversation contains embedded code*, and (d) code descriptions *i.e., if the conversation contains descriptions related to the embedded code.*

Observations: We observed that 349 out of 400 (87.25%) conversations received majority agreement among human judges. Out of 349 conversations, 251 conversations were marked as good quality, and 98 conversations were marked as poor. The rest of the conversations either did not receive a majority voting (e.g. {Poor, Good, Average}), or were marked average (neither good nor poor) by majority voting (e.g. {Average, Average, Average}); and thus not considered for our analyses since we focused on binary quality labels (good or poor) for this study because they provide stronger signals. The high percentage of good conversations in our dataset indicates that developer conversations can be useful to software engineers beyond the participants.

Our manual analysis of the judges' responses from the second question in the study suggests characteristics inherent in *post hoc quality* (good) and *non post hoc quality* (poor) conversations, respectively. From the conversations that were marked 'poor', we observed that a conversation is not considered to be *post hoc quality* if it has any of the following characteristics, supported by example responses from the study: 1) the conversation-initiating question lacks details and context, e.g., *"The question is not clear, it contains a block of code but the issue with the code is not stated"*, 2) the conversation contains announcements or statements not relevant to future readers, e.g., *"There is no relevant information about django or python learnt from this conversation."*, 3) the answer to the question is not present, or is delivered in a way that reflects reservations or low confidence, e.g., *"The responder does not answer the question."*, 4) the answer is too short and lacks relevant details thus providing little to no value to the questioner, e.g., *"Convoluted usage scenario; no explanation for a basic case usage is provided and why actually it is valid to do so."*, 5) the conversation contains too much back and forth discussion, misspellings and poor grammar, thus making it hard to identify useful information, e.g., *"The conversation jumps from one question to another..."*, 6) the conversation contains irrelevant messages, thus making it difficult to read and understand, e.g., *"The conversation is not easy to read and complex in nature..."*, 7) the topic of discussion contains personal information that does not have value to readers, e.g., *"discussed her/his current level of understanding of a language(python) ... doesn't pertain to all programmers..."*, 8) the discussion is not on a technical topic, e.g., *"It doesn't follow any specific software related topic nor provide any insights."*. Figure 1 shows the distribution of the observed characteristics of poor quality conversations in the study responses. We summarize the characteristics of both post hoc and non post hoc quality conversations at the end of this section.

Analysis of the conversations that received majority agreement showed that:

- 200/251 (80%) good-quality conversations vs. 65/98 (66%) poor-quality conversations contain a question in the first utterance.
- 204/251 (81%) good-quality conversations vs. 48/98 (49%) poor-quality conversations contain any accepted answer.
- 89/251 (35%) good-quality conversations vs. 17/98 (17%) poor-quality conversations contain code.
- 71/251 (28%) good-quality conversations vs. 11/99 (11%) poor-quality conversations contain descriptions of embedded code.

Summary: Observations from the post-conversation analysis led us to describe a *post hoc quality* conversation as follows: *A conversation that contains information that could be useful to other users, whether in the chat channel or elsewhere. A conversation is considered post hoc quality based on the availability and ease of identifying information that could help a person to gain useful software-related knowledge.*

Post hoc quality characteristics include containing: (PH1) discussion related to software and programming topics, (PH2) specific questions with relevant details and context, (PH3) one or more answers to questions (as text/code/references to other resources), (PH4) explanation of technical concept or suggested code or proposed solution.

Non post hoc quality characteristics include containing: (NPH1) discussion unrelated to software and programming topics, (NPH2) a question lacking relevant details and context, or no question or problem to be addressed, (NPH3) no answer or explanation of proposed solution, (NPH4) personal information.

4 AUTOMATICALLY IDENTIFYING POST HOC QUALITY CONVERSATIONS

Based on our studies of Slack software-related chats, we developed a suite of techniques for automatically identifying *post hoc quality* developer conversations. Automatic identification takes as input a segment of a software-related chat channel collected over some time period, executes a conversation disentanglement process to extract individual conversations from the interleaved chats, and classifies each conversation as *post hoc quality* or *non post hoc quality*.

4.1 Conversation Disentanglement

Since messages in chats form a stream, with conversations often interleaving such that a single conversation thread is entangled with other conversations, preprocessing is required to separate, or disentangle, the conversations for analysis. The disentanglement problem has been addressed by researchers in the context of IRC and similar chat platforms [33, 75]. In earlier work [17, 18], we modified Elsner and Charniak's algorithm [33] to customize it for modern developer chats by: 1) using a significantly larger window of messages, 2) computing the disentanglement graph on the last five messages regardless of elapsed time, and 3) introducing new features specific to Slack, for instance, the use of emoji or code blocks in a message. For disentangling an IRC chat transcript, Elsner and Charniak [33] used a certain number of messages (i.e., the window of messages) as candidates to form a separate conversation. 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. Hence, we expanded the window of messages by using a union of 1) messages within a time threshold (1477 seconds), and 2) the prior 5 messages. Our approach to chat disentanglement can achieve relatively better performance than the off-the-shelf Elsner and Charniak algorithm, with a micro-averaged F-measure of 0.80.

In this paper, we use a subset of our previously released disentangled chat dataset, more details about the disentanglement algorithm can be found in our previous work [17]. Each utterance of a conversation contains metadata such as timestamp and author information. Additionally, for each conversation in our dataset, we rerun the released disentanglement code to generate a disentanglement graph, which represents relationships between pairs of utterances, and we use the graph when computing *post hoc quality* features.

4.2 Machine Learning-based Classification

We investigated several supervised machine learning-based approaches to automatically identify *post hoc quality* conversations. We describe the conversation features followed by the suite of machine learning algorithms investigated for this classification task.

4.2.1 Features and Feature Extraction. We present five sets of features by theme, that map to our definition of *post hoc quality*: Knowledge Seeking/Sharing (PH2, PH3, NPH2, NPH3), Contextual (PH3, PH4, NPH2, NPH4), Succinct (PH4, NPH2), Well-written (NPH2), and Participant Experience (PH1, PH4, NPH1). Table 3 lists the features in each set with their value range; descriptions of why and how we extract each feature follow.

Knowledge Seeking/Sharing: Software-related chats can vary widely in their content based on their intent, such as for personal, team-wide or community-wide communication [48]. A proposed knowledge source built from chats for mining-based software engineering tools and human information-seeking readers would be most useful if it contains conversations where developers share their knowledge (typically in response to an information-seeking question). We discern such a conversation type by analyzing its form using the following features:

Primary Question: This feature captures the Q&A conversation style by identifying a primary, or leading question. Cong et al. [26] used 5W1H words and question mark to extract question-answer pairs from online forums, while others

Table 3. Features to Identify Post Hoc Quality Conversations

Feature Set	Quality Characteristics	Features	Value
Knowledge Seeking/ Sharing (KS)	PH2, PH3, NPH2, NPH3	1. Primary Question? 2. Knowledge-seeking Question? 3. Accepted Answers? 4. #Authors	Binary (1/0) Binary (1/0) Binary (1/0) Numeric count
Contextual (CX)	PH3, PH4, NPH2, NPH4	1. #API Mentions 2. #URLs 3. Contains Code? 4. Contains Code Description? 5. Amount of Code 6. Contains Error Message? 7. #Software-specific Terms	Numeric count Numeric count Binary (1/0) Binary (1/0) Numeric count Binary (1/0) Numeric count
Succinct (SC)	PH4, NPH2	1. #Utterances 2. #Sentences 3. #Words 4. Time Span 5. #Text Speaks 6. #Questions 7. Unique Info 8. DG: Avg Shortest Path 9. DG: Avg Graph Degree	Numeric count Numeric count Numeric count Numeric measure Numeric count Numeric count Numeric measure Numeric measure Numeric measure
Well-Written (WW)	NPH2	1. #Misspellings 2. #Incomplete Sentences 3. Automatic Readability Index 4. Coleman Liau Index 5. Flesch Reading Ease Score 6. Flesch Kincaid Grade Level 7. Gunning Fog Index 8. SMOG Grade	Numeric count Numeric count Numeric measure Numeric measure Numeric measure Numeric measure Numeric measure Numeric measure
Participant Experience (PE)	PH1, PH4, NPH1	1. Questioner: #Convs 2. Questioner: #Utterances 3. All Participants: #Convs 4. All Participants: #Utterances	Numeric count Numeric count Numeric count Numeric count

[43, 62] also used interrogative phrases (e.g., 'why', 'how','who', 'where', and 'what'). Similarly, we identify the primary question feature by the presence of a question mark and 5W1H words [26] in the first utterance.

Knowledge-seeking Question: Harper et al. [39] noted that informational questions, which are asked to seek information, frequently contain either "what", "where", or "how". We determine a primary question to be knowledge-seeking if it contains any of these three words.

Accepted Answers: Accepted answers suggest sharing of good quality information. In their predictions of question quality in Stack Overflow, Ponzanelli et al. [56] considered a question to be high quality if it has an accepted answer. Similarly, we hypothesize that conversations that have an accepted answer are *post hoc quality*. In our previous work, we created a list of words/phrases/emojis that indicate answer acceptance in a conversation [18]. We found those indicators to be too specific for this task, thus we developed a set of seed words that includes words that express gratitude (e.g., thanks, appreciate) and words indicating that the solution worked (e.g., works, helps). We use the word embedding model from the Python package spaCy [2], and calculate the similarity of each sentence in a conversation to each word in the seed word list. If a sentence has a similarity score over 0.5, and the sentence belongs to an utterance whose author matches the author who asked the primary question, we consider that conversation to contain an accepted answer. The similarity threshold is based on our experience with the development set.

#Authors: A higher number of authors indicates variety of shared knowledge, which could contribute to the richness of information. However, it is also possible that a too high number of authors could splinter the topic of the conversation, contributing to noise. #Authors is extracted from conversation meta-data.

Contextual: Q&A conversations containing contextual information (e.g., when I do ..., I get...) help developers to understand the relevance of the content to their issue and help mining-based software engineering tools to extract specific types of information. To determine whether a conversation contains sufficient *contextual* information, we compute:

#API Mentions: Based on our preliminary analysis (Section 3), we designed a regular expression for API mentions in each programming language (python, clojure, elm, racket) in our manual analysis dataset (Section 3: Dataset). We count the number of unique APIs mentioned in the natural language text in a conversation [67] (details included in our replication package). Specifically, we removed duplicates of the same API mention in a conversation and considered only unique or distinct APIs.

#URLs: Several researchers have used number of URLs or links to external resources to predict deleted and closed questions [27, 28, 79] and quality of questions [56] and answers on Stack Overflow [36, 67]. We count the number of URLs/links from the chat to Stack Overflow or tutorials using regular expressions.

Contains Code: In a qualitative analysis of questions on Stack Overflow, Asaduzzaman et al. [8] observed that unanswered questions often contain no example code. Without example code, it can be difficult for readers to identify the problem and offer a solution. Similarly, we hypothesize that conversations containing example code would be more *post hoc quality*. We determine if a conversation contains inline/multiline code examples by Slack's single quotes for enclosing inline code and triple quotes for multiline code, and links to code such as to Gist or Pastebin.

Contains Code Description: We compute the total number of sentences describing any code in a conversation. Specifically, we extract all the code identifiers by tokenizing the embedded code snippets of a conversation [19]. This feature is not specific to a given programming language. If a sentence contains any of the extracted identifiers, we consider it to be a description of code.

Amount of Code: Researchers have found that size of embedded code segments is an important feature in predicting deleted and closed questions on Stack Overflow [27, 79]. Hence, we count number of non-whitespace characters in all code snippets that occur in each conversation [36, 67].

Contains Error Messages: We detect stack trace or error/exception context in the primary question by "Error:", which we frequently observed in our manual analysis dataset (Section 3) to be present along with error information. This feature could be adapted to include other error strings.

#Software-specific Terms: We count the number of software-specific words or phrases [22, 42, 74] (e.g., deprecated, wrapper, debugger, etc) in a conversation using a list that combines morphological terms collected by Chen et al. [22] and software-related terms by Christensson [24].

Succinct: Conversations that are unclear or difficult to understand add little value to a knowledge source. We use structural features to consider a conversation to be *succinct*, as follows:

#Utterances: We count the number of utterances or messages in a conversation as an indicator of interactivity between users.

#Sentences and #Words: Conversations could be too short to provide useful information, or too long and provide noisy or redundant information. Researchers used word and sentence counts to detect low quality Stack Overflow posts [8, 36, 57]. We use the word and sentence tokenizer from NLTK [11] for these counts.

Time Span: A conversation over a long time could indicate less succinctness. We extract conversation time from metadata.

#Text Speaks: Ponzanelli et al. [56] observed that text speak in a Stack Overflow question, where a lengthy phrase is abbreviated or replaced by letters and numbers, (e.g.,'afaik', 'rotfl'), indicates low quality. We used a list of 50 most

common text speaks in chats [1, 57] to count the number of sentences in a conversation containing one or more text speak instances.

#Questions: Kitzie et al. [43] assumed that presence of multiple questions might confuse readers, so they used number of questions to predict question quality in Q&A forums. We first tokenized all sentences in a conversation, and then used two heuristics on each of those sentences to determine if it is a question. Specifically, we search for question marks at the end and 5W1H words [26] in the beginning of a sentence, to count questions in a conversation.

Unique Info: Kitzie et al. [44] proposed that the amount of novel information communicated within a question could help an answerer in interpreting the question with a higher level of specificity. We calculate this measure as the ratio of the number of distinct words over the total number of words in a conversation.

Disentanglement Graph: Average Shortest Path Length: The disentanglement graph consists of utterances as nodes and weighted undirected edges indicating strength of the relationship between two utterances. The average shortest path length indicates how connected any two pairs of utterances are in the conversation. Our hypothesis is that connected conversations (low value for average shortest path) are likely to represent a single cohesive topic.

Disentanglement Graph: Average Graph Degree: We also measure how connected, on average, each utterance is to other utterances in the disentanglement graph by degree. Our hypothesis is that graphs with higher degrees show that the utterances are more connected to each other, and thus likely to represent one technical topic and have less drift to additional off-topic discussions.

Well-written: Syntactically incorrect sentences accompanied by misspelled words make mining of essential information difficult. We use both form and readability as features to distinguish well-written chats:

#Misspellings: Researchers hypothesize that questions in Q&A forums that contain many misspelled words may be unclear [43, 62]. We used pyspellchecker to count misspelled words detected by computing the word's Levenshtein distance [53] from words in a corpus of English and commonly used terms in software engineering [22, 42, 74].

#Incomplete Sentences: We consider a sentence to be incomplete if it does not contain a subject or object. We use Python package spaCy to extract the subject and object of sentences [2].

Readability Scores: Researchers have used readability (e.g. Coleman, Flesch Kincaid) metrics to automatically detect low quality posts [56], predict unanswered questions [8] and estimate question quality on Q&A forums [43, 62]. We compute several readability measures including Automated Reading Index [68], Coleman Liau Index [25], Flesch Reading Ease Score [35], Flesch Kincaid Grade Level [35], Gunning Fog Index [37], and SMOG Grade [51] of a conversation as we hypothesize that conversations with high readability scores are more suitable for information extraction.

Participant Experience: Apart from content-related characteristics, the quality of a conversation could be impacted by the participant users' prior experience on Slack and the conversation's topic. Inexperienced users could potentially contribute low quality content [10, 28, 57]. Since Slack does not provide a built-in user reputation system or badge status for users, we measure experience of users participating in a conversation in two parts: (a) questioner experience, and (b) experience of all participants.

Questioner: The questioner's experience is estimated by counting the conversations and the individual utterances that one has participated in our whole dataset. Higher questioner #Conversations and #Utterances indicate that the primary question is asked by an experienced user on Slack. We have calculated the questioner's experience based on their interactions in our dataset since we focus on the participants' experience regarding the topic discussed in a conversation or channel (from which the dataset is taken) rather than general experience. Since our questioner's experience is only based on their interactions in the analyzed dataset, in some cases, it is possible to be biased towards novice developers who often ask questions in the same channel.

All Participants: Similar to the previous feature, we estimate the experience of all participants in a conversation by counting the conversations and the individual utterances that each participant has contributed to in our dataset. Higher all-participant #Conversations and #Utterances indicate that the given conversation contains information that is contributed by experienced Slack users.

4.2.2 Machine Learning-based Classifiers. With our features, we trained multiple classifiers using the Weka toolkit [38] (for Logistic Regression, Stochastic Gradient Boosted Trees and Random Forest) and Python scikit-learn package (for Sequential Neural Network). We explored other machine learning-based classifiers (e.g., Support Vector Machines); however, we do not discuss them, since they yielded significantly inferior results. Here, we provide overview and explanation of our classifier choices.

Logistic Regression (LR) is a discriminative classification model that predicts the class by calculating the probability for each class and choosing the class with highest probability. In our case, the class probability is the likelihood of a conversation being *non post hoc quality*. Logistic regression has been widely used for predicting duplicate and closed questions on Stack Overflow [4, 27], and classifying high quality questions on Stack Overflow and Yahoo! Answers [29, 36, 47, 62].

Stochastic Gradient Boosted Tree (SGBT) is an ensemble-based classification framework where a sequence of decision trees is constructed, and each tree minimizes the residual error of the preceding sequence of trees. Given a set of conversations with human-labeled informational judgments, SGBT automatically selects and uses combinations of features in a conversation, combining evidence from each *post hoc quality* characteristic. Ensemble classifiers such as SGBT have been observed to have high accuracy in predicting closed questions on Stack Overflow [27, 28] and identifying high quality content in social media [3].

Random Forest (RF) is an ensemble-based classifier that constructs a set of decision trees in randomly selected spaces of the feature space. The predictions of the individual decision trees are combined by applying bagging or bootstrap aggregating to generate the final classification. RF has been used for classifying high/low quality questions on Stack Overflow and Yahoo! Answers [29, 62].

Sequential Neural Network (SNN) is trained to perform binary classification by using the logarithmic loss function and Adam optimization algorithm for gradient descent. Our model has three hidden layers; each layer consists of 64 neurons (twice the number of our features) and uses relu activation function. We standardized the data using Python scikit-learn package. We use a batch size of 5 and train the model over 10 epochs. Our SNN takes 10 minutes to perform 10-fold classification of 2000 conversation instances on a system with 2.5 GHz Intel Core i5 processor and 8GB DDR3 RAM. All features are pre-computed before classification.

5 EVALUATION STUDY DESIGN

We designed our evaluation to analyze the automatic classifiers' effectiveness, features, and misclassifications.

5.1 Evaluation Metrics

We use measures that are widely used for evaluation in information retrieval and classification: precision, recall, F-measure, AUC and Matthews correlation coefficient. To measure the fraction of automatically identified conversations that are indeed *post hoc quality*, we use precision, the ratio of true positives (tp) over the sum of true and false positives

Table 4. Evaluation Dataset

Community	#Conv	#Utterances	#Users
pythondev#help	400	7746	304
clojurians#clojure	400	6521	387
elmlang#beginners	400	8897	310
elmlang#general	400	8793	318
racket#general	400	6020	81
total	**2000**	**37977**	**1400**

(fp). To see how often our approaches miss *post hoc quality* conversations, we use recall, the ratio of true positives over the sum of true positives and false negatives (fn). F-measure combines these measures by harmonic mean. To measure robustness, we use AUC, the area under the ROC (Receiver Operating Characteristics) curve, which represents the degree of separability between prediction classes. These metrics range between 0 and 1, where 1 represents complete agreement between prediction and gold set.

Lastly, because our data is not completely balanced, we compute MCC (Matthews correlation coefficient), which is a correlation coefficient between observed and predicted binary classifications that is well suited for unbalanced data; MCC lies between -1 and +1, where +1 represents a perfect prediction, 0 no better than random prediction and -1 total disagreement between prediction and observation.

5.2 Evaluation Dataset

5.2.1 Source and Size. We extract a subset of the developer conversations [2] of our previously released dataset [17], following the data selection procedure for our studies in section 3. Specifically, we first collected all the conversations from the dataset which occurred within a period of 489 days (June 2017- November 2018), which enabled us to collect sufficient data for analysis, and then we curated our evaluation dataset by randomly selecting a representative portion of the channel activity. In total, our evaluation dataset, as described in Table 4, consists of 2000 conversations (400 from each of five communities), 37,977 utterances, and 1,400 users.

5.2.2 Gold Set Creation. Our preliminary study helped us systematically define *post hoc quality* conversations, which we now leverage to create a large and representative annotated dataset for evaluation. We recruited two human judges with experience in programming (3+ years) and in using Slack, but no knowledge of our techniques or features. Our annotation instructions focused on labeling each conversation as either *post hoc quality* or *non post hoc quality*, based on the description and characteristics of *post hoc quality* and *non post hoc quality* conversations in section 3. *Post hoc quality* conversations were annotated as 1 and *non post hoc quality* conversations as 0. To avoid errors arising from the disentanglement to affect our automatic quality classification, our human judges corrected the errors before annotating any incorrectly disentangled conversations.

Frequencies of *post hoc quality* conversations across the channels are *pythondev#help*: 251/400, *clojurians#clojure*: 288/400, *elmlang#beginners*: 328/400, *elmlang#general*: 263/400, *racket#general*: 180/400. Racket has the lowest number of *post hoc quality* conversations because, several discussions were about specific projects and temporary design changes/bug fixes, or events and announcements that would not be useful for future readers. To promote future research in this area, we release our code and dataset along with the gold set annotation[3].

Both judges first annotated shared 200 conversations (40 from each Slack channel). This sample size is sufficient to compute the agreement measure with high confidence [14]. We computed Cohen's Kappa inter-rater agreement

[2] https://www.zenodo.org/record/3763432 - version 2
[3] https://bit.ly/3dkgA9g

between the two judges, who iteratively discussed and resolved conflicts, resulting in an agreement of 0.79, which is considered to be sufficient (> 0.6) [45]. Thus, the judges separately annotated the remaining conversations to reach 2000.

5.3 Procedures

We configured classifiers and ran them as follows.

Hyperparameter Tuning: We investigated hyperparameters to adjust each classifier - (RF #trees {100, 500, **1000**, 2000}, SGBT #boosting stages {10, **100**, 500}, and SNN #hidden layers {1, **3**, 5}, #neurons {32, **64**, 128}, and #epochs {**10**, 40, 100}); the bolded configurations produced the best classifications. We observed that the classification task was not very sensitive to parameter choices, as they had little discernible effect on the effectiveness metrics (in most cases $<= 0.01$). For all other classifier parameters, we used the reasonable defaults offered by popular libraries, *Weka*, *scikit-learn*, and *keras*.

Configurations: We investigated 24 classifier configurations. Each of the four machine learning-based techniques was configured with all features as well as singly with each of the five feature sets: Knowledge Seeking/Sharing (KS), Contextual (CX), Succinct (SC), Well-written (WW), Participant Experience (PE).

Class Imbalance Handling: Our dataset is imbalanced, with almost twice as many *post hoc quality* than *non post hoc quality* conversations. To address this imbalance in training our classifiers, we explored both over-sampling (SMOTE) and under-sampling techniques. We found that neither led to significant improvements in the results. Since, in most cases, we observed same or slightly inferior results (≤ 0.1), we opted against using over or under-sampling in our study.

Evaluation Process: Results from the classifiers were obtained using stratified 10-fold cross validation i.e., the conversation set was partitioned into ten equal-sized sub-samples with stratification, ensuring that the original distribution of conversation types (% *post hoc quality* conversations) is retained in each sub-sample.

5.4 Baselines

To the best of our knowledge, this is the first work to automatically identify *post hoc quality* and *non post hoc quality* developer conversations in chat forums. Thus, we developed two heuristic-based techniques as baselines to evaluate our classifiers' performance.

From manual analysis, we found that knowledge is shared in conversations by someone asking a question and people responding with information from their own knowledge, possibly with follow-up questions and answers. Thus, we look for discussions initiated by a question. The presence of a question is determined by a question mark (?) at the end of a sentence in the first utterance of a conversation.

As discussed in Section 3, a *post hoc quality* conversation is one that could help a person to gain useful software-related knowledge. For designing the baselines, we explore the first characteristic (PH1 and NPH1 defined in Section 3) i.e., determine if a discussion is related to software and programming topics. Specifically, we investigated two proxies for determining whether a conversation is software-related. The first baseline, *Q&A:SEterms* detects software-related conversations based on the existence of software-related words. We consider a conversation to be of post hoc quality if it contains at least one software-specific term. We use the list of software-related words described in section 4.2.1 to detect a software-specific term. The second baseline, *Q&A:Code* considers a conversation as software-related based on containing at least one code segment. Multi-line code snippets in Slack are encoded as markdown using triple quotes. Hence, for *Q&A:Code*, we use the presence of triple quotes to detect a code segment.

Table 5. Classification Results (Classifiers - LR: Logistic Regression, SGBT: Stochastic Gradient Boosted Trees, RF: Random Forest, SNN: Sequential Neural Net; Feature Sets - KS: Knowledge Sharing, CX: Contextual, SC: Succinct, WW: Well-written, PE: Participant Exp)

Classifier	Feature	Evaluation				
		Precision	Recall	F1	AUC	MCC
Baseline	**Q&A:SEterms**	0.78	0.67	0.72	0.66	0.30
	Q&A:Code	**0.91**	0.30	0.45	0.62	0.28
SGBT	**All**	0.78	0.78	0.78	0.83	0.50
	KS	0.77	0.77	0.76	0.76	0.46
	CX	0.77	0.77	0.77	0.82	0.48
	SC	0.78	0.77	0.75	0.78	0.47
	WW	0.75	0.75	0.74	0.77	0.43
	PE	0.74	0.75	0.74	0.78	0.41
LR	**All**	0.79	0.79	0.79	0.84	0.52
	KS	0.75	0.75	0.74	0.77	0.43
	CX	0.76	0.76	0.76	0.81	0.46
	SC	0.72	0.72	0.69	0.75	0.34
	WW	0.72	0.73	0.72	0.75	0.37
	PE	0.68	0.68	0.61	0.76	0.21
RF	**All**	0.81	0.81	0.81	**0.86**	**0.57**
	KS	0.77	0.77	0.75	0.76	0.45
	CX	0.76	0.76	0.76	0.80	0.46
	SC	0.75	0.75	0.74	0.77	0.42
	WW	0.73	0.74	0.73	0.77	0.40
	PE	0.74	0.75	0.74	0.77	0.41
SNN	**All**	0.82	0.90	**0.86**	**0.86**	0.55
	KS	0.77	**0.92**	0.84	0.77	0.46
	CX	0.80	0.87	0.84	0.82	0.50
	SC	0.77	0.89	0.83	0.78	0.44
	WW	0.77	0.86	0.82	0.78	0.42
	PE	0.76	0.86	0.80	0.78	0.37

5.5 Evaluation Results and Discussion

RQ1: How effective are machine learning-based techniques for automatic identification of post hoc quality developer chats? Table 5 presents precision, recall, F-measure, AUC and MCC for each configuration. We compare both of our baselines *Q&A:SEterms* and *Q&A:Code* with the machine learning-based techniques. When using *Q&A:SEterms*, we observe a reasonable F-measure of 0.72, but low MCC of 0.30. This discrepancy in the results is from *Q&A:SEterms* being reasonably good at recognizing *post hoc quality* conversations, but inadequate at effectively recognizing instances of the *non post hoc quality* (minority) class. In *Q&A:Code*, precision rises from *Q&A:SEterms*'s 0.78 to 0.91, but recall falls from 0.67 to 0.30, indicating that *Q&A:Code* is much more restrictive in labeling a conversation as *non post hoc quality*. The MCC score, 0.28, for *Q&A:Code* is a bit lower than *Q&A:SEterms*, 0.30. In Table 5, we provide the evaluation measures for both the baselines. Since, *Q&A:SEterms* performs better than *Q&A:Code* in terms of all measures (except precision), we only show the graphical representation of *Q&A:SEterms* in Figure 2. We have bolded *Q&A:SEterms* in Table 5, to emphasize that *Q&A:SEterms* perform better than *Q&A:Code*. Although the two baselines perform reasonably well with some metrics, they are poor particularly for MCC that adjusts for class imbalance. We compare the various machine learning-based classifiers as part of RQ2.

RQ2: Which classifiers and features result in more effective automatic identification?
Classifier Effectiveness. Figure 2 graphically depicts the overall effectiveness using *Q&A:SEterms* for the baseline and all features for the ML-based classifiers. From Table 5, and Figure 2 we observe that precision across all methods is

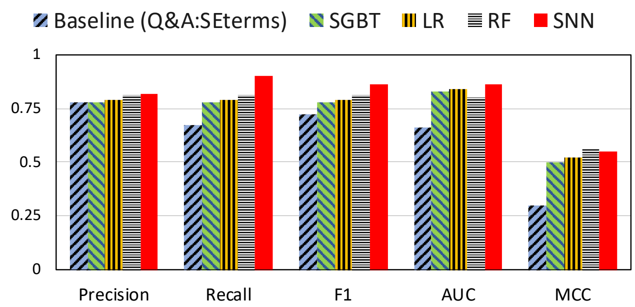


Fig. 2. Comparing Effectiveness (with all features)

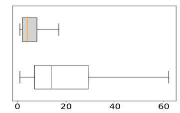
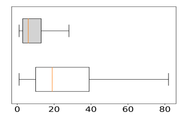
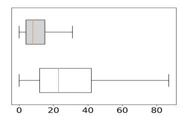
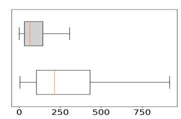
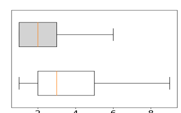
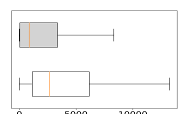
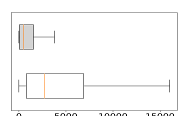
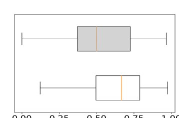
nearly the same (except *Q&A:Code* which has a precision of 0.91, but a low recall of 0.30); however, we see differences in the rest of the measures. Across most metrics, the best performance is achieved using Sequential Neural Network (SNN) with all features, achieving both F-measure and AUC = 0.86, and MCC = 0.55. However, when considering only MCC, Random Forest (RF) with all features performs slightly better (0.57). The remaining machine learning-based classifiers produce slightly worse results, with precision, recall and F-measure all ranging 0.78-0.79, AUC ranging 0.83-0.84, and MCC ranging 0.50-0.52 when using all features.

Overall, we observed the best performance when all features are used; however, knowledge-seeking/sharing (KS) and contextual (CX) features perform better than other sets when single feature sets are used. This suggests that conversation structure (e.g., Q&A -based structure, indication of answer acceptance) and context (e.g., code snippets and their descriptions) are strong indicators for distinguishing *post hoc quality* vs *non post hoc quality* conversations. It is interesting to note that although SGBT has overall worse performance (compared to LR and RF) when using all features, it produces better results when using individual feature sets. This indicates the strengths of different classifiers for handling larger vs. smaller feature sets.

Feature Importance. To understand the overlap between our 32 features, i.e., how many underlying dimensions of *post hoc quality* are expressed by our features collectively, we applied Principal Component Analysis (PCA) to the gold set. We specifically use PCA to extract the relevant information in our high-dimensional dataset, i.e., capturing the principal components that explain the spread or variance of features. To capture 90% of the variance in the dataset, PCA produces 15 different components, which shows relatively high diversity among our feature set. The most highly expressed component, accounting for 35% of the variance, broadly groups the features pertaining to the length of a conversation: #Utterances, #Sentences, #Words, #Software-specific Terms, and #Incomplete Sentences. The second component, accounting for 12% variance, broadly combines the readability metrics: Flesch Reading Ease Score, Flesch Kincaid Grade Level, Automatic Readability Index, and Gunning Fog Index. The remaining components produced by PCA consist of even smaller sets of the remaining features, which account for decreasing portions of the variance in the dataset, from 5% to 2%. We also separately applied PCA to the conversations from our manual analysis dataset in Section 3, observing highly similar components and feature distributions. Overall, the results of applying PCA show that our features are diverse, expressing separable notions of *post hoc quality*, with overlap between small groups of features.

We also determined the information gain [58] of each feature in our feature sets. Table 6 shows the top eight features across all feature sets, arranged in decreasing order of information gain. The values in the column *'Info Gain'* on Table

Table 6. Information Gain of Top 8 Features (Decreasing Order) and Data Distribution (White: Post Hoc Quality, Grey: Non Post Hoc Quality)

Feature	Info Gain	Data Distribution
#Utterances (SC)	0.204	
#Sentences (SC)	0.182	
Software-specific Terms (CX)	0.174	
#Words (SC)	0.171	
#Authors (KS)	0.158	
Time Span (SC)	0.156	
All Participants:#Convs (PE)	0.151	
DG: Avg Graph Degree (SC)	0.146	

6 indicate that the length (#Utterances, #Sentences, #Words, Time Span), coherence (DG: Avg Graph Degree), topic of conversation (#Software-specific Terms), and participant knowledge (#Authors, All Participants:#Convs) are the most informative features for our classification task. In the column *'Data Distribution'* of Table 6 we show box plot representations; the x-axis represents the range of values for each feature, and the boxes indicate the distribution of *post hoc quality* and *non post hoc quality* conversations represented in white and grey, respectively. For example, the median number of utterances for *post hoc quality* conversations is 14 and *non post hoc quality* conversations is 4, which indicates that too short conversations do not presumably provide useful information. Similar observations can be made for the other features related to the length of the conversations, such as, number of sentences, number of words, and time span. We observe that *post hoc quality* conversations have higher frequency of software specific terms, which

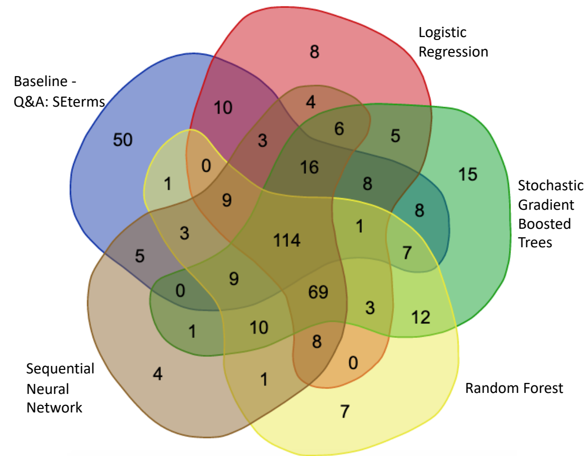


Fig. 3. False Positive Overlap Among Approaches

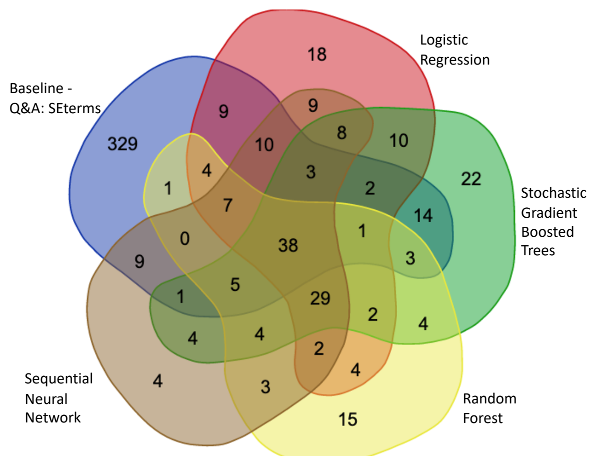


Fig. 4. False Negative Overlap Among Approaches

serve as indicators to identify software-related conversations. *Post hoc quality* conversations have higher number of authors and participant experience (with high variance), possibly contributing to the variety and richness of shared knowledge. The results also confirm our hypothesis that *post hoc quality* conversations are likely to represent one coherent topic and have less drift to off-topic discussions; thus have higher average graph degrees than *non post hoc quality* conversations.

RQ3: What types of conversations are difficult to automatically detect as post hoc quality using our techniques?

To perform classification error analysis, we qualitatively analyzed the False Positives (FP) and False Negatives (FN) across our baseline (*Q&A:SEterms*) and all ML-based classifiers (using all features). We chose *Q&A:SEterms* as baseline for this analysis since it performs better than *Q&A:Code* in all measures except precision. We found that a set of 114 conversations were marked as FP, and 38 conversations were marked as FN, by all the classifiers (i.e., intersection sets).

The analysis procedure consisted of the following steps: (1) First we collected the conversation instances that were marked FP by all classifiers, and instances that were marked FN by all classifiers. (2) Following an open coding procedure [63], two authors of this paper independently studied the conversation instances from step 1. We manually analyzed the conversations to identify the characteristics of conversations in each category (FP and FN), in terms of various representative feature values such as #Utterances, #Authors, Primary Question?, Accepted Answers?, Contains Code?. We also recorded additional comments and reflections from the manual analysis of the conversations in the form of words or short phrases, e.g. "Conversation contains general discussion about code editors for Python", "Conversation contains general discussion about Elm apps". These insights helped us investigate additional characteristics that our features failed to capture. (3) The two authors then met to discuss and group common observed characteristics of FP and FN conversations. The analysis was performed in an iterative approach composed of multiple sessions, which helped in generalizing the hypotheses and revising the characteristics. For example, the previous two mentioned observations were grouped into a category of "Topic of discussion not related to specific programming-related questions".

Figure 3 presents a Venn diagram of the False Positives (FP). We manually analyzed the 114 conversations marked FP by all classifiers, and observed that most of them lack specific programming-related questions. Instead, these conversations are software-related discussions where the participants discuss/ask for recommendations about code editors, testing practices, tutorials, etc. In addition, our classifiers struggled distinguishing conversations based on the quality of the answers provided. For example, some FP conversations were not completely answered or the proposed solution did not seem to work as indicated by follow-on discussion. A third group of FP conversations that we misclassified were either long and noisy (contain utterances that do not add value), or too specific (e.g. discussions about possible feature improvements to a language that are unlikely to be relevant to others post-conversation).

Figure 4 is a Venn diagram of False Negatives (FN) across all classifiers. We manually analyzed the 38 conversations marked FN by all classifiers, and observed that most are very short with an average of 3-4 utterances. These conversations are misclassified since they do not offer a lot of content for our features. Additionally, we are not able to correctly classify conversations that do not typically start with a specific question since most of our features are based on Q&A conversations.

5.6 Threats to Validity

Construct Validity: There might be some cases where the humans misclassified a *post hoc quality* conversation in the gold set. To limit this threat, we ensured the annotators had considerable experience in programming and using Slack, followed a consistent procedure piloted in advance, and we computed Cohen's Kappa inter-rater agreement between the two annotation sets for a shared sample of 200 conversations, observing a 0.79 agreement, which is more than 0.6 considered to be sufficient [14, 45]. Another potential threat is the description of *post hoc quality* conversation. Since Slack does not provide a built-in mechanism for evaluation of the quality of the conversations, we used the results from a post-conversation knowledge-seeking analysis (Section 3) to refine our understanding of *post hoc quality* conversations. The constructs to measure the phenomena under the study are the features for the machine learning-based approach, that were designed based on the characteristics of post hoc quality conversations and quality indicators in Stack Overflow. The characteristics of post hoc quality were based on the results of our preliminary manual analysis.

Internal Validity: Errors arising from the automatically disentangled conversations, particularly, some orphaned sequences of 1-2 messages, could pose a threat to internal validity resulting in misclassification. We mitigated this threat by humans without knowledge of our techniques manually refining the conversation disentanglement (Section 5.2.2). Since chat communications are informal in nature, it is possible that punctuations are omitted in the text [85].

To minimize this threat, we have used 5WIH words along with question mark to detect questions. Our heuristics for question identification could potentially be improved to handle more complex forms, such as indirect questions. Other potential threats could be related to errors in our scripts and evaluation bias. To overcome these threats, we used the conversations from our manual analysis dataset in Section 3 as the development set, to develop the scripts and classifier. We wrote separate test cases to test our scripts and performed code reviews.

External Validity: We selected the subjects of our study from Slack, which is one of the most popular software developer chat communities. Our study's results may not transfer to other chat platforms or developer communications. To mitigate this threat, we selected four active programming language communities (5 Slack channels) for our study. There is a broad set of topics related to a particular programming language in each channel. Since the quality characteristics (Section 3) and the features (Section 4.2.1) are not specific to the Slack chat platform, our automatic approach is likely to be applicable to all developer chat conversations, regardless of platform. It is also possible that our 2000 conversations are not representative of the full corpus of Slack conversations for a given community. The size of this dataset was chosen to give us a statistically representative sample (statistical significance with confidence of 95% ± 5%), feasible for our human judges to annotate. However, scaling to larger datasets might lead to different classification results.

6 RELATED WORK

Analyzing Software Developer Chats. Studies have focused on how chat communities are used by development teams, learning about developer behaviors, and examining information embedded in chats. Shihab et al. analyzed content, participants, contributions and communication styles of Internet Relay Chat (IRC) meeting logs [65, 66]. Yu et al. studied IRC and mailing lists to understand how real time and asynchronous communication methods could be used across global software development projects [82]. Elliott and Scacchi showed that open source communities use IRC channels, email discussions and community digests to mitigate and resolve conflicts [32]. Lin et al. showed that most developers use Slack for team-wide purposes such as facilitating communication and team collaboration through team management, file and code sharing [48]. Ehsan et al. conducted an empirical study to analyze the characteristics of the posted questions and the impact on the response behavior on Gitter developer chat platform [31]. Lebeuf et al. investigated how chatbots can reduce developers' collaborative friction points [46]. Paikari et al. compared chatbots for software development [54]. Panichella et al. investigated emerging developers' collaborations in software projects, and how those collaboration links complement each other by analyzing communication data from mailing lists, issue trackers, and IRC chat logs [55]. Our previous work showed that Q&A chats contain descriptions of code snippets and specific APIs, and identified challenges in mining developer chats [18]. Chowdhury and Hindle automatically filter off-topic IRC discussions [23]. Alkadhi et al. showed that machine learning can be leveraged to detect rationale in IRC messages [5–7]. Wood et al. created a supervised classifier to automatically detect speech acts in developer Q&A bug repair conversations [78]. Shi et al. detected feature-request dialogues from chat messages using deep Siamese network [64].

Outside the domain of software engineering, researchers have extensively studied chat communications to analyze discourse acts [70, 83], and informal writing styles [85]. More recently, researchers are focusing on the broad areas of developing conversational intelligence and understanding the social interactions embedded within chats [15, 16, 72].

Analyzing Quality of Q&A Forums. To our knowledge, there is a lack of research on assessing quality of information shared in developer chat communities. Other developer communications, such as Q&A forums, have explicit signals of quality. Specifically in Stack Overflow, users can vote on the posts they think are of good quality. In addition, Stack Overflow has a user reputation system built up mainly by answering questions on the site. A high reputation carries a significant amount of prestige within the forum community as well as externally. Additional prestige is earned via badges awarded when a contributor reaches specific point thresholds or when she becomes one of the top contributors to a specific topic. However, the level of distinction is achieved by a very few contributors ($< 1\%$ of active contributors have gold badge status) [12]. Beginning contributors are allowed only limited influence on the site, such as voting up or down for questions or answers. Experienced users with exceedingly high numbers of points receive additional moderation capabilities, with a few (~24) elected to become Stack Overflow moderators, who can perform additional tasks, such as closing or opening questions [28]. Low quality posts are identified through a review queue system managed by moderators. Based on several system criteria, Stack Overflow has 7 review queues: Late Answers, First Posts, Low Quality Posts, Close/Reopen Votes, Suggested Edits and Community Eval [57]. No such built-in mechanisms for indication of quality is present in chat platforms.

Beyond built-in mechanisms in Q&A forums, researchers have conducted studies of the characteristics associated with the quality of questions, answers, and code segments in posts on those forums. Sillito et al. found that the most helpful Stack Overflow answers contain concise examples with contextual explanations [67]. Yang et al. observed that good answers mostly contain multiple line code [80]. Duijn et al. determined that code-to-text ratio and code readability are most important in determining question quality [29]. Baltadzhieva and Grzegorz observed that questions in Q&A forums containing incorrect tags or that are too localized, subjective, or off topic are considered bad quality [10]. Correa and Sureka proposed a predictive model to detect the deletion of a Stack Overflow question at creation time [27, 28]. Ponzanelli et al. automatically identify and remove misclassified good quality Stack Overflow posts from the review queue [57]. Yao et al. predict high impact questions and useful answers just after they are posted by predicting voting scores [81]. Some researchers [4, 84] have also designed automatic techniques to help moderators detect duplicate questions on Stack Overflow.

Outside the software domain, researchers have focused on identifying high quality content in Yahoo Answers [3], designing techniques to automatically predict or detect question quality [43, 44, 47, 62], and best answer prediction on Community Question Answering (CQA) forums [34, 40, 52]. Harper et al. found that conversational questions have potentially lower archival value than informational questions [39]. Guy et al. replicated [39] on a larger dataset, and developed machine learning classifiers that use a large dataset of unlabeled data and achieve enhanced performance on automatically identifying informational vs. conversational questions on community question answering archives. Our techniques use Harper et al.'s question words for informational questions as a feature to automatically detect *post hoc quality* questions.

7 SUMMARY AND FUTURE WORK

This paper reports on the first work to automatically identify and extract *post hoc quality* information from developer chat communications. In this paper, we presented machine learning-based classifiers to classify software-related Slack conversations in terms of *post hoc quality*. Our evaluation shows that the machine-learning based approach could achieve a reasonable performance of 0.82 precision and a higher recall of 0.90. The qualitative analysis suggests that we can further improve the performance of our approach by refining the techniques of question identification (to handle more complex forms such as indirect questions) and for assessing the quality of the answers by understanding the conversation context, for example.

Automatically identifying *post hoc quality* conversations takes a first step in the research of quality assessment with developer chat communities. Understanding the quality of the information in those chats is essential for building software maintenance tools so that they contribute to efficient problem solving and enrich existing knowledge-bases and community knowledge. With the capability to automatically identify *post hoc quality* conversations from software-related chats, we provide a significant advancement opening up many opportunities to leverage the quality developer knowledge embedded in chats for software development tools and ultimately the software engineer. Our immediate future work focuses on improving the effectiveness by analyzing the quality of answers using text analysis clues. We will also expand to a larger and more diverse developer chat communication dataset, including conversations from other developer chat platforms. We envision improving developer communications by designing a chatbot to assign quality scores to previous conversations in the medium, and to help developers find high-quality information in public chat platforms. This would prevent duplication of questions asked on the medium, and help developers better manage their communications.

ACKNOWLEDGMENTS

We acknowledge the support of the National Science Foundation under grants 1812968 and 1813253.

REFERENCES

[1] 2020. smart-words.org. https://www.smart-words.org/abbreviations/text.html. Accessed: 2020-09-02.

[2] 2020. spaCy. https://spacy.io/. Accessed: 2020-09-02.

[3] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. 2008. Finding High-quality Content in Social Media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining* (Palo Alto, California, USA) *(WSDM '08)*. ACM, New York, NY, USA, 183–194. https://doi.org/10.1145/1341531.1341557

[4] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. 2016. Mining Duplicate Questions in Stack Overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 402–412. https://doi.org/10.1145/2901739.2901770

[5] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge. 2017. REACT: An Approach for Capturing Rationale in Chat Messages. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 175–180. https://doi.org/10.1109/ESEM.2017.26

[6] R. Alkadhi, T. Lata, E. Guzmany, and B. Bruegge. 2017. Rationale in Development Chat Messages: An Exploratory Study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 436–446. https://doi.org/10.1109/MSR.2017.43

[7] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge. 2018. How do developers discuss rationale?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Vol. 00. 357–369. https://doi.org/10.1109/SANER.2018.8330223

[8] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K. Roy, and Kevin A. Schneider. 2013. Answering Questions About Unanswered Questions of Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, Piscataway, NJ, USA, 97–100. http://dl.acm.org/citation.cfm?id=2487085.2487109

[9] Alberto Bacchelli, Luca Ponzanelli, and Michele Lanza. 2012. Harnessing Stack Overflow for the IDE. In *Proc. 3rd Int'l Wksp. on Recommendation Systems for Software Engineering*. 26–30.

[10] Antoaneta Baltadzhieva and Grzegorz Chrupala. 2015. Question Quality in Community Question Answering Forums: A Survey. *SIGKDD Explor. Newsl.* 17, 1 (Sept. 2015), 8–13. https://doi.org/10.1145/2830544.2830547

[11] Steven Bird. 2002. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics.*

[12] Amiangshu Bosu, Christopher S. Corley, Dustin Heaton, Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. 2013. Building Reputation in StackOverflow: An Empirical Investigation. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, Piscataway, NJ, USA, 89–92.

[13] Wesley Brants, Bonita Sharif, and Alexander Serebrenik. 2019. Assessing the Meaning of Emojis for Emotional Awareness - A Pilot Study. In *Companion Proceedings of The 2019 World Wide Web Conference* (San Francisco, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 419?423. https://doi.org/10.1145/3308560.3316550

[14] Mohamad Adam Bujang and Nurakmal Baharum. 2017. A Simplified Guide to Determination of Sample Size Requirements for Estimating the Value of Intraclass Correlation Coefficient: a Review. *Archives of Orofacial Science* 12, 1 (2017).

[15] Jonathan P. Chang, Caleb Chiam, Liye Fu, Andrew Z. Wang, Justine Zhang, and Cristian Danescu-Niculescu-Mizil. 2020. ConvoKit: A Toolkit for the Analysis of Conversations. arXiv:2005.04246 [cs.CL]

[16] Jonathan P. Chang and Cristian Danescu-Niculescu-Mizil. 2019. Trouble on the Horizon: Forecasting the Derailment of Online Conversations as they Develop. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 4743–4754. https://doi.org/10.18653/v1/D19-1481

[17] P. Chatterjee, K. Damevski, N.A. Kraft, and L. Pollock. 2020. Software-related Slack Chats with Disentangled Conversations. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR'20)*.

[18] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N.A. Kraft. 2019. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR'19)* (Montreal, Canada). https://doi.org/10.1109/MSR.2019.00075

[19] P. Chatterjee, B. Gause, H. Hedinger, and L. Pollock. 2017. Extracting Code Segments and Their Descriptions from Research Articles. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 91–101. https://doi.org/10.1109/MSR.2017.10

[20] C. Chen, S. Gao, and Z. Xing. 2016. 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*. 338–348. https://doi.org/10.1109/SANER.2016.21

[21] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised Software-specific Morphological Forms Inference from Informal Discussions. In *Proc. 39th Int'l Conf. on Software Engineering*. 450–461. https://doi.org/10.1109/ICSE.2017.48

[22] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised Software-specific Morphological Forms Inference from Informal Discussions. In *Proceedings of the 39th International Conference on Software Engineering* (Buenos Aires, Argentina) *(ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 450–461. https://doi.org/10.1109/ICSE.2017.48

[23] S. A. Chowdhury and A. Hindle. 2015. Mining StackOverflow to Filter Out Off-Topic IRC Discussion. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 422–425. https://doi.org/10.1109/MSR.2015.54

[24] Per Christensson. Accessed: 2020-09-02. TechTerms.com. https://techterms.com/category/software.

[25] M. Coleman and T. L. Liau. 1975. A computer readability formula designed for machine scoring. *Journal of Applied Psychology* (1975), 283–284.

[26] Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yueheng Sun. 2008. Finding Question-answer Pairs from Online Forums. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Singapore, Singapore) *(SIGIR '08)*. ACM, New York, NY, USA, 467–474. https://doi.org/10.1145/1390334.1390415

[27] Denzil Correa and Ashish Sureka. 2013. Fit or Unfit: Analysis and Prediction of 'Closed Questions' on Stack Overflow. In *Proceedings of the First ACM Conference on Online Social Networks* (Boston, Massachusetts, USA) *(COSN '13)*. ACM, New York, NY, USA, 201–212. https://doi.org/10.1145/2512938.2512954

[28] Denzil Correa and Ashish Sureka. 2014. Chaff from the Wheat: Characterization and Modeling of Deleted Questions on Stack Overflow. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14)*. ACM, New York, NY, USA, 631–642. https://doi.org/10.1145/2566486.2568036

[29] Maarten Duijn, Adam Kučera, and Alberto Bacchelli. 2015. Quality Questions Need Quality Code: Classifying Code Fragments on Stack Overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Florence, Italy) *(MSR '15)*. IEEE Press, Piscataway, NJ, USA, 410–413. http://dl.acm.org.udel.idm.oclc.org/citation.cfm?id=2820518.2820574

[30] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. 2017. Confusion Detection in Code Reviews. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 549–553.

[31] Osama. Ehsan, Safwat Hassan, Mariam E. Mezouar, and Ying Zou. 2020. An Empirical Study of Developer Discussions in the Gitter Platform. *Transactions on Software Engineering and Methodology (TOSEM)* (July 2020).

[32] Margaret S. Elliott and Walt Scacchi. 2003. Free Software Developers As an Occupational Community: Resolving Conflicts and Fostering Collaboration. In *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '03)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/958160.958164

[33] Micha Elsner and Eugene Charniak. 2008. You talking to me? A Corpus and Algorithm for Conversation Disentanglement. In *Proc. Association of Computational Linguistics: Human Language Technology*. 834–842.

[34] Pnina Fichman. 2011. A comparative assessment of answer quality on four question answering sites. *Journal of Information Science* 37, 5 (2011), 476–486. https://doi.org/10.1177/0165551511415584 arXiv:https://doi.org/10.1177/0165551511415584

[35] Rudolph Flesch. 1948. A new readability yardstick. *Journal of Applied Psychology* 32, 3 (June 1948), p221 – 233. http://libezproxy.open.ac.uk/login?url=http://search.ebscohost.com.libezproxy.open.ac.uk/login.aspx?direct=true&db=pdh&AN=apl-32-3-221&site=ehost-live&scope=site

[36] Neelamadhav Gantayat, Pankaj Dhoolia, Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2015. The Synergy Between Voting and Acceptance of Answers on Stackoverflow, or the Lack Thereof. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Florence, Italy) *(MSR '15)*. IEEE Press, Piscataway, NJ, USA, 406–409. http://dl.acm.org/citation.cfm?id=2820518.2820573

[37] R. Gunning. 1952. The Technique of Clear Writing. *McGraw-Hill* (1952).

[38] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278

[39] F. Maxwell Harper, Daniel Moy, and Joseph A. Konstan. 2009. Facts or Friends?: Distinguishing Informational and Conversational Questions in Social Q&A Sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. ACM, New York, NY, USA, 759–768. https://doi.org/10.1145/1518701.1518819

[40] F. Maxwell Harper, Daphne Raban, Sheizaf Rafaeli, and Joseph A. Konstan. 2008. Predictors of Answer Quality in Online Q&Amp;A Sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. ACM, New York, NY, USA, 865–874. https://doi.org/10.1145/1357054.1357191

[41] Liangjie Hong and Brian D. Davison. 2009. A Classification-based Approach to Question Answering in Discussion Boards. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Boston, MA, USA) *(SIGIR '09)*. ACM, New York, NY, USA, 171–178. https://doi.org/10.1145/1571941.1571973

[42] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. 2013. Automatically Mining Software-Based, Semantically-Similar Words from Comment-Code Mappings. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 377–386. https://doi.org/10.1109/MSR.2013.6624052

[43] Vanessa Kitzie, Erik Choi, and Chirag Shah. 2013. Analyzing Question Quality Through Intersubjectivity: World Views and Objective Assessments of Questions on Social Question-answering. In *Proceedings of the 76th ASIS&T Annual Meeting: Beyond the Cloud: Rethinking Information Boundaries* (Montreal, Quebec, Canada) *(ASIST '13)*. American Society for Information Science, Silver Springs, MD, USA, Article 5, 10 pages. http://dl.acm.org/citation.cfm?id=2655780.2655785

[44] Vanessa Kitzie, Erik Choi, and Chirag Shah. 2013. From Bad to Good: An Investigation of Question Quality and Transformation. In *Proceedings of the 76th ASIS&T Annual Meeting: Beyond the Cloud: Rethinking Information Boundaries* (Montreal, Quebec, Canada) *(ASIST '13)*. American Society for Information Science, Silver Springs, MD, USA, Article 107, 4 pages. http://dl.acm.org/citation.cfm?id=2655780.2655887

[45] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[46] Carlene Lebeuf, Margaret-Anne D. Storey, and Alexey Zagalsky. 2017. How Software Developers Mitigate Collaboration Friction with Chatbots. *CoRR* abs/1702.07011 (2017). http://arxiv.org/abs/1702.07011

[47] Baichuan Li, Tan Jin, Michael R. Lyu, Irwin King, and Barley Mak. 2012. Analyzing and Predicting Question Quality in Community Question Answering Services. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12 Companion)*. ACM, New York, NY, USA, 775–782. https://doi.org/10.1145/2187980.2188200

[48] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion* (San Francisco, California, USA) *(CSCW '16 Companion)*. ACM, New York, NY, USA, 333–336. https://doi.org/10.1145/2818052.2869117

[49] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. 2019. Pattern-Based Mining of Opinions in Q&A Websites. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 548?559. https://doi.org/10.1109/ICSE.2019.00066

[50] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, Prague, Czech Republic, 285–294. https://doi.org/10.18653/v1/W15-4640

[51] H. G. McLaughlin. 1969. SMOG grading - a new readability formula. *Journal of Reading* (May 1969), 639–646.

[52] Piero Molino, Luca Maria Aiello, and Pasquale Lops. 2016. Social Question Answering: Textual, User, and Network Features for Best Answer Prediction. *ACM Trans. Inf. Syst.* 35, 1, Article 4 (Sept. 2016), 40 pages. https://doi.org/10.1145/2948063

[53] Gonzalo Navarro. 2001. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.* 33, 1 (March 2001), 31–88. https://doi.org/10.1145/375360.375365

[54] Elahe Paikari and André van der Hoek. 2018. A Framework for Understanding Chatbots and Their Future. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering* (Gothenburg, Sweden) *(CHASE '18)*. ACM, New York, NY, USA, 13–16. https://doi.org/10.1145/3195836.3195859

[55] S. Panichella, G. Bavota, M. D. Penta, G. Canfora, and G. Antoniol. 2014. How Developers' Collaborations Identified from Different Sources Tell Us about Code Changes. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 251–260. https://doi.org/10.1109/ICSME.2014.47

[56] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter. In *Proc. 11th Working Conf. on Mining Software Repositories*. 102–111. https://doi.org/10.1145/2597073.2597077

[57] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. 2014. Improving Low Quality Stack Overflow Post Detection. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 541–544. https://doi.org/10.1109/ICSME.2014.90

[58] J. R. Quinlan. 1986. Induction of Decision Trees. *MACH. LEARN* 1 (1986), 81–106.

[59] M.M. Rahman, C.K. Roy, and D. Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *Proc. IEEE 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering*. 349–359. https://doi.org/10.1109/SANER.2016.80

[60] M.M. Rahman, S. Yeasmin, and C.K. Roy. 2014. 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*. 194–203. https://doi.org/10.1109/CSMR-WCRE.2014.6747170

[61] M. M. Rahman, C. K. Roy, and I. Keivanloo. 2015. Recommending Insightful Comments for Source Code using Crowdsourced Knowledge. In *Proc. IEEE 15th Int'l Working Conf. on Source Code Analysis and Manipulation*. 81–90. https://doi.org/10.1109/SCAM.2015.7335404

[62] Manasa Rath, Long T. Le, and Chirag Shah. 2017. Discerning the Quality of Questions in Educational Q&Ausing Textual Features. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval* (Oslo, Norway) *(CHIIR '17)*. ACM, New York, NY, USA, 329–332. https://doi.org/10.1145/3020165.3022145

[63] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples* (1st ed.). Wiley Publishing.

[64] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Li Shoubin, and Qing Wang. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. ACM, New York, NY,

USA.

[65] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09)*. IEEE Computer Society, Washington, DC, USA, 107–110. https://doi.org/10.1109/MSR.2009.5069488

[66] E. Shihab, Z. M. Jiang, and A. E. Hassan. 2009. Studying the Use of Developer IRC Meetings in Open Source Projects. In *2009 IEEE International Conference on Software Maintenance*. 147–156. https://doi.org/10.1109/ICSM.2009.5306333

[67] Jonathan Sillito, Frank Maurer, Seyed Mehdi Nasehi, and Chris Burns. 2012. What Makes a Good Code Example?: A Study of Programming Q&A in StackOverflow. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM) (ICSM '12)*. IEEE Computer Society, Washington, DC, USA, 25–34. https://doi.org/10.1109/ICSM.2012.6405249

[68] E. A. Smith and R. J. Senter. 1967. Automated readability index. *AMRL TR* (May 1967), 1–14.

[69] The Statistics Portal Statista. 2020. https://www.statista.com/statistics/652779/worldwide-slack-users-total-vs-paid/.

[70] Matthew Stone, Una Stojnic, and Ernest Lepore. 2013. Situated Utterances and Discourse Relations. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013) – Short Papers*. Association for Computational Linguistics, Potsdam, Germany, 390–396. https://www.aclweb.org/anthology/W13-0214

[71] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (R) Evolution of Social Media in Software Engineering. In *Proceedings of the on Future of Software Engineering* (Hyderabad, India) *(FOSE 2014)*. ACM, New York, NY, USA, 100–116. https://doi.org/10.1145/2593882.2593887

[72] Naama Tepper, Anat Hashavit, Maya Barnea, Inbal Ronen, and Lior Leiba. 2018. Collabot: Personalized Group Chat Summarization. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (Marina Del Rey, CA, USA) *(WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 771?774. https://doi.org/10.1145/3159652.3160588

[73] Y. Tian, D. Lo, and J. Lawall. 2014. Automated construction of a software-specific word similarity database. In *Proc. IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering*. 44–53. https://doi.org/10.1109/CSMR-WCRE.2014.6747213

[74] Y. Tian, D. Lo, and J. Lawall. 2014. Automated Construction of a Software-Specific Word Similarity Database. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. 44–53. https://doi.org/10.1109/CSMR-WCRE.2014.6747213

[75] David C Uthus and David W Aha. 2013. Multiparticipant Chat Analysis: A Survey. *Artificial Intelligence* 199 (2013), 106–121.

[76] W. Wang and M.W. Godfrey. 2013. Detecting API usage obstacles: A study of iOS and Android developer questions. In *Proc. 10th Working Conf. on Mining Software Repositories*. 61–64. https://doi.org/10.1109/MSR.2013.6624006

[77] Edmund Wong, Jinqiu Yang, and Lin Tan. 2013. AutoComment: Mining Question and Answer Sites for Automatic Comment Generation. In *Proc. 28th IEEE/ACM Int'l Conf. on Automated Software Engineering*. 562–567. https://doi.org/10.1109/ASE.2013.6693113

[78] Andrew Wood, Paige Rodeghero, Ameer Armaly, and Collin McMillan. 2018. Detecting Speech Act Types in Developer Question/Answer Conversations during Bug Repair. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 491?502. https://doi.org/10.1145/3236024.3236031

[79] X. Xia, D. Lo, D. Correa, A. Sureka, and E. Shihab. 2016. It Takes Two to Tango: Deleted Stack Overflow Question Prediction with Text and Meta Features. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 73–82. https://doi.org/10.1109/COMPSAC.2016.145

[80] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From Query to Usable Code: An Analysis of Stack Overflow Code Snippets. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 391–402. https://doi.org/10.1145/2901739.2901767

[81] Yuan Yao, Hanghang Tong, Tao Xie, Leman Akoglu, Feng Xu, and Jian Lu. 2015. Detecting High-quality Posts in Community Question Answering Sites. *Inf. Sci.* 302, C (May 2015), 70–82. https://doi.org/10.1016/j.ins.2014.12.038

[82] Liguo Yu, Srini Ramaswamy, Alok Mishra, and Deepti Mishra. 2011. *Communications in Global Software Development: An Empirical Study Using GTK+ OSS Repository*. Springer Berlin Heidelberg, Berlin, Heidelberg, 218–227. https://doi.org/10.1007/978-3-642-25126-9_32

[83] Amy Zhang, Bryan Culbertson, and Praveen Paritosh. 2017. Characterizing Online Discussion Using Coarse Discourse Sequences.

[84] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. 2015. "Multi-Factor Duplicate Question Detection in Stack Overflow". *Journal of Computer Science and Technology* 30, 5 (01 Sep 2015), 981–997. https://doi.org/10.1007/s11390-015-1576-4

[85] L. Zhou and Dongsong Zhang. 2005. A heuristic approach to establishing punctuation convention in instant messaging. *IEEE Transactions on Professional Communication* 48, 4 (2005), 391–400.