MINING INFORMATION FROM DEVELOPER CHATS TOWARDS BUILDING SOFTWARE MAINTENANCE TOOLS

by

Preetha Chatterjee

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer and Information Sciences

Spring 2021

MINING INFORMATION FROM DEVELOPER CHATS TOWARDS BUILDING SOFTWARE MAINTENANCE TOOLS

by

Preetha Chatterjee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ______________________________

Lori Pollock, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ______________________________

Kostadin Damevski, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ______________________________

Nicholas A. Kraft, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: ______________________________

Sunita Chandrasekaran, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I would first like to express my deepest gratitude to my advisor, Prof. Lori Pollock, for providing me the opportunity to work and learn under her. I thank her for providing insightful feedback, sharing her valuable expertise, and for guiding and mentoring me throughout the course of my graduate studies at University of Delaware. You are everything one could look for in an advisor. You made my PhD an interesting and memorable experience. I will always be grateful to you for your support and kindness. Thank you for being such a great role model.

I thank Dr. Kostadin Damevski, Dr. Sunita Chandrasekaran, and Dr. Nicholas Kraft for being on my thesis committee. Special thanks to Dr. Damevski for his constant guidance throughout my time here at Delaware through online sessions. Our conversations always inspired me to work harder. I learnt a lot from him.

I would also like to thank the undergraduate researchers at University of Delaware with whom I had the pleasure to work with: Minji Kong, Brian Phillips, Humpher Owusu, Benjamin Gause, Hunter Hedinger, and Qilin Ma.

Across the sea, I thank my parents and sister for their unequivocal support and blessings. I thank my husband, Sandipan, for his constant support and understanding, especially when times were hard. You kept me going on and this work would not have been possible without your counsel. This thesis is dedicated to you.

Finally, I would like to acknowledge the financial support by National Science Foundation (NSF) and University of Delaware.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Software developers are increasingly having conversations about software development via online chat services. Many of those chat communications contain valuable information, such as description of code snippets and APIs, opinions on good programming practices, and causes of common errors/exceptions. Researchers have demonstrated that various software engineering tasks (e.g., recommend mentors in software projects, aid API learning) can be supported by mining similar information from other developer communications such as email, bug reports and Q&A forums. However, limited work has focused on investigating the availability and mining of information from developer chats.

To successfully mine developer chat communications, one has to address several challenges unique to chats. The nature of chat community content is transient. Developer chats are typically informal conversations, with rapid exchanges of messages between two or more developers, where several clarifying questions and answers are often communicated in short bursts. Chats thus contain shorter, quicker responses, often interleaved with non-information-providing messages. As a result, it is difficult to find relevant information in a large chat history, and important advice is lost over time. My thesis is that software developers communications through online chat forums are a valuable resource to mine and valuable knowledge can be automatically mined from this resource towards improving and building new tools to support software engineers.

The focus of this dissertation is *Mining Information from Developer Chats Towards Building Software Maintenance Tools*: (1) As a first step towards mining, we investigated the availability of information in chats through an empirical study. We also analyzed characteristics of chat conversations that might inhibit accurate automated analysis. (2) Next, we extended an existing algorithm to automatically extract (or disentangle) conversations for analysis by researchers or automatic mining tools. (3) Assessing the quality of information is essential for extracting useful information for building effective software maintenance tools. Hence, we designed an automatic technique to identify post hoc quality conversations, i.e. conversations containing useful information for mining or reading after the conversation has ended. (4) Finally, we studied the use of online chat platforms as a resource towards collecting developer opinions that could potentially help in building opinion Q&A systems, as a specialized instance of virtual assistants and chatbots for software engineers. We developed techniques for automatic identification of opinion-asking questions and extraction of participants' answers from public online developer chats.

This dissertation takes a significant step to positively impact new research directions on mining developer chats, and enables advances in areas including: information retrieval tasks from unstructured communications, enriching existing knowledge bases and community knowledge, efficient information gathering to improve code efficiency and increase developer productivity, building/enhancing recommendation systems and virtual assistants for software engineering.

Chapter 1

INTRODUCTION

Researchers studying the impacts of social media on software engineering have observed how increasingly social the modern software development communities have become, by contributing to crowd-sourced content and using new social tools, including GitHub, Stack Overflow and Slack [162]. More than ever, software developers are having conversations about software development via online chat services. In particular, developers are turning to public chat communities hosted on services such as Slack, IRC, HipChat, Gitter, Microsoft Teams, and Freenode to discuss specific programming languages or technologies. Developers use these communities to ask and answer specific development questions, with the aim of improving their own skills and helping others. The emerging trend of increased participation in developer chat communications motivates our research to extract useful information communicated in developers' chat communication channels towards building software maintenance tools. My thesis is that software developers communications through online chat forums are a valuable resource to mine and valuable knowledge can be automatically mined from this resource towards improving and building new tools to support software engineers.

Our exploratory study [31] shows that chat communications contain valuable information, such as descriptions of code snippets and specific APIs, good programming practices, and causes of common errors/exceptions. Researchers have demonstrated that various software engineering tasks can be supported by mining similar information from emails, bug reports, tutorials, and Q&A forums. For example, information mined from emails and bug reports is used to recommend mentors in software projects [27]. Further, the natural language text in tutorials is analyzed to aid API learning [87, 127]. Over the years, researchers have also mined the knowledge embedded in Q&A

forums, such as Stack Overflow, for supporting IDE recommendation [9, 13, 48, 53, 130, 138], learning and recommendation of APIs [38, 137, 183], automatic generation of comments for source code [134, 186], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40, 169]. Thus, availability of all these types of information in software-related chats shows promise for mining that information in building and improving software maintenance tools.

Furthermore, while Q&A forums such as Stack Overflow explicitly forbid posting of questions that ask for opinions or recommendations, we observed that developers regularly use chat forums to share opinions on best practices, APIs, and tools [31]. Recent research has shown that mined opinions are valuable to software developers [105]. 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 [34, 58, 107]. Additionally, developer chats have already served as a mining source for identifying feature requests [151] and extracting developer rationale [6].

To successfully mine developer chat communications, one has to address several challenges unique to chats:

- There is **no predefined delineation of conversation in chat communications**; each conversation could span from two messages to hundreds. In addition, conversations are often interleaved, which motivates the need for creating and/or adapting chat disentanglement techniques [61, 178] to identify individual conversations from chat history.

- **Public chat conversations 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. Relative to other developer communications such as Stack Overflow, chat platforms do not contain built-in quality indicators (e.g., accepted answers, vote counts). 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) [18, 55, 155, 194], to our knowledge, there are no known techniques to automatically assess the quality of the content in developer chat conversations.

- **Chat conversations contain shorter, quicker responses, often interleaved with non-information-providing messages.** A question asked at the start of a conversation may be followed by a series of clarification or follow-up questions and their answers, before the answers to the original question are given. Moreover, along with the answers, conversations sometimes contain noisy and unrelated information. To extract information from chats, and use it elsewhere out of the original context, it is important to analyze the context of the information being provided and remove irrelevant content.

The focus of this dissertation is *Mining Information from Developer Chats Towards Building Software Maintenance Tools*: (1) As a first step towards mining, we investigated the availability of information in chats through an exploratory study. We also analyzed characteristics of chat conversations that might inhibit accurate automated analysis. (2) Next, we extend one existing algorithm to automatically extract (or disentangle) conversations for analysis by researchers or automatic mining tools. (3) Assessing the quality of information is essential for extracting useful information for building effective software maintenance tools. Hence, we designed a technique 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. (4) Finally, we studied the use of online chat platforms as a resource towards collecting developer opinions that could potentially help in building opinion Q&A systems, as a specialized instance of virtual assistants and chatbots for software engineers.

We developed techniques for automatic identification of opinion-asking questions and extraction of participants' answers from public online developer chats.

This dissertation takes a significant step to positively impact existing research that has focused on mining information from written developer communications, and demonstrating the usefulness of this information for software tasks and tools. Both developers and mining tools using information from developer chat communications will benefit from this research. Specifically, this dissertation makes the following contributions in developing analyses for automatically identifying and extracting information in developers' chat communications:

1. An exploratory study to investigate the potential of developer chats as a mining source for software maintenance and evolution tools. Our results indicate the prevalence of useful information in software-related chats, including API mentions and code snippets with descriptions, and several hurdles that need to be overcome to automate mining that information [31].

2. A supervised machine learning technique customized to automatically disentangle conversations in software developer chats. The model with our enhancements produced a micro-averaged F-measure of 0.80 [34].

3. A public knowledge archive of conversations in developer chat communications, suitable for research beyond this project. Specifically, we present a dataset of software-related Q&A chat conversations, curated for two years from three open Slack communities (python, clojure, elm). Our dataset consists of 38,955 conversations, 437,893 utterances, contributed by 12,171 users. The data is openly available to be downloaded for further reuse by the community [34].

4. Identification of properties of post hoc quality information or quality metrics for developer chat communications [29].

5. An approach based on text processing and machine learning to automatically identify quality of developer chats for post hoc use. Evaluation shows that our

approach could achieve a reasonable performance of 0.82 precision and a higher recall of 0.90 [29].

6. A linguistic pattern-based technique to automatically identify opinion-asking questions in chat conversations. Evaluation shows that this approach achieves a precision and recall of 0.87 and 0.49 recall respectively, and significantly outperforms existing sentiment analysis tools and a pattern-based technique that was designed for emails [30].

7. A deep learning-based model to automatically extract participants' answers to opinion-asking questions in public Q&A chats. This answer extraction model could potentially be leveraged to extract answers to other types of questions in developer chats. Our deep learning approach customized to the software domain achieves a precision and recall of 0.77 and 0.59 respectively, and outperforms heuristics-based, machine-learning-based and deep learning for answer extraction in community question answering [30].

The remainder of this dissertation is organized as follows. Chapter 2 provides background information and related work on analyzing developer communications. Chapter 3 describes an exploratory study that investigates the potential usefulness and challenges of mining developer Q&A conversations for supporting software maintenance and evolution tools. Chapter 4 presents a modified chat disentanglement technique, and a dataset of software-related Q&A chat conversations curated for two years from three open Slack communities. Chapter 5 describes a preliminary analysis that indicates potential characteristics of post hoc quality, followed by a machine learning-based approach for automatically identifying conversations of post hoc quality. Chapter 6 provides a technique for automatic identification of opinion-asking questions and extraction of participants' answers from public online developer chats. Finally, in Chapter 7, we present a summary of our contributions and discuss potential future work.

Chapter 2
BACKGROUND AND STATE OF THE ART

2.1 Software Developer Communications as a Source of Knowledge

As software development teams are more globally distributed and the open source community has grown, developers rely increasingly on written documents for help that they might have previously obtained through in-person conversations. Table 2.1 lists different types of software artifacts that developers can learn from, with example sources of these artifacts. Developer communications (e.g., developer blog posts, bug reports, API documentation, mailing lists, code reviews, and question and answer forums) offer the unique opportunity to learn from what the developers are saying or asking about the code snippets they are discussing. E-books, online course materials, videos, and programming manuals offer learning opportunities, often explaining concepts using code examples. Similarly, technical presentations and research papers often use code examples to demonstrate challenges addressed by their work. Developers can also learn from code in benchmark suites, standards documents, and code snippets from repositories such as GitHub Gists and Pastebin.

While individual developers can learn from the aforementioned sources, the publicly available, large corpora of software-related documents also provide the opportunity to automatically learn common API behaviors, properties, and vulnerabilities, as well as to complement similar information learned by mining software repositories. Researchers have performed empirical analysis of developer communications to understand specific software enclaves or novel techniques used by developers in the field, for instance, examining modern code reviews using tools like Gerrit [19], or the information encoded in tutorials for mobile development [171]. The captured insights can

Table 2.1: Types of Software Artifacts

Document Type	**Example**
Blog Posts	MSDN, Personal blogs
Benchmarks	OpenMP NAS
Bug Reports	GitHub Issues, Bugzilla
Code reviews	GitHub Pull Requests,
Course materials	cs.*.edu
Documentation	readthedocs.org
E-Books	WikiBooks
Mailing lists	lkml.org
Papers	ACM Digital Library, IEEE Xplore
Presentations	SlideShare
Public Chat	Gitter, Slack
Q&A sites	Stack Overflow, MSDN
Code snippets	GitHub Gists
Standards docs	ISO C++ or Java Language Specification
Programming manuals	OpenGL

be integrated into the development of new models and analyses to enhance software engineering techniques and tools.

Researchers have also developed analyses to mine various information from the natural language text surrounding code snippets in emails [12, 123], bug reports [123], Q&A forums [35, 175, 179, 185], and tutorials [128]. For example, Panichella et al. [123] developed a feature-based approach to automatically extract method descriptions from bug tracking systems and mailing lists. Vassallo et al. built on their previous work [123] to design a tool called CODES that extracts candidate method documentation from Stack Overflow discussions and creates Javadoc descriptions [179]. Wong et al. [185] mine comments from Stack Overflow by extracting text preceding code snippets and using the code-description mappings in the posts to automatically generate descriptive comments for similar code snippets matched in open-source projects.

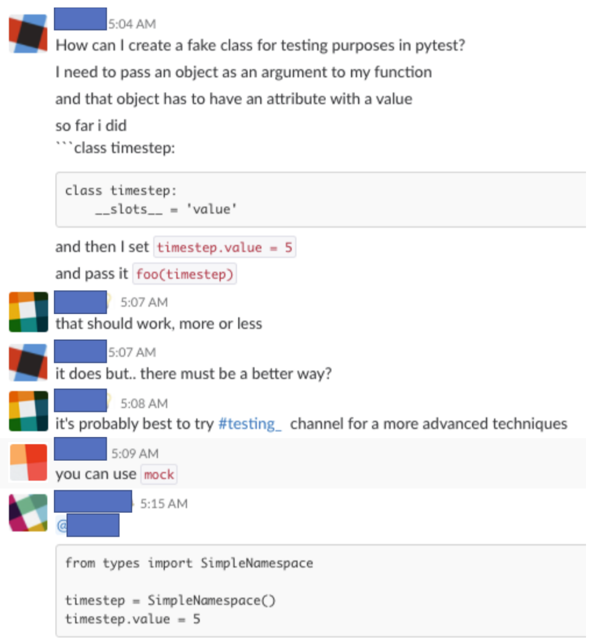
A key mechanism for providing the mined information for recommendations is through an Integrated Development Environments (IDE). Common recommendations include programming errors and exceptions [9, 48, 138], linking relevant discussions to any source code based on context [13,130] or through a query in the IDE [53,73]. Treude and Robillard proposed an approach for automatically augmenting API documentation with informative sentences from Stack Overflow [174]. Rahman et al. [137] proposed a technique that takes a natural language query as input and uses keyword-API associations mined from Stack Overflow to produce API recommendations as output. Other works for enriching API documentations include augmenting code examples [90, 166], identifying iOS and Android API classes for which developers regularly face usage obstacles [183], and integrating crowdsourced frequently asked questions (FAQs) from the web into API documents [39]. Collectively, these efforts demonstrate the potential for extracting and using information embedded in natural language text of software-related documents for software engineering tools.

2.2 Software Developer Chats as a Source of Knowledge

This section describes (1) how developers use public chat communities to ask and answer software development questions, (2) prior research in analyzing and mining software-related chats, and (3) potential of public chats to serve as a valuable data source to improve software maintenance tools.

2.2.1 Use of Chats in Developer Sharing and Learning

The most popular chat communities used by software developers include Slack, IRC, Gitter, HipChat, Microsoft Teams, and Flowdock. Slack, with over 10 million daily active users [161], is easily accessible to users as a mobile application (Windows, iOS, and Android) as well as a web-based and OS-based (Windows, Linux, and Mac) application. Public chats in Slack are comprised of multiple communities focused on particular topics such as a technology (e.g., Python or Ruby-on-Rails), with specific channels within a given community assigned to general discussion or to particular subtopics [163]. Within each channel, users participate in chat conversations, or chats, by posting messages. Across all messaging options, users can send text, emojis, and/or multimedia (image and video) messages. Chats in some channels follow a Q&A


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

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

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

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

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

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

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

[redacted] 5:15 AM
@[redacted]

```
from types import SimpleNamespace

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

Figure 2.1: Sample of a conversation in Slack (Python-dev community)

format, with information seekers posting questions and others providing answers, possibly including code snippets or stack traces, as shown in Figure 2.1. Slack provides easy integration to frequently used developer tools (e.g., Github, Bitbucket, JIRA, and Jenkins) through a set of conversation-based bots and apps [172]. These bots and apps have been widely adopted by many developers for different software engineering tasks such as maintaining code quality, testing, conducting development operations, supporting customers, and creating documentation [100].

In spite of the easy availability of bots and apps, when developers run into problems they can't solve on their own, they face several hurdles [66]. (1) Developers have trouble finding answers because it is difficult to find specific conversations in a large chat history. Also, they might not be aware of the suitable person to ask the question directly. (2) Inexperienced developers repeatedly ask the same questions. As a result, developers with specialized knowledge get distracted, thereby causing loss of productivity. (3) Unable to retrieve the necessary knowledge required to solve their problem,
```

developers spend considerable time reading extensive wikis and pages of documentation. (4) In the absence of a proper information extraction mechanism, newcomers struggle during on-boarding to get up to speed with their colleagues. Identifying and extracting useful knowledge from developer chats would help in reducing the above mentioned hurdles, and change how knowledge is shared and used among development teams. It would also help to increase developer productivity and improve quality of software products.

2.2.2 Analysis of Software Developer Chats

Recent studies have focused on learning about how chat communities are used by development teams and the usefulness of the conversations for learning about developer behaviors. Shihab et al. [152, 153] analyzed developer Internet Relay Chat (IRC) meeting logs to analyze the content, participants, their contribution and styles of communications. Yu et al. [197] conducted an empirical study to investigate the use of synchronous (IRC) and asynchronous (mailing list) communication mechanisms in global software development projects. Elliott and Scacchi [59] showed that open source communities use IRC channels, email discussions and community digests to mitigate and resolve conflicts. Lin et al. [103] conducted an exploratory study to learn how Slack impacts development team dynamics. Their results show that most developers use Slack for team-wide purposes including communication and collaboration with other team members. Ehsan et al. [58] conducted an empirical study to analyze the characteristics of the posted questions and the impact on the response behavior on Gitter developer chat platform. Stray et al. [164] investigate how distributed global development teams use Slack. Panichella et al. [124] investigate collaboration links identified through data from three different kinds of communication channels: mailing lists, issue trackers, and IRC chat logs.

Additionally, developer chats have served as a mining source to extract specific types of information related to software engineering tasks [33]. Chowdhury and Hindle [43] proposed an approach to automatically filter out off-topic IRC discussions by exploiting Stack Overflow programming discussions and YouTube video comments. Alkadhi et al. [6–8] conducted exploratory studies to examine the frequency and completeness of available rationale in chat messages, contribution of rationale by developers, and the potential of automatic techniques for rationale extraction. Their results show that machine learning algorithms can be leveraged to detect rationale in IRC messages with 0.76 precision and 0.79 recall. Shi et al. [151] detected feature-request dialogues from chat messages using deep Siamese network, to facilitate the requirements gathering process during software development. The automation in these analyses was used to learn about developer behavior, and the potential of machine learning techniques to extract specific types of information from developer chat communications. To the best of our knowledge, no research has focused on assessing the quality of information in chats to build effective data-driven software maintenance tools.

Recognizing the increasing capabilities of virtual assistants that use conversational artificial intelligence (AI) (e.g., chatbots, voice assistants), some researchers in software engineering are working towards the development of virtual assistants to help programmers. Lebeuf et al. [101] investigated how chatbots can help reduce the friction points that software developers face when working collaboratively. Paikari et al. [122] characterized and compared chatbots related to software development in six dimensions (type, direction, guidance, predictability, interaction style, and communication channel). Romero et al. [146] developed a chatbot that detects a troubleshooting question asked on Gitter and provides possible answers retrieved from querying similar Stack Overflow posts. Researchers have also conducted studies to gain insights into the design of a programmer conversational agent [98], proposed techniques to automatically detect speech acts in conversations about bug repair to aid the assistant in mimicking different conversation types [188], and designed virtual assistants for API usage [56].

Shortcomings and Remaining Issues. In general, despite the increasing popularity and research of developer chat communications in the software engineering community, there is not a clear understanding about developer chat communities from the perspective of understanding what kind of information can be mined from them, how much of that information is available, and how difficult that information is to extract.

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 [24]. There is no formal mechanism for voting or accepting an answer in chat platforms. 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) [18, 55, 155, 194], to our knowledge, there are no known techniques to automatically assess the quality of the content in developer chat conversations.

2.2.3 Opportunity: Chats as a Valuable Data Source To Improve Tools

In Chapter 3, we discuss several new opportunities in mining chat conversations [31]. We found that Q&A chats in Slack provide the same information as can be found in Q&A posts on Stack Overflow. Over the years, researchers have mined the knowledge embedded in Q&A forums, such as Stack Overflow, for supporting IDE recommendation [13, 130, 138], learning and recommendation of APIs [38, 137, 183], automatic generation of comments for source code [134, 186], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40, 169]. Presence of similar information in Slack Q&A chats suggests that it can serve as a resource for several mining-based software engineering tools. Additionally, we found that, developers use chat platforms to share opinions on best practices, APIs, or tools. Q&A forums such as Stack Overflow explicitly forbid posting of questions that ask for opinions or recommendations. However, recent research has shown that mined opinions are valuable to software developers [105].

In Chapter 4, we release a dataset of software-related chat conversations, that could be used to pursue several potential research directions. For example, we noticed that, along with few links to Stack Overflow and GitHub Gists, there were sporadic links to other sites in our dataset. We believe that embedded links on chats are used in many different contexts, and as such can be mined to provide more context to other data sources (tutorials, Q&A forums), and thus improve or augment developer learning resources. Chat communications could also be studied to identify 'hot' topics of discussion in a programming community [149], and understand common challenges and misconceptions among developers [16]. The results of these studies would provide guidance to future research in developing software support and maintenance tools.

In Chapter 5, 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, 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.

In Chapter 6, we leverage the availability of opinion-asking questions in developer chat platforms to explore the feasibility of building opinion-providing virtual assistants for software engineers. One type of virtual assistant that can benefit from opinions are Conversational Search Assistants (CSAs) [52]. CSAs support information seekers who struggle forming good queries for exploratory search, e.g., seeking opinions/recommendations on API, tools, or resources, by eliciting the actual need from the user through conversation. Studies indicate developers conducting web searches or querying Q&A sites for relevant questions often find it difficult to formulate good queries [117, 140]. Wizard of Oz studies have explicitly shown the need for opinions within CSAs [181]. A key result of my dissertation is the availability of opinions on chat platforms, which would enable the creation of a sizable opinion Q&A corpus that could be used by CSAs. The opinion Q&A corpus generated from chats by our technique can be used in a few different ways to build a CSA: 1) matching queries/questions asked to the CSA with questions from the corpus and retrieving the answers; 2) summarizing related groups of opinion Q&A to generate (e.g., using a GAN) an aggregate response for a specific software engineering topic. Beyond reducing developers' effort of manual searches on the web and facilitating information gathering through CSAs, mining of opinions could help in increasing developer productivity, improving code efficiency, and building better recommendation systems.

Chapter 3

INVESTIGATING AVAILABILITY OF INFORMATION IN CHATS THROUGH EMPIRICAL STUDY

In this chapter, we describe an exploratory study into the potential usefulness and challenges of mining developer Q&A conversations for supporting software maintenance and evolution tools. We designed the study to investigate the availability of information that has been successfully mined from other developer communications, particularly Stack Overflow. We also analyze characteristics of chat conversations that might inhibit accurate automated analysis.

3.1 Problem and Motivation

Researchers have demonstrated that various software engineering tasks can be supported by mining information from emails, bug reports, tutorials, and Q&A forums. For example, information mined from emails and bug reports is used to re-document source code [123] or to recommend mentors in software projects [27]. Further, the natural language text in tutorials is analyzed to aid API learning [87, 127]. Over the years, researchers have also mined the knowledge embedded in Q&A forums, such as Stack Overflow, for supporting IDE recommendation [9,13,48,53,130,138], learning and recommendation of APIs [38, 137, 183], automatic generation of comments for source code [134, 186], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40, 169]. These successes suggest that other kinds of developer communications may also provide information for mining-based software engineering tools.

Software developers are increasingly having conversations about software development via online chat services. In particular, developers are turning to public chat communities hosted on services such as Slack, IRC, Hipchat, Gitter, Microsoft Teams, and Freenode to discuss specific programming languages or technologies. Developers use these communities to ask and answer specific development questions, with the aim of improving their own skills and helping others. Over ten million active users participate daily on Slack, which is currently the most popular platform for these public chat communities and hosts many active public channels focused on software development technologies [161].

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

Most studies of developer chat communities have focused on learning about how they are used by development teams or for analyzing developer behaviors/interactions [7, 44, 100, 104, 152, 197]. Panichella et al. [124] investigated how developer collaboration links differ across three various kinds of communication channels, including mailing lists, issue trackers, and IRC chat logs. However, limited work has focused on mining and extracting specific types of information from developer chats. For instance, Alkadhi et al. [7,8] examined the frequency and completeness of available rationale in HipChat and IRC messages, and the potential of automatic techniques for rationale extraction. To the best of our knowledge, no previous work has investigated

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

chat communities from the perspective of understanding what kind of information can be mined from them, how much of that information is available, and how difficult that information is to extract.

3.2 Exploratory Study

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

3.2.1 Research Questions

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

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

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

3.2.2 Data Set Creation

3.2.2.1 Community Selection and Extraction

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

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

Because programmatic access to the data in Slack communities is controlled by the administrators of the Slack team, we contacted several public Slack teams and asked for an API token that would allow us to read and store their data. Public Slack teams typically use Slack's free tier, which only stores the most recent 10,000 messages. Thus, for each Slack community that we selected for our study, we downloaded all of the discussion data from each channel every day for 489 days (June 2017- November 2018). Since Stack Overflow regularly publishes all of its data in XML format via the Stack Exchange Data Exchange, we extracted posts for this study from the 27-August-2017 release of the Stack Exchange Data Dump [63].

3.2.2.2 Documents, Posts, Conversations, and Disentanglement

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

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

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

The disentanglement problem has been studied before in the context of IRC and similar chat platforms [178]. We leveraged the technique proposed by Elsner and Charniak [61] that learns a supervised model based on a set of features between pairs of chat messages that occur within a window of time of each other. The features include the elapsed time between the message pair, whether the speaker in the two messages is the same, occurrence of similar words, use of cue words (e.g., hello, hi, yes, no), and the use of technical jargon. More details on our disentanglement technique can be found in Chapter 4.

3.2.2.3 Curating a Slack-Stack Overflow Comparison Dataset

To enable comparison study between related communities on Stack Overflow and Slack, we first selected Slack programming communities and then identified related Stack Overflow communities by extracting all of the posts tagged with the language name from the Stack Exchange Data Dump. Table 3.1 shows the programming language communities (channels on Slack and tags on Stack Overflow) used as subjects

Table 3.1: Data sets and sizes
SO: Stack Overflow

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

of study. A key concern in comparing Slack and Stack Overflow was that the Slack conversations may not contain the same distribution of subtopics within a particular technology as Stack Overflow (e.g., if most conversations on pythondev#help were on Django but not a significant proportion of ‘python‘ tagged Stack Overflow questions). To address this concern, we curated our comparison dataset by organizing the extracted Slack conversations and Stack Overflow posts by similar subtopics, using the following workflow:

1. Remove URLs and code snippets from the Slack and Stack Overflow datasets. Filter out terms that appear in more than 80% of documents or in less than 15 documents.

2. Train a Latent Dirichlet Allocation (LDA) topic model using the larger Stack Overflow data. By manually examining topics for interpretability and coherence, we chose to extract 20 LDA topics in each community.

3. Remove background topics, if they exist, which are very strongly expressed in the corpus (strongest topic for $> 50\%$ of documents) and do not express technical terms.

4. Infer topic distributions for each document (i.e., post in Stack Overflow and conversation in Slack).

5. Select documents that strongly express the same LDA topic in Stack Overflow and Slack.

Throughout this chapter, we refer to these comparison datasets as $StackOverflow_{auto}$ and $Slack_{auto}$. As Table 3.1 shows, $Slack_{auto}$ consists of 23,893 conversations, while $StackOverflow_{auto}$ contains 825,294 posts.

3.2.2.4 Datasets for Manual Analysis

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

Two of the coauthors computed the measures in section 3.2.3 on the $Slack_{manual}$ dataset using the following process. We wanted to ensure that the inter-rater agreement is sufficient to allow two researchers to compute measures separately without duplication, to create a larger manual set with confidence. First, both authors computed all the measures on a shared set of 60 Slack conversations on $Slack_{manual}$. Using this initial shared set, we computed the Cohen's Kappa inter-rater agreement metric between the two authors, observing an agreement of more than 0.6 for each measure, which is considered to be sufficient [99]. Choosing a shared sample of 60 conversations also ensured the sample produced high confidence in the computed value of Cohen's Kappa based on the number of categories in our measures [25]. Second, the authors divided and separately computed the measures in the remaining manual samples to reach a total of 400 conversations in $Slack_{manual}$ and 400 questions in $StackOverflow_{manual}$.

Table 3.1 includes the number of conversations from each Slack channel and number of posts from each Stack Overflow community.

3.2.3 Methodology

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

We also conducted a qualitative analysis using Zagalsky et al.'s knowledge types, which they developed to compare Stack Overflow to the R-help mailing list [198]. Namely, we computed the occurrence of questions, answers, updates, flags, and comments on our channels to determine whether Slack Q&A conversations are similar in structure to Stack Overflow Q&A. In the context of Q&A chat conversations, 'question' represents the primary question which initiates the main topic of discussion; 'answer' provides a solution to the primary question; 'update' requests a modification to a question or answer; 'comment' provides clarification to a specific part of the question or answer; and 'flag' requests moderator attention (e.g., off-topic or repeated questions, spam).

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

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

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

Table 3.2: Measures for RQ1. The measures labeled *auto* are computed by applying scripts to the $Slack_{auto}$ and $StackOverflow_{auto}$ datasets, whereas the measures labeled *manual* are computed manually using the $Slack_{manual}$ and $StackOverflow_{manual}$ datasets.

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

To answer this question, we focused on information that has been commonly mined in other software artifacts. Specifically, we analyzed code snippets, links to code snippets, API mentions, and bad code snippets. To provide context, we also collected data on document length in Slack and Stack Overflow. Table 3.2 shows the measures computed.

The first step in analysis is identifying the particular elements in Slack and Stack Overflow documents, (e.g., code snippet, link, API mention). Since the text of each Slack document is stored in the Markdown text format, we used regular expressions to extract the code snippets and the URLs of links to code snippets from each conversation. Since the text of each Stack Overflow post is stored as HTML, we used the HTML parser, Beautiful Soup [144], to extract the code and link elements from each post. We wrote Python scripts to analyze conversations and posts for all the automatically collected information shown in Table 3.2. Next, we describe how and why we computed each measure for RQ1.

First, we compute the document lengths for context in understanding the prevalence of the measures described below. *Document length* is defined as the number of sentences in a document (i.e., a Slack conversation or a Stack Overflow post). We computed this measure on the natural language text in each document using the sentence tokenizer from NLTK [21].

High occurrence of code snippets within a forum indicates more opportunity to support various software engineering tasks such as IDE recommendation, code clone detection, and bug fixing. *Code snippet count* is computed as the number of code snippets per document, for which we counted all inline and multiline code snippets in a document. The answers that solve the original question on Stack Overflow mostly contain multiple line snippets, so the measure can be an indicator of snippet quality [194]. *Code snippet length* is the number of non-whitespace characters in each code snippet. For multiline snippets, we also counted the number of non-blank lines in each snippet.

URLs that link different software artifacts are useful to establish traceability between documents, which has been shown to be useful for tasks such as automatic comment generation. Specifically, by establishing mappings between code and associated descriptive text, it is possible to automatically generate descriptive comments for similar code segments in open-source projects [187]. We counted all URLs to Gist and Stack Overflow in a document. For instance, for *Gist links*, we counted all links to http://gist.github.com. For *Stack Overflow links*, we counted all links to either stackexchange.com or stackoverflow.com. The $StackOverflow_{auto}$ dataset also includes relative links in this measure.

High frequency of API mentions in code and text show potential to build tools such as API linking and recommendation and augmenting API documentation [37,135, 173]. *API mentions* in code snippets and text are the numbers of APIs mentioned in the code snippets and in the natural language text, respectively, for each document.

Bad code snippets could be mined to build tools for debugging and bug fixing, testing and maintenance, e.g., in designing test cases for mutation testing [69]. Percentage of *bad code snippets* is defined to be the number of "bad" code snippets divided by the total number of (inline and multiline) code segments in a document. Bad code snippets are defined as code segments that are described by negative words or phrases such as "ugly," "error", "does not work", or "horrible." These descriptors indicate that one or more of the forum or chat participants find that the code snippet does not implement the desired functionality, is inefficient, or has poor readability or syntax errors. We manually searched for the presence of such negative indicators in the natural language text to identify and count bad code snippets in the documents. We did not evaluate the code snippet itself for badness, as we were interested in bad code snippets based on participant interpretation, not our annotators' interpretations.

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

To answer this question, we focused on measures that could provide some insights into the form of Slack Q&A conversations (participant count, questions with no answer, answer count) and measures that could indicate challenges in automation (how participants indicate accepted answers, questions with no accepted answer, natural language text context pertaining to code snippets, incomplete sentences, noise within a document, and knowledge construction process) that suggest a need to filter.

Table 3.3: Measures for RQ2.

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

All measures shown in table 3.3 were computed manually on the $Slack_{manual}$ dataset, except *Participant count*, which could be computed with high accuracy automatically with scripts. Manual analysis was performed by two of the authors. Since the research question investigates challenges in mining information in developer chat communications to support software engineering tools, we only computed the measures on Slack.

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

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

Higher answer count indicates variety to the pool of solutions, which gives the developers an option to select the answer most specific to the problem in context. However, identifying multiple answers is non-trivial as it requires identifying the responsive sentences and the question to which those sentences are responding. For computing *Answer count*, we manually determined which sentence mapped to a given question and counted the number of solutions proposed in response to the initial question in the conversation that explained the problem. Specifically, we calculated the number of Zagalsky [198] "Answer" type of artifact in a conversation, and computed the minimum, median, and maximum number of answers per conversation in $Slack_{manual}$.

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

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

Embedded code snippets in Slack conversations are surrounded by natural language (NL) text that describes various code metadata such as the functionality being implemented, data structures, code complexity and effectiveness, errors and exceptions. *NL text context per code snippet* is defined as the number of sentences used to describe a particular code snippet. Extracting such sentences or phrases that provide additional information about a code segment embedded in a forum is valuable to software engineering tasks such as automatic comment generation and augmentation of documentation [134,173,187]. Identifying such sentences is not straightforward due to the informal and intertwined nature of utterances in conversations. Thus, we manually analyzed all the sentences in the disentangled conversations to count the number of NL sentences related to a particular code snippet. We report the minimum, median, and maximum counts of sentences describing code per code segment in conversations of $Slack_{manual}$.

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

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

We also examined the approach to constructing knowledge as the percentage of conversations belonging to each type of knowledge construction in $Slack_{manual}$ dataset, per Zagalsky's definitions - participatory versus crowd knowledge construction [198]. In participatory knowledge construction, answers are contributed by multiple users in the same thread. Participants enrich each others' solutions by discussing several aspects such as the pros and cons of each answer, different viewpoints, additional information, and examples. In crowd knowledge construction, participants add solutions individually, without discussing other solutions, thereby creating a resource of independent solutions. The *knowledge construction process* is computed as the percentage of conversations belonging to each category of knowledge construction in $Slack_{manual}$ dataset.

3.2.4 Threats to Validity

Construct validity: As with any study involving manual human analysis, there might be some cases where the humans may have incorrectly analyzed the documents. To limit this threat, we ensured that the authors who analyzed the manual data set had considerable experience in programming and qualitative analysis, and followed a consistent coding procedure that was piloted in advance. We computed the Cohen's Kappa inter-rater agreement metric between the two sets of results, observing an agreement of more than 0.6 for each measure, which is considered to be sufficient [99].

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

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

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

It is also possible that our smaller datasets for manual analysis are not representative of the full corpus of Slack conversations or Stack Overflow posts for a given community. The size of these datasets was chosen to give us a statistically representative sample, feasible for our annotators to analyze. However, scaling to larger datasets for the manual analysis might lead to different results.

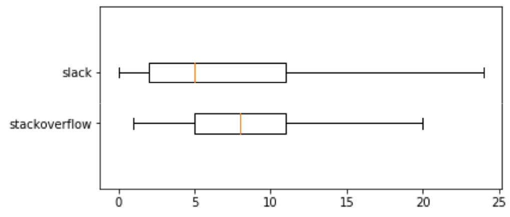
3.2.5 Findings

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

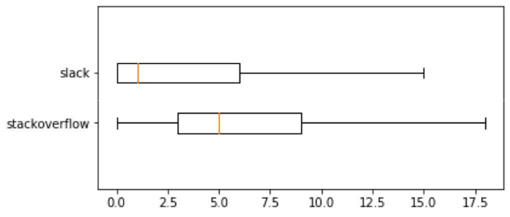
RQ1: How prevalent is the information that has been successfully mined from Stack Overflow Q&A forums to support software engineering tools in developer Q&A chats such as Slack?

For RQ1, we display the results primarily as box plots in Figure 3.1. The box plots for *Gist links*, *Stack Overflow links*, and *Bad code snippets* would be difficult to read because the median of the metric was 0. Therefore, we present the results for *links* as an aggregate in Figure 3.1d. For *Bad code snippets*, despite the medians being 0, the mean for Stack Overflow (21.2) was more than double the mean for Slack (9.4).

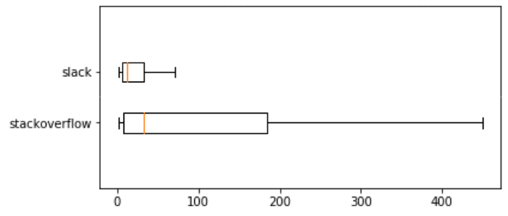
Table 3.4 presents the results of Mann Whitney U tests comparing our measures using the Slack and Stack Overflow datasets. All tests are two-tailed, checking the null



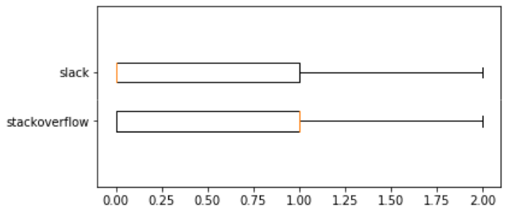
(a) Document length.



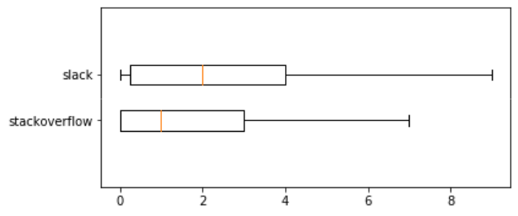
(b) Snippet count.



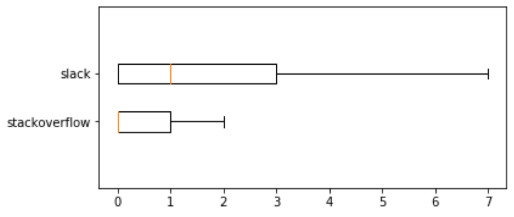
(c) Snippet length.



(d) URL count.



(e) API mentions in snippets.



(f) API mentions in text.

Figure 3.1: Box plots of RQ1 measurements by community.

hypothesis that *the metric values for Slack and Stack Overflow do not differ significantly.* The 'U' and 'p-value' columns list the U statistic and the corrected p-value, respectively. For all measures, we can reject the null hypothesis (at $p < 0.05$) of no difference between the two sources.

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

Table 3.4: Statistical Results for RQ1 (Slack vs. Stack Overflow).

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

However, most of this information, with the exception of API mentions, is available in larger quantities on Stack Overflow. For instance, Figure 3.1b indicates that Stack Overflow posts can have a much larger number of code snippets than Slack conversations. This suggests that code snippets could be mined from chat communities, but more conversations than posts would need to be analyzed to extract similar numbers of code snippets from Slack. We also observed this as we were curating our datasets for this study.

As shown in Figure 3.1c, the median code snippet lengths from both sources are similar, although the variation of snippet length is much larger for Stack Overflow code snippets than those included in Slack conversations. Thus, if one wants to mine sources where longer snippets might be available, this data suggests to mine Stack Overflow rather than Slack.

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

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

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

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

Table 3.5 and Table 3.6 present the results for RQ2. Table 3.5 shows all the natural language clues (words, phrases, and emojis) used as accepted answer indicators over all analyzed Slack communities. The most prevalent indicator is "Thanks/thank you", followed by phrases acknowledging the participant's help such as "okay", "got it", and other positive sentiment indicators such as "this worked", "cool", and "great". Accepted answers were also commonly indicated using emojis as listed in the table.

Table 3.6 presents the results for the remaining measures. Results represented as percentages are reported directly, while other results, computed as simple counts, are reported as minimum < median < maximum. For example, participant frequency in $Slack_{topic}$ ranged from a single participant in a conversation to 34 participants, with a median of 2 participants. A single participant conversation was one with no

Table 3.5: Indicators of an accepted answer in $Slack_{manual}$ dataset.

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

Emojis: 🙂; 👍; 💡; 🎉

answers. Based on the results reported in Table 3.6, approximately 16% of questions go unanswered, while the maximum number of answers was five for a question. Of the questions with answers, approximately 52% of questions have no accepted answer.

Code snippets have from 0 to 13 sentences that contained some kind of descriptive information about the code snippet. The results also indicate that the number of incomplete sentences describing code is low, 13%, and similarly the noise in a conversation can be as high as 11%. We also observed that 61.5% conversations followed an approach of crowd knowledge construction, while the rest 38.5% were participatory [198], where multiple developers collaborate to settle on an answer to a challenging question. To gain insight into the semantic information provided in Slack conversations, we analyzed the kinds of information provided in the conversations. Using the labels defined in one of our previous studies [36], we categorized the information as providing context in terms of design, efficiency, erroneous nature of the code, explanatory description of the functionality of the code, or a detail about the structure of the code. Figure 3.2 presents the prevalence of each type of information in our $Slack_{manual}$ dataset.

The largest proportion of Slack Q&A conversations discuss software design. We observed that the most prevalent types of information on Slack is "Design". This aligns

Table 3.6: Findings for RQ2. Cell data is of the form $Min < Median < Max$ or a percentage.

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

with the fact that the main purpose of developer Q&A chats is to ask and answer questions about alternatives for a particular task, specific to using a particular language or technology. Often the focal point of conversations are APIs, mentions to which we observed occur often on Slack Q&A channels, where a developer is asking experts on the channel for suggestions on API or proper idioms for API usage. Conversations that were "Explanatory" often involved a developer explaining, in long form, the lineage of a particular decision in the language or technology, or contrasting the capabilities of different languages, while "Structure" conversations often focused around a specific data structure (e.g., representing a dictionary in Elm) or control structure (e.g., how code can be split in multiple files in Clojure).

Accepted answers are available in chat conversations, but require more effort to discern. The existence of an accepted answer, selected by the question asker using a subsequent response, present the clearest indication of a Slack conversation's quality. There is a significant proportion of accepted answers available in Slack, based on Table 3.6. The median number of answers is only one, which suggests that there are expert people on the channels, such that only one answer is given and it is accepted. One might think

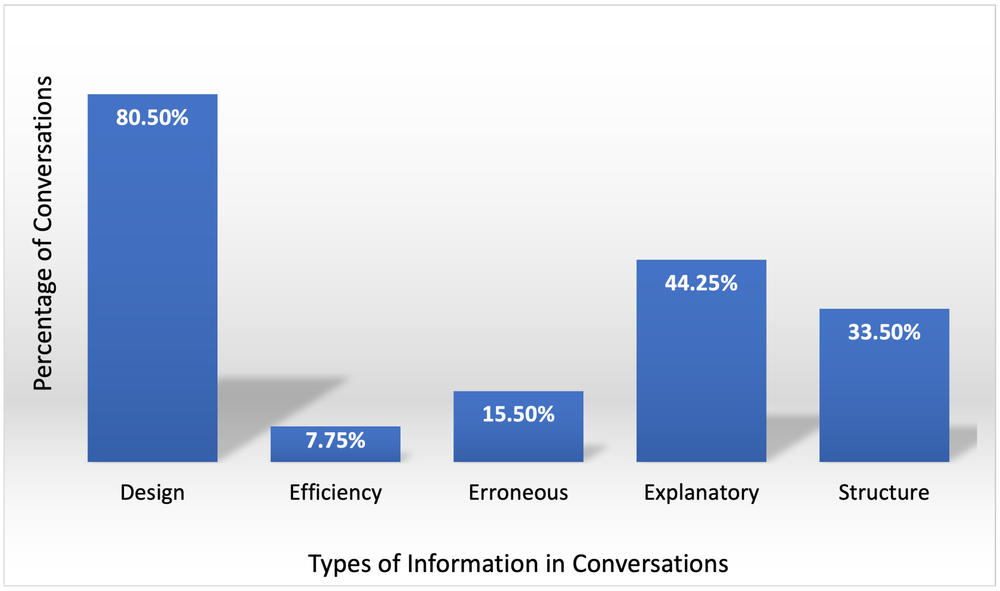


Figure 3.2: Percentage conversations of each type in $Slack_{manual}$

it is then easy to automatically identify the answer to a given question, especially with the use of emojis and the word/phrase indicators of accepted answers. However, an automatic mining tool needs to automatically identify the sentence in a conversation that is an answer to a question and which question it is answering. This implies that NLP techniques and sentiment analysis will most likely be needed to automatically identify and match answers with questions. Due to the speed of Slack conversations, we observed cases where the question asker accepts an answer, only to come back a few minutes later to declare that the answer did not help. Therefore, mining techniques have to be able to track the conversation to the final accepted answer.

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

3.3 Related Work

For several years, researchers have been developing techniques to mine Stack Overflow and other software artifacts for information to help software developers. A key mechanism for providing the mined information is through recommendations in an Integrated Development Environments (IDE). Common recommendations include programming errors and exceptions [9,48,138], linking relevant discussions to any source code based on context [13, 130] or through a query in the IDE [53, 73].

Treude and Robillard proposed an approach for automatically augmenting API documentation with informative sentences from Stack Overflow [174]. Rahman et al. [137] proposed a technique that takes a natural language query as input and uses keyword-API associations mined from Stack Overflow to produce API recommendations as output. Other works for enriching API documentations include augmenting code examples [90, 166], identifying iOS and Android API classes for which developers regularly face usage obstacles [183], and integrating crowdsourced frequently asked questions (FAQs) from the web into API documents [39].

Code snippets and their descriptive text in Stack Overflow have also been analyzed to automatically generate comments for source code in open source projects [134, 186]. Nagy and Cleve [115] mined code blocks that appear in Stack Overflow questions to identify common error patterns in SQL statements. Yang et al. [193] investigated the usability of code snippets on Stack Overflow. Badashian et al. [15] used developers' contributions to Stack Overflow as a proxy for their expertise for consideration during bug triaging.

Stack Overflow has also been analyzed to build thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40,169,201]. Others have applied topic analysis and mining of domain-specific information [180,205], exploring gender bias [67, 104, 113], and emotions [118, 119].

Stack Overflow includes built-in quality signaling in the form of up & down votes, accepted answers, and user reputation. Some researchers [53,130,134,138,186,193] have used these quality signals to filter the posts that they use as input for their mining techniques. The need for such filtering partially motivates our study, as we want to understand, for instance, whether and how developers indicate accepted answers in chat communities.

To our knowledge, no research has focused on mining similar information from chat communities. However, developer chats have served as a mining source to extract specific types of information such as rationale [7, 8] and feature requests [151]. Researchers have also studied chat communities to learn about how they are used by development teams and the usefulness of the conversations for understanding developer behaviors [104, 124, 152, 164, 197].

More recently, researchers have focused on investigating chatbots related to several software development activities [100, 122, 143]. Slack provides easy integration to other frequently used developer tools (e.g., Github, Bitbucket, JIRA, and Jenkins) through a set of conversation-based bots and apps. These bots and apps have been widely adopted by many developers for different software engineering tasks such as maintaining code quality, testing, conducting development operations, supporting customers, and creating documentation [100].

3.4 Summary

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

At first sight, one might believe that the informal nature and interleaved conversations of Slack would make it difficult to mine automatically. There is also no pre-defined notion of conversation; each could span from two messages to hundreds.

However, our work to implement this study required us to disentangle conversations. We found that adapting the technique and training sets can indeed achieve high accuracy in disentangling the Slack conversations.

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

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

Participatory Q&A conversations are available on Slack in large quantities. These conversations often provide interesting insights about various technologies and their use, incorporating various design choices. While such conversations have also been observed on mailing lists, on Slack, the availability and speed of such conversations is much higher.

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

Chapter 4

CHAT DATASET THROUGH DISENTANGLEMENT FOR INFORMATION ANALYSIS AND EXTRACTION

In this chapter, we present and make publicly available a dataset of software-related Q&A chat conversations, curated for two years from three open Slack communities (python, clojure, elm). Our dataset consists of 38,955 conversations, 437,893 utterances, contributed by 12,171 users. We also share the code for a customized machine-learning based algorithm that automatically extracts (or disentangles) conversations from the downloaded chat transcripts.

4.1 Problem and Overview

Public Slack channels have been created around specific software technologies allowing participants to ask and answer a variety of questions. Our preliminary studies in Chapter 3 show that public Slack chat transcripts contain valuable information, which could provide support for improving automatic software maintenance tools or help researchers understand developer struggles or concerns. However, there are two key barriers to collecting data from Slack for conducting research: (1) free Slack teams only allow access to the 10,000 most recent messages (across all channels), and (2) data is only available to community members who have received an API token from an administrator. Hence, we create a dataset of software-related Slack conversations and make it openly available to be downloaded for further reuse by the community.

Different from many other sources of software development-related communication, the information on chat forums is shared in an unstructured, informal, and asynchronous manner. There is no predefined delineation of conversation in chat communications; each conversation could span from two messages to hundreds. Chat

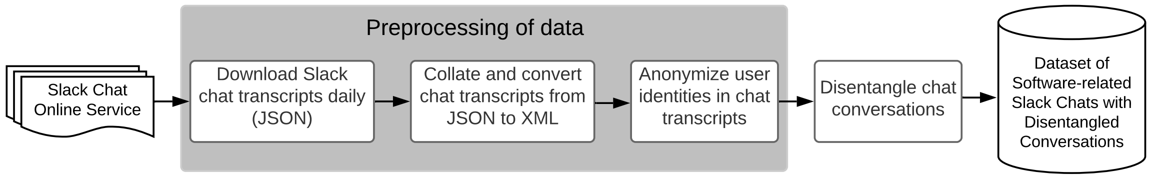

Slack Chat Online Service
Preprocessing of data
Download Slack chat transcripts daily (JSON)
Collate and convert chat transcripts from JSON to XML
Anonymize user identities in chat transcripts
Disentangle chat conversations
Dataset of Software-related Slack Chats with Disentangled Conversations

Figure 4.1: Overview of Data Collection, Preprocessing and Storage of Slack Chats

conversations are also often interleaved, where multiple questions are discussed and answered in parallel by different participants. Therefore, a technique is required to separate, or *disentangle*, the conversations for analysis by researchers or automatic mining tools.

In this chapter, we describe a released dataset of software-related developer chat conversations. A subset of this dataset was analyzed as part of our research in understanding the content of developer chat conversations on publicly available Slack channels [31]. We publish our dataset in XML format, where each XML node represents a chat utterance, containing the anonymized name of the participant, a timestamp, the message text, and an attribute (conversation id) to associate the message with its corresponding conversation. The conversation id is created through a chat disentanglement technique, which is a modified version of Elsner and Charniak's well-known algorithm that better matches the constraints of Slack and the type of software-related Q&A conversations in our corpus [61].

4.2 Methodology

Figure 4.1 presents an overview of our process for automatic data collection, preprocessing, disentanglement and storage of Slack developer chats. First, we download daily chat transcripts from each Slack channel in JSON format. Second, we collate the daily chat transcripts and convert them into XML format. Next, we anonymize the user identities of the chat participants to preserve privacy, as, otherwise, the Slack user ids can be used to retrieve the participant's e-mail via the channel of origin. Finally, we run a disentanglement algorithm to produce XML attributes that associate identified utterances (i.e., messages) with their corresponding conversations.

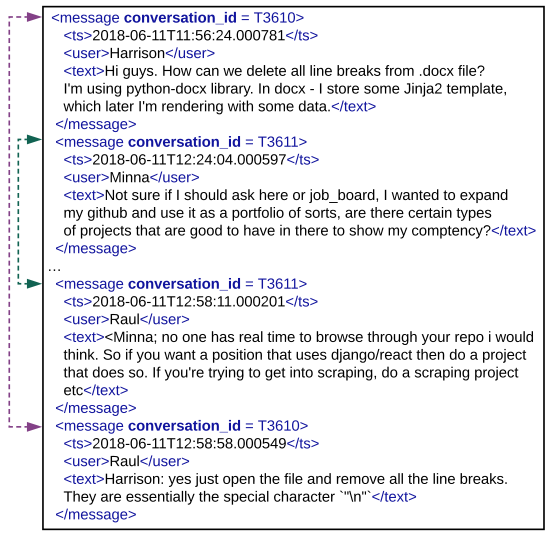
4.2.1 Data Selection

For the purpose of creating a dataset reusable for software developers and maintenance tools, we identified groups that primarily discuss software development topics and have a substantial collection of participants. We selected three programming communities who have active presence on Slack, and were willing to provide us API tokens for download. Within those selected communities, we focused on four channels that follow a Q&A format: pythondev#help, clojurians#clojure, elmlang#beginners, and elmlang#general. The channels are advertised on the Web and allow anyone to join, with a joining process only requiring the participant to create a username (any unique string) 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).

4.2.2 Data Collection and Preprocessing

Because programmatic access to the data in Slack communities is controlled by the administrators of the Slack team, we contacted several public Slack teams and asked for an API token that would allow us to read and store their data. Public Slack teams typically use Slack's free tier, which only stores the most recent 10,000 messages. Thus, for each Slack community, we downloaded all of the discussion data from each channel incrementally, every day for two years (July 2017- Jun 2019).

The downloaded chats from Slack were in JSON format. We collated all the downloaded chat transcripts and converted them to XML format, in which each message contains a timestamp, the id of the participant, and the message text. During the JSON to XML file conversion, we only use Slack events that correspond to messages, ignoring all other recorded events (e.g., channel joins). In the next step, we obfuscated the participant's ids for privacy, by replacing the original usernames with randomly generated human names.



```
<message conversation_id = T3610>
  <ts>2018-06-11T11:56:24.000781</ts>
  <user>Harrison</user>
  <text>Hi guys. How can we delete all line breaks from .docx file?
  I'm using python-docx library. In docx - I store some Jinja2 template,
  which later I'm rendering with some data.</text>
</message>
<message conversation_id = T3611>
  <ts>2018-06-11T12:24:04.000597</ts>
  <user>Minna</user>
  <text>Not sure if I should ask here or job_board, I wanted to expand
  my github and use it as a portfolio of sorts, are there certain types
  of projects that are good to have in there to show my comptency?</text>
</message>
...
<message conversation_id = T3611>
  <ts>2018-06-11T12:58:11.000201</ts>
  <user>Raul</user>
  <text><Minna; no one has real time to browse through your repo i would
  think. So if you want a position that uses django/react then do a project
  that does so. If you're trying to get into scraping, do a scraping project
  etc</text>
</message>
<message conversation_id = T3610>
  <ts>2018-06-11T12:58:58.000549</ts>
  <user>Raul</user>
  <text>Harrison: yes just open the file and remove all the line breaks.
  They are essentially the special character `"\n"`</text>
</message>
```

Figure 4.2: Data Format

4.2.3 Conversation Disentanglement

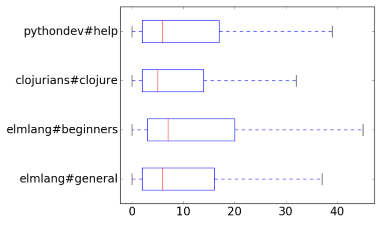
Since messages in chats form a stream, with conversations often interleaving such that a single conversation thread is entangled with other conversations, a technique is required to separate, or *disentangle*, the conversations for analysis. Figure 4.2 shows an example of an interwoven conversation in pythondev#help channel on Slack. In this example, a question follows another question, while the answers do not follow a chronological order; the third and the fourth utterances are answers to the second and first questions, respectively. This free form nature of chat communications makes the task of tracing and understanding chat transcripts difficult for automated tools.

The chat disentanglement problem has been studied before in the context of IRC and similar chat platforms [178]. We leveraged the effective technique proposed by Elsner and Charniak [61] that learns a supervised model based on a set of features between pairs of chat messages that occur within a window of time of each other. The features include the elapsed time between the message pair, whether the speaker in the two messages is the same, occurrence of similar words, use of cue words (e.g., hello, hi, yes, no), the use of technical jargon, among others. For the training set, we manually disentangled a set of 500 messages from each Slack channel and trained the model using the combined set.

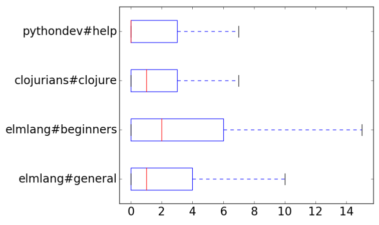
After we observed that some Slack channels can become dormant for a few hours at a time and that participants can respond to each other with considerable delay, we modified Elsner and Charniak's algorithm to expand the window of message pairs. Our modification computes features between the current utterance and every utterance that 1) occurred $<=$ 1477 (1.5^{18}) seconds prior to it, or 2) is within the last 5 utterances observed in the channel. We also added to the set of features used by Elsner and Charniak, introducing several specific to Slack, for instance, the use of URLs, Slack channel references, or code blocks within a message. Leveraging the fact that our conversations are mostly Q&A, we added features corresponding to gratitude (e.g., thanks, this works, makes sense), which sometimes occurs at the end of a conversation, when the question is answered satisfactorily. We also followed the procedure prescribed by Elsner and Charniak to create a better set of technical words for the model by extracting all of the words occurring in Stack Overflow documents tagged with a particular tag that do not co-occur in the English Wikibooks corpus. To measure the accuracy of the disentangling process, we manually disentangled separate sets of messages from each of the channels. The model with our enhancements produced a micro-averaged F-measure of 0.80; a strong improvement over the vanilla Elsner and Charniak approach's micro-averaged F-measure of 0.66. Since during the process of creating the gold set of disentangled chat conversations annotators can disagree whether a new conversation branches off from the original or not, micro-averaged F-measure is considered more appropriate than the standard F-measure [61]. With the permision of the original authors, we provide our modified Elsner and Charniak disentanglement code along with the Slack dataset.

4.2.4 Data Format

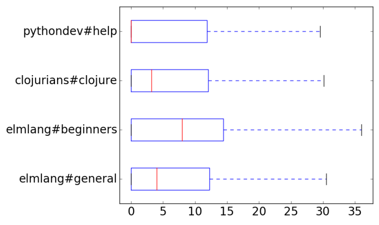
We publish our dataset in an XML format as shown in Figure 4.2, produced as an output of conversation disentanglement. In the disentangled files, each message has



(a) Conversation length



(b) Code Snippet count



(c) Code Snippet length

Figure 4.3: Box Plots of Measures by Community

Table 4.1: Dataset of Disentangled Slack Conversations

Community	**#Conversations**	**#Utterances**	**#Participants**
pythondev#help	8,887	106,262	3,295
clojurians#clojure	7,918	72,973	2,422
elmlang#beginners	13,169	168,689	3,695
elmlang#general	8,981	899,69	2,759
Total	**38,955**	**437,893**	**12,171**

<*message conversation_id*> which is a markup to associate each message with its corresponding conversation id, timestamp <*ts*> in Epoch format, anonymized participant names <*user*>, and the content <*text*> of the message.

4.3 Data Metrics

In Table 4.1, we show the breakdown of number of conversations, utterances, and participants for each of the 4 channels in our dataset. We also computed and report a few additional measures on our dataset that describe its basic characteristics: conversation length, code snippets, and urls. The results are displayed as boxplots in Figure 4.3.

Conversation length is defined as the number of sentences in a conversation. We computed this measure on the natural language text in each document using the sentence tokenizer from NLTK [22]. *Code snippet count* is computed as the number of code snippets per conversation, counting both inline and multiline code snippets in a conversation. In Slack, inline code snippets are enclosed in single quotes, whereas multiline code snippets are enclosed in triple quotes. *Code snippet length* is the number of non-whitespace characters in each code snippet.

As shown in Figure 4.3a, the median conversation lengths for each of the communities are similar, ranging from 5-7 sentences. Figure 4.3b indicates that *elmlang#beginners* can have larger number of code snippets than the other communities. The median code snippet count in *elmlang#beginners* is 2, whereas the median code snippet count in *elmlang#general* and *clojurians#clojure* is 1. The median code snippet count for *pythondev#help* is zero, probably because sufficient resources about coding in python are already available online. From Figure 4.3c, we observe that both the median and the variation of code snippet length for *elmlang#beginners* are larger than the rest of the communities. Intuitively, this is because *elmlang#beginners* is for novice programmers who frequently ask and answer more programming-related questions, such as errors and exceptions related to specific code snippets.

4.4 Related Work

Chat Disentanglement Techniques: Most previous research on conversation disentanglement has focused on developing data and models based on chats extracted from IRC channels [60, 61]. Elsner and Charniak's dataset and disentanglement algorithm, extracted from the #Linux IRC channel, has been used for training and evaluation in subsequent disentanglement research [88, 110]. Riou et al. [145] adapted Elsner and Charniak's technique [60] to a French corpus extracted from the Ubuntu platform, while Adams and Martell [3] investigated methods of topic detection and topic thread extraction. Lowe et al. [107, 108] used a heuristic-based approach to extract conversations from the #Ubuntu channel. More recently, Kummerfeld et al. [97] released a manually annotated IRC conversation disentanglement dataset with reply-to relations between messages. To the best of our knowledge, our work presents the first large-scale dataset of automatically disentangled software related conversations from the Slack platform.
Software-related Chat Datasets: Software-related chat datasets have served as a mining source to extract specific types of information related to software engineering tasks, such as rationale and feature requests [6–8, 151]. Apart from supporting

knowledge gathering, chat datasets could be potentially used in dialog research, such as curating a training corpus for virtual assistants that support software development tasks [34,58,107].

4.5 Limitations and Extensions

Our dataset originates from public Slack channels, focusing on conversations that start with a question followed by a discussion with answers. Thus, the content of our dataset, to some extent, resembles Q&A based forums such as Stack Overflow. If others are interested in datasets that represent team dynamics inside an organization, they would need to augment with private conversations.

We selected the chat transcripts from Slack, which is one of the most popular software developer chat communities. We chose three active programming language communities (4 Slack channels) for our dataset. There is a broad set of topics related to a particular programming language in each channel; however, if others want broader topics represented in their datasets, they will need to broaden the set.

We modified Elsner and Charniak's disentanglement algorithm to account for several features specific to Slack. The code of our modified disentanglement algorithm may need to be adapated to work well on other chat platforms or developer communications. Any changes in disentangled conversations could be handled manually by post processing or by further automation adaptation.

4.6 Summary and Discussion

In this chapter, we presented a dataset of software-related chat conversations from three programming communities on Slack (python, clojure, elm), gathered over two years (July 2017- June 2019). The overall dataset consists of 38,955 conversations, 437,893 utterances, contributed by 12,171 users. To enable others to process additional Slack transcripts and disentangle them into conversations, we also share the code we used to process daily chat logs, convert them to XML, and extract individual conversations from the collected chat transcripts. Both code for disentanglement algorithm and data[1] are openly available to be downloaded for further reuse by the community.

Our dataset could be explored for several new research opportunities in mining chats, some of which are discussed next. In our previous study [31], we found that Q&A chats in Slack provide the same information as can be found in Q&A posts on Stack Overflow. Over the years, researchers have mined the knowledge embedded in Q&A forums, such as Stack Overflow, for supporting IDE recommendation [13, 130, 138], learning and recommendation of APIs [38, 137, 183], automatic generation of comments for source code [134, 186], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40, 169]. Presence of similar information in Slack Q&A chats suggests that it can serve as a resource for several mining-based software engineering tools.

Developers use Slack to share opinions on best practices, APIs, or tools (e.g., *API X has better design or usability than API Y*). Q&A forums such as Stack Overflow explicitly forbid posting of questions that ask for opinions or recommendations. However, it is clear that receiving opinions is valuable to software developers. The availability of opinions or recommendations in chats may lead to new mining opportunities for software tools.

We noticed that, along with few links to Stack Overflow and GitHub Gists, there were sporadic links to other sites in our dataset. We believe that embedded links on Slack are used in many different contexts, and as such can be mined to provide more context to other data sources (tutorials, Q&A forums), and thus improve or augment developer learning resources.

Due to its increased popularity, Slack is becoming a popular media to disseminate information between software engineers across the globe. Lin et al. [104] have shown that developers use Slack to discover news/information on technological trends.

[1] https://zenodo.org/record/3627124

Our dataset could be studied to identify ‘hot’ topics of discussion in a programming community [149], and understand common challenges and misconceptions among developers [16]. The results of these studies would provide guidance to future research in developing software support and maintenance tools.

The widespread use of chat communication platforms such as Slack provides a thriving opportunity to build new conversation-based tools and integrations, such as chatbots. Bots have become increasingly prominent due to the ease of their integration with communication tools and accessibility to various APIs and data sources [100]. Sharing chat datasets such as ours could potentially facilitate further research on training and designing chatbots for software development activities [122].

Chapter 5

AUTOMATICALLY IDENTIFYING POST HOC QUALITY CONVERSATIONS

In this chapter, we investigate automatically detecting developer conversations of post hoc quality (i.e., conversations that contain useful information for mining or reading) 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. To our knowledge, this is the first automated technique for detecting developer conversations of post hoc quality.

5.1 Problem and Overview

Our earlier study in Chapter 3 indicates that Q&A chats in Slack provide the same information as can be found in Q&A posts on Stack Overflow [31]. This suggests that chats can also serve as a resource for mining-based software engineering tools.

Understanding the quality of the information in the mining source is essential for building effective data-driven software tools. From our preliminary analyses (Section 5.2 and 5.3), 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 [24]. 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) [18, 55, 155, 194], to our knowledge, there are no known techniques to automatically assess the quality of the content in developer chat conversations.

In this chapter, 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.

5.2 Motivational Examples

Developer public chats are becoming increasingly important corpora for several applications including, understanding topic trends of developer discussions [8, 58, 59, 103, 152], extracting feature requests [151] and developer rationale [6], and building virtual assistants for software engineers [107]. 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 5.3), 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 5.1. For readability, the conversations in Tables 5.1 and 5.2 are shown as already disentangled. 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 conversation (1b) could be useful for researchers studying developer communications (e.g., confusion detection [57]), 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 [31]. 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

Table 5.1: Example Problem-Solving Conversations on *pythondev#help* (For readability, shown as already disentangled)

(a) Post hoc Quality Conversation

Author	**Utterance**
Alexia	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
Elaina	tee hee 🙂 `sed -r ’s|(.+)/[^/]+/([^/]+)/.+|\1\2|g’`
Corina	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
Elaina	‘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.; (.+)/ 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
Corina	Ah ok, thanks a lot for the explanation!

(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?

Table 5.2: 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	“‘$In[5] : 120/(18*3)Out[5] :>>> 120/(18*3)2.222...$“‘
Velva	‘$120.0/(18*3)$‘
Rachael	or ‘from_future_ import division‘
Carylon	>>> from_future__ import division >>> $140/(18*3)2.59259...$; 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‘?

applications that extract API-related information, ensuring completeness and credibility of information in the source is crucial. Table 5.2 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, 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 5.1) and API-related (Table 5.2) 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.

5.3 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. We specifically extract a subset of 400 randomly chosen developer conversations [1] from our previously released dataset [34]. Our *analysis dataset* of 400 developer

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

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.

We followed a quantitative and qualitative analysis procedure [148] to analyze the ratings of the Slack conversations. The analysis procedure consisted of the following steps. 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.

Next, following a qualitative content analysis procedure, we independently studied the responses to the second question of the survey. More specifically, we used the technique of Explanation Building [148], 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 judges' 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.

Finally, 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. [31]*, (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..."*,

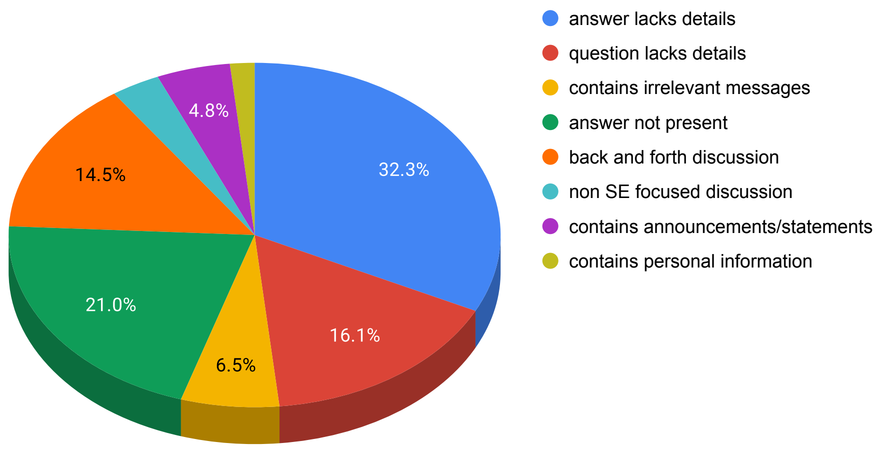


Figure 5.1: Study Responses Suggesting Characteristics of Poor Quality Conversations

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 5.1 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.

5.4 Automatic Identification Techniques

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.*

5.4.1 Conversation Disentanglement

In this chapter, we use a subset of our previously released disentangled chat dataset (in Chapter 4). 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.

5.4.2 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.

5.4.2.1 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 5.3 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 [103]. 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. [47] used 5W1H words and question mark to extract question-answer pairs from online forums, while others [93, 141] 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 [47] in the first utterance.

Table 5.3: Features to Identify Post Hoc Quality Conversations

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

Knowledge-seeking Question: Harper et al. [77] 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. [130] 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 [31]. 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, we designed a regular

expression for API mentions in each programming language (python, clojure, elm, racket) in our manual analysis dataset (Section 5.3). We count the number of unique APIs mentioned in the natural language text in a conversation [155] (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 [50, 51, 190] and quality of questions [130] and answers on Stack Overflow [68, 155]. 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. [11] 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 [32]. 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 [50, 190]. Hence, we count number of non-whitespace characters in all code snippets that occur in each conversation [68, 155].

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 5.3) 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 [41, 81, 170] (e.g., deprecated, wrapper, debugger, etc) in a conversation using a list that combines morphological terms collected by Chen et al. [41] and software-related terms by Christensson [45].

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 [11, 68, 129]. We use the word and sentence tokenizer from NLTK [22] 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. [130] 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, 129] to count the number of sentences in a conversation containing one or more text speak instances.

#Questions: Kitzie et al. [93] 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 [47] in the beginning of a sentence, to count questions in a conversation.

Unique Info: Kitzie et al. [94] 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 [93, 141]. We used pyspellchecker to count misspelled words detected by computing the word's Levenshtein distance [116] from words in a corpus of English and commonly used terms in software engineering [41, 81, 170].

#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 [130], predict unanswered questions [11] and estimate question quality on Q&A forums [93, 141]. We compute several readability measures including Automated Reading Index [157], Coleman Liau

Index [46], Flesch Reading Ease Score [65], Flesch Kincaid Grade Level [65], Gunning Fog Index [74], and SMOG Grade [109] 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 [18, 51, 129]. 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 questioners' 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.

5.4.2.2 Machine Learning-based Classifiers

With our features, we trained multiple classifiers using the Weka toolkit [76] (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 [5, 50], and classifying high quality questions on Stack Overflow and Yahoo! Answers [55, 68, 102, 141].

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 [50, 51] and identifying high quality content in social media [4].

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 [55, 141].

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.

Table 5.4: 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**

5.5 Evaluation Study

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

5.5.1 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 (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.

5.5.2 Evaluation Dataset

5.5.2.1 Source and Size

We extract a subset of the developer conversations [2] of our previously released dataset [34], following the data selection procedure for our studies in section 5.3. 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 5.4, consists of 2000 conversations (400 from each of five communities), 37,977 utterances, and 1,400 users.

5.5.2.2 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 5.3. *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 *python-dev#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

[2] https://www.zenodo.org/record/3763432 - version 2

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 [25]. We computed Cohen's Kappa inter-rater agreement 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) [99]. Thus, the judges separately annotated the remaining conversations to reach 2000.

5.5.3 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,

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

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.

5.5.4 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 5.3, 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 5.3) 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 5.4.2.1 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

Table 5.5: 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

triple quotes. Hence, for *Q&A:Code*, we use the presence of triple quotes to detect a code segment.

5.5.5 Evaluation Results and Discussion

RQ1: How effective are machine learning-based techniques for automatic identification of post hoc quality developer chats?

Table 5.5 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

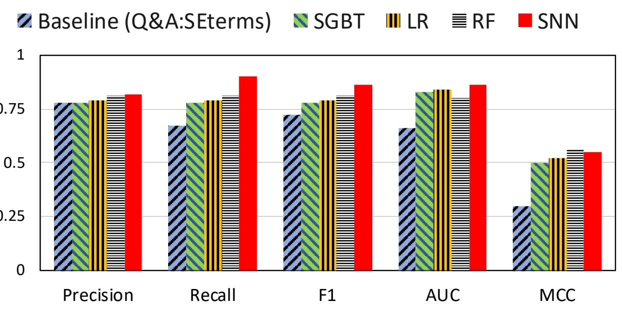


Figure 5.2: Comparing Effectiveness (with all features)

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 5.5, 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 5.2. We have bolded *Q&A:SEterms* in Table 5.5, to emphasize that *Q&A:SEterms* performs 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?

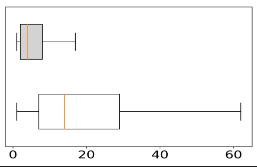
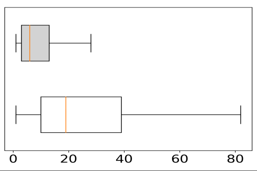
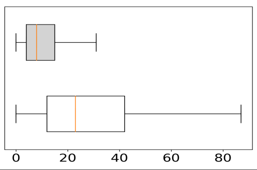
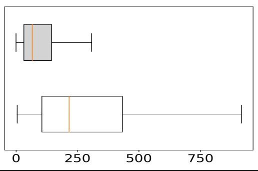
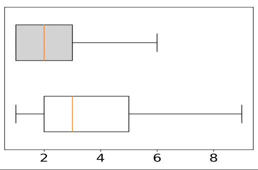
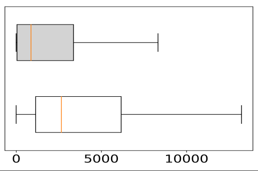
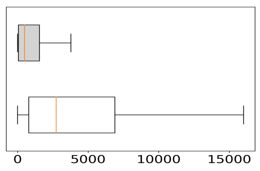
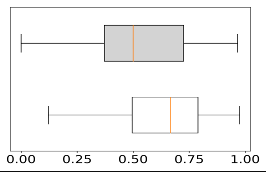
Classifier Effectiveness. Figure 5.2 graphically depicts the overall effectiveness using *Q&A:SEterms* for the baseline and all features for the ML-based classifiers. From Table 5.5, and Figure 5.2 we observe that precision across all methods is nearly the same (except *Q&A:Code* which has a precision 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

Table 5.6: 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	

dataset in Section 5.3, 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 [133] of each feature in our feature sets. Table 5.6 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 5.6 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 5.6 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 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

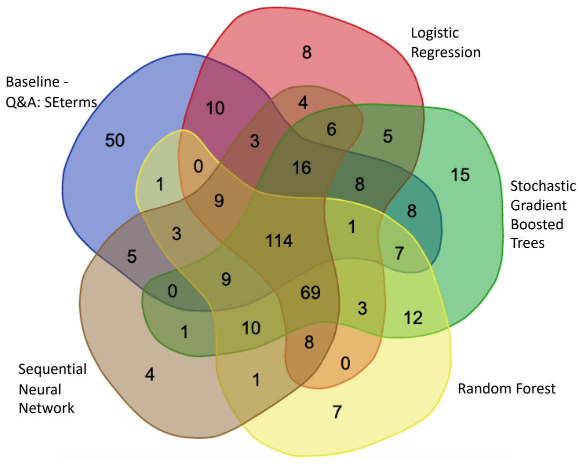


Figure 5.3: False Positive Overlap Among Approaches

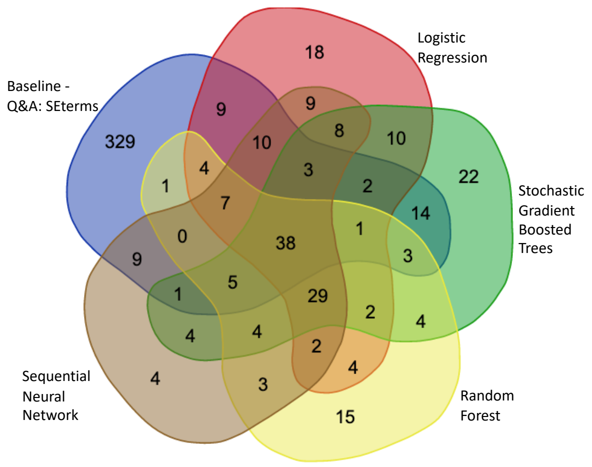


Figure 5.4: False Negative Overlap Among Approaches

(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 [148], we 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 5.3 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 5.4 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.

5.5.6 Threats to Validity

Construct Validity: The constructs to measure the phenomena under the study are the features for the machine learning-based approach, which 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. 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 [25,99]. 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 5.3) to refine our understanding of *post hoc quality* conversations.

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 5.5.2.2). Since chat communications are informal in nature, it is possible that punctuations are omitted in the text [203]. 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 5.3 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 5.3) and the features (Section 5.4.2.1) 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.

5.6 Related Work

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) [23]. 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 ($\sim$24) elected to become Stack Overflow moderators, who can perform additional tasks, such as closing or opening questions [51]. 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 [129]. 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 [155]. Yang et al. observed that good answers mostly contain multiple line code [194]. Duijn et al. determined that code-to-text ratio and code readability are most important in determining question quality [55]. 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 [18]. Correa and Sureka proposed a predictive model to detect the deletion of a Stack Overflow question at creation time [50,51]. Ponzanelli et al. automatically identify and remove misclassified good quality Stack Overflow posts from the review queue [129]. Yao et al. predict high impact questions and useful answers just after they are posted by predicting voting scores [195]. Some researchers [5, 200] 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 [4], designing techniques to automatically predict or detect question quality [93, 94, 102, 141], and best answer prediction on Community Question Answering (CQA) forums [64, 78, 112]. Harper et al. found that conversational questions have potentially lower archival value than informational questions [77]. Guy et al. replicated [77] 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.

5.7 Summary

This chapter reports on the first work to automatically identify and extract *post hoc quality* information (i.e., information useful for reading or mining) from developer chat communications. 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.

Chapter 6

AUTOMATIC EXTRACTION OF OPINION-BASED Q&A FROM ONLINE DEVELOPER CHATS (CHATEO)

In this chapter, we study the use of online chat platforms as a resource towards collecting developer opinions that could potentially help in building opinion question answering (Q&A) systems, as a specialized instance of virtual assistants and chatbots for software engineers. Opinion Q&A has a stronger presence in chats than in other developer communications, thus mining them can provide a valuable resource for developers in quickly getting insight about a specific development topic (e.g., *What is the best Java library for parsing JSON?*). We address the problem of opinion Q&A extraction by developing automatic identification of opinion-asking questions and extraction of participants' answers from public online developer chats.

6.1 Problem and Motivation

Recognizing the increasing capabilities of virtual assistants that use conversational artificial intelligence (AI) (e.g., chatbots, voice assistants), some researchers in software engineering are working towards the development of virtual assistants to help programmers. They have conducted studies to gain insights into the design of a programmer conversational agent [98], proposed techniques to automatically detect speech acts in conversations about bug repair to aid the assistant in mimicking different conversation types [188], and designed virtual assistants for API usage [56].

While early versions of conversational assistants were focused on short, task-oriented dialogs (e.g., playing music or asking for facts), more sophisticated virtual assistants deliver coherent and engaging interactions by understanding dialog nuances such as user intent (e.g., asking for opinion vs knowledge) [139]. They integrate specialized instances dedicated to a single task, including dialog management, knowledge retrieval, opinion-mining, and question-answering [86]. To build virtual assistants for software engineers, we need to provide similar specialized instances based on the available information from software engineers' daily conversations. Recent studies indicate that online chat services such as IRC, Slack, and Gitter are increasingly popular platforms for software engineering conversations, including both factual and opinion information sharing and now playing a significant role in software development activities [31, 103, 146, 153]. These conversations potentially provide rich data for building virtual assistants for software engineers, but little research has explored this potential.

In this chapter, we leverage the availability of opinion-asking questions in developer chat platforms to explore the feasibility of building opinion-providing virtual assistants for software engineers. Opinion question answering (Opinion QA) systems [17, 126, 165, 182] aim to find answers to subjective questions from user-generated content, such as online forums, product reviews, and discussion groups. One type of virtual assistant that can benefit from opinions are Conversational Search Assistants (CSAs) [52]. CSAs support information seekers who struggle forming good queries for exploratory search, e.g., seeking opinions/recommendations on API, tools, or resources, by eliciting the actual need from the user through conversation. Studies indicate developers conducting web searches or querying Q&A sites for relevant questions often find it difficult to formulate good queries [117, 140]. Wizard of Oz studies have explicitly shown the need for opinions within CSAs [181]. A key result of this work is the availability of opinions on chat platforms, which would enable the creation of a sizable opinion Q&A corpus that could actually be used by CSAs. The opinion Q&A corpus generated from chats by our technique can be used in a few different ways to build a CSA: 1) matching queries/questions asked to the CSA with questions from the corpus and retrieving the answers; 2) summarizing related groups of opinion Q&A to generate (e.g., using a GAN) an aggregate response for a specific software engineering topic.

Opinion extraction efforts in software engineering have focused on API-related opinions and developer emotions from Q&A forums [37, 106, 120, 135, 177], developer sentiments from commit logs [156], developer intentions from emails and issue reports [82, 159] and detecting software requirements and feature requests from app reviews [28, 75]. These studies suggest that, beyond reducing developers' effort of manual searches on the web and facilitating information gathering, mining of opinions could help in increasing developer productivity, improving code efficiency [177], and building better recommendation systems [106].

Findings from an exploratory study [31] of Slack conversations in Chapter 3 suggests that developer chats include opinion expression during human conversations. Thus, we investigate the problem of opinion Q&A extraction from public developer chats in this chapter. We decompose the problem of opinion Q&A extraction into two subproblems: (1) identifying questions where the questioner asks for opinions from other chat participants, (which we call posing an opinion-asking question) and (2) extracting answers to those opinion-asking questions within the containing conversation.

Researchers extracting opinions from software-related documents have focused on identifying sentences containing opinions, using lexical patterns [159] and sentiment analysis techniques [106,176,177]. However, these techniques are not directly applicable to identifying opinion-asking questions in chats for several reasons. Chat communities differ in format, with no formal structure and informal conversation style. The natural language text in chats could follow different syntactic patterns and contain incomplete sentences [31], which could potentially inhibit automatic mining of opinions.

Outside the software engineering domain, researchers have addressed the problem of answer extraction from community question-answering (CQA) forums by using deep neural network models [150,184,204], and syntactic tree structures [89,114]. Compared to CQA forums, chats contain rapid exchanges of messages between two or more developers in short bursts [31]. A question asked at the start of a conversation may be followed by a series of clarification or follow-up questions and their answers, before the answers to the original question are given. Moreover, along with the answers, conversations sometimes contain noisy and unrelated information. Therefore, to determine the semantic relation between a question and answer, understanding the context of discussion is crucial.

We are the first to extract opinion Q&A from developer chats, which could be used to support SE virtual assistants as well as chatbots, programmer API recommendation, automatic FAQ generation, and in understanding developer behavior and collaboration. Our automatic opinion Q&A extraction takes a chat conversation as input and automatically identifies whether the conversation starts with an opinion-asking question, and if so, extracts one or more opinion answers from the conversation.

6.2 Opinion-asking Questions in Developer Online Communications

Since developer chats constitute a subclass of developer online communications, we began by investigating whether we could gain insights from work by others on analyzing the opinion-asking questions in other kinds of developer online discussions (emails, issue reports, Q&A forums).

Emails. The most closely related work, by Di Sorbo et al. [159], proposed an approach to classify email sentences according to developers' intentions (feature request, opinion asking, problem discovery, solution proposal, information seeking and information giving). Their taxonomy of intentions and associated linguistic patterns have also been applied to analyze user feedback in app reviews [54, 125].

In their taxonomy, Di Sorbo et al. define "opinion asking" as: *requiring someone to explicitly express his/her point of view about something (e.g., What do you think about creating a single webpage for all the services?*. They claim that sentences belonging to "opinion asking" may emphasize discussion elements useful for developers' activities; and thus, make it reasonable to distinguish them from more general information requests such as "information seeking". Of their manually labelled 1077 sentences from mailing lists of Qt and Ubuntu, only 17 sentences (1.6%) were classified as "opinion asking", suggesting that opinion-asking questions are infrequent in developer emails.

Table 6.1: Example of Opinion-asking Question And Answers on Slack #python-dev channel

Ques: hello everyone, I've requirement where dataset is large like 10 millions records, i want to use django rest framework in order to provide that data. Question: which one is the best ORM which is efficient for large datasets? 1. Django ORM 2. SQL alchemy

Ans 1: SQLalchemy is more performant especially if you're using Core.
Ans 2: yea, you can mix sqlalchemy and django orm. it's all just python at the end of the day. however, if you swap out one for the other you lose all the benefits of the django orm and how it integrates with the framework.
Ans 3: If performance is a factor than use SQLAlchemy core to work on this large data set If deadlines are more of a factor that use Django ORM since you're already use DRF. Just make sure to use eager loading on relationships where you can to optimize queries.

Issue Reports. To investigate the comprehensiveness and generalizability of Di Sorbo et al.'s taxonomy, Huang et al. [82] manually labelled 1000 sentences from issue reports of four projects (TensorFlow, Docker, Bootstrap, VS Code) in GitHub. Consistent with Di Sorbo et al.'s findings, Huang et al. [82] reported that only 11 (1.1%) sentences were classified as "opinion asking". Given this low percentage and that "opinion asking" could be a sub-category of "information seeking", they merged these two categories in their study. To broaden their search of opinions, Huang et al. introduced a new category "aspect evaluation", defined as: *express opinions or evaluations on a specific aspect (e.g., "I think BS3's new theme looks good, it's a little flat style.", "But I think it's cleaner than my old test, and I prefer a non-JS solution personally.?).*" They classified 14-20% sentences as "aspect evaluation". Comparing the two definitions and their results, it is evident that although opinions are expressed widely in issue reports, questions asking for others' opinions are rare.

Chats. Chatterjee et al.'s [31] results showing potentially more opinions in Slack developer chats motivated us to perform a manual study to systematically analyze the occurrence of opinion-asking questions and their answers in developer chats.

Dataset: To create a representative analysis dataset, we identified chat groups that primarily discuss software development topics and have a substantial number of participants. We selected three programming communities with an active presence on Slack. 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. Our analysis dataset of 400 Slack developer conversations consists of 100 conversations from Slack Python-dev#help channel, 100 from clojurians#clojure, 100 from elmlang#beginners, and 100 from elmlang#general, all chosen randomly from the dataset released by Chatterjee et al. [34].

Procedure: Using the definition of opinion-asking sentences proposed by Di Sorbo et al. [159], we independently identified conversations starting with an opinion-asking question. We also investigated if those questions were answered by others in a conversation. The authors annotated a shared set of 400 conversations, which indicates that the sample size is sufficient to compute the agreement measure with high confidence [25]. We computed Cohen's Kappa inter-rater agreement between the 2 annotators, and found an agreement of 0.74, which is considered to be sufficient (> 0.6) [99]. The annotators further discussed their annotations iteratively until all disagreements were resolved.

Observations: We observed that out of our 400 developer conversations, 81 conversations (20%) start with an opinion-asking question. There are a total of 134 answers to those 81 opinion-asking questions, since each conversation could contain no or multiple answers.

Table 6.1 shows an opinion-asking question (*Ques*) and its answers (*Ans*) extracted from a conversation on #python-dev channel. Each of the answers contain sufficient information as a standalone response, and thus could be paired with the question to form separate Q&A pairs. Given that conversations are composed of a sequence of utterances by each of the people participating in the conversation in a back and forth manner, the Q&A pairs are pairs of utterances.

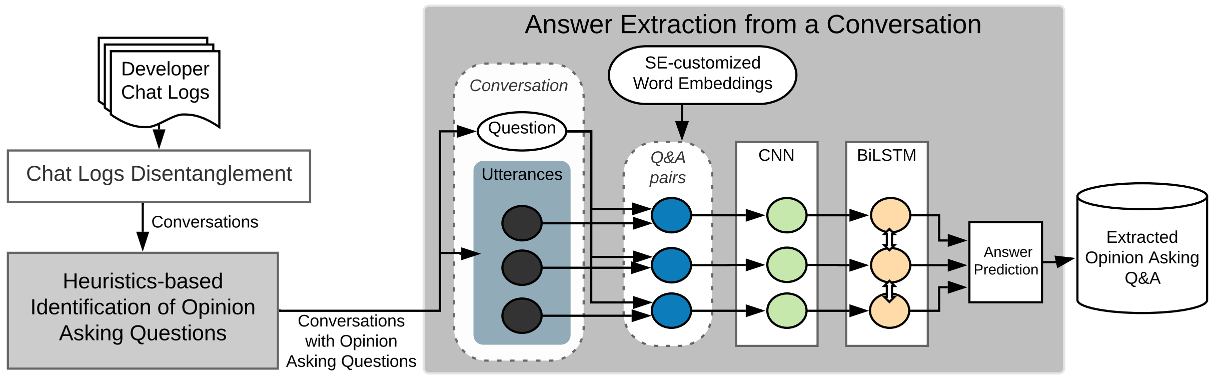

Developer Chat Logs
Chat Logs Disentanglement
Conversations
Heuristics-based Identification of Opinion Asking Questions
Conversations with Opinion Asking Questions
Answer Extraction from a Conversation
Conversation
Question
Utterances
SE-customized Word Embeddings
Q&A pairs
CNN
BiLSTM
Answer Prediction
Extracted Opinion Asking Q&A

Figure 6.1: Overview of ChatEO Automatic Extraction of Opinion Q&A from Developer Chats

Summary: Compared to other developer communications, conversations starting with opinion-asking questions in developer chats are much more frequent. Thus, chats may serve as a better resource to mine for opinion Q&A systems.

6.3 Automatically Extracting Opinion Q&A from Developer Chats

Figure 6.1 describes the overview of our approach, ChatEO, to automatically *E*xtract *O*pinion Q&A from software developer *Chat*s. Our approach takes a developer chat history as input and extracts opinion Q&A pairs using the three major steps: (1) Individual conversations are extracted from the interleaved chat history using conversation disentanglement. (2) Conversations starting with an opinion-asking question are identified by applying textual heuristics. (3) Possibly multiple available answers to the opinion-asking question within the conversation are identified using a deep learning-based approach.

6.3.1 Conversation Disentanglement

As described in Chapter 4, utterances in chats form a stream and conversations often interleave such that a single conversation thread is entangled with other conversations. Hence, to ease individual conversation analysis, we separate, or disentangle, the conversation threads in each chat log. In this chapter, we used the best available disentanglement approaches proposed for Slack and IRC chat logs, respectively:

- Slack chat logs: We used a subset of the publicly available disentangled Slack chat dataset[1] described in Chapter 4 since our modified disentanglement algorithm customized for Slack developer chats achieved a micro-averaged F-measure of 0.80.

- IRC chat logs: We used Kummerfeld et al.'s [96] technique, a feed-forward neural network model for conversation disentanglement, trained on 77K manually annotated IRC utterances, and achieving 74.9% precision and 79.7% recall.

In the disentangled conversations, each utterance contains a unique conversation id and metadata including timestamp and author information.

6.3.2 Heuristics-based Identification of Opinion-asking Question

Di Sorbo et al. [159] claim that developers tend to use recurrent linguistic patterns within discussions about development issues. Thus, using natural language parsing, they defined five linguistic patterns to identify opinion-asking questions in developer emails. First, to investigate the generalizability of Di Sorbo et al.'s linguistic patterns, we used their replication package of DECA[2] to identify opinion-asking questions in developer chats. We found that, out of 400 conversations in our manual analysis dataset, only 2 questions were identified as opinion-asking questions by DECA. Hence, we conducted a deeper analysis of opinion-asking questions in our manual dataset in Section 6.2, to identify additional linguistic patterns that represent opinion-asking questions in developer chats.

We followed a qualitative content analysis procedure [148], where the same two annotators first independently analyzed 100 developer conversations to understand the structure and characteristics of opinion-asking questions in chats. The utterances of the first speaker in each conversation were manually analyzed to categorize if they were asking for an opinion. When an opinion-asking question was manually identified, the

[1] https://www.zenodo.org/record/3627124

[2] https://www.ifi.uzh.ch/en/seal/people/panichella/tools/DECA.html

annotator identified the parts of the the utterance that contributed to that decision and identified part-of-speech tags and recurrent keywords towards potential linguistic patterns. Consider the question in Table 6.1. First, the annotator selects the text that represents an opinion-asking question, in this case: "which one is the best ORM which is efficient for large datasets?". 'ORM" is noted as a noun referring to the library, and "best" as an adjective related to the opinion about ORM. Thus, a proposed linguistic pattern to consider is: "*Which one <verb_to_be> [positive_adjective] [rec_target_noun] <verb_phrase>?*".

Throughout the annotation process, the annotators wrote memos to facilitate the analysis, recording observations on types of opinions asked, observed linguistic patterns of opinion-asking questions, and researcher reflections. The two annotators then met and discussed their observations on common types of questions asking for opinions, which resulted in a set of preliminary patterns for identifying opinion-asking questions in developer chats. Using the preliminary patterns, the two authors then independently coded the rest of the opinion-asking questions in our manual analysis dataset, after which they met to further analyze their annotations and discuss disagreements. Thus, the analysis was performed in an iterative approach comprised of multiple sessions, which helped in generalizing the hypotheses and revising the linguistic patterns of opinion-asking questions.

Our manual analysis findings from 400 Slack conversations showed that an opinion-asking question in a developer chat is a question occurring primarily at the beginning of a conversation and could exhibit any of these characteristics:

- Expects subjective answers (i.e., opinions) about APIs, libraries, examples, resources, e.g., "Is this a bad style?","What do you think?"
- Asks for which path to take among several paths, e.g., "Should I use X instead of Y?"
- Asks for an alternative solution (other than questioner's current solution), e.g., "Is there a better way?"

Table 6.2: Most Common Pattern for opinion-asking Question Identification in Chats.

Pattern code: P_ANY_ADJ
Description: question starting with "any" and followed by a positive adjective and target noun.
Rule: Any [rec_positive_adj] [rec_target_noun] verb_phrase
Definitions:
[rec_positive_adj] = "good","better","best","right","optimal",...
[rec_target_noun] = "project","api","tutorial","library",...
Example: *Any good examples or things I need to be looking at?*

Thus, we extended Di Sorbo et al.'s linguistic pattern set for identifying opinion-asking questions, by adding 10 additional linguistic patterns. Table 6.2 shows the most common pattern, P_ANY_ADJ, in our dataset with its description and example question. Most of the patterns utilize a combination of keywords and part-of-speech-tagging. The annotators curated sets of keywords in several categories, e.g., [rec_target_noun], [rec_verbs], [rec_positive_adjective] related to nouns, verbs, and adjectives, respectively. The complete set of patterns and keywords list are available in our replication package.

6.3.3 Answer Selection from a Conversation

We build upon work by Zhou et al. [204], who designed R-CNN, a deep learning architecture, for answer selection in community question answering (CQA). Since R-CNN was designed for application in CQA for the non-SE domain[3], we customize for application in chats for software engineering and then compare to the non-customized R-CNN in our evaluation. We chose to build on R-CNN because other answer extraction models [147, 168, 202] only model the semantic relevance between questions and answers. In contrast, R-CNN models the semantic links between successive candidate

[3] http://alt.qcri.org/semeval2015/task3/

answers in a discussion thread, in addition to the semantic relevance between question and answer. Since developer chats often contain short and rapid exchanges of messages between participants, understanding the context of the discussion is crucial to determine the semantic relation between question and answer. Hence, we adapt R-CNN to extract the relevant answer(s) to an opinion-asking question based on the context of the discussion in a conversation.

Zhou et al. [204] regarded the problem of answer selection as an answer sequence labeling task. First, they apply two convolution neural networks (CNNs) to summarize the meaning of the question and a candidate answer, and then generate the joint representation of a Q&A pair. The learned joint representation is then used as input to long short-term memory (LSTM) to learn the answer sequence of a question for labeling the matching quality of each answer.

To design ChatEO, we make the following adaptations to account for both the SE domain content and specifically software-related chats. First, we preprocess the text, apply a software-specific word-embedding model, and use those embeddings as input to a CNN to learn joint representation of a Q&A pair. We use TextCNN [92] since text in chat (utterances) are much shorter compared to CQA (post). The representations from the CNN are then passed as input to Bidirectional LSTM (BiLSTM) instead of LSTM to improve prediction of the answers from a sequence of utterances in a conversation. We detail ChatEO answer extraction as follows:

6.3.3.1 Preprocessing

To help ChatEO with the semantics of the chat text, the textual content in the disentangled conversations is preprocessed. We replace url, user mentions, emojis, and code with specific tokens *‘url’*, *‘username’*, *‘emoji’*, and *‘code’* respectively. To handle the informal style of communication in chats, we use a manual set of common phrase expansions (e.g., “you’ve” to “you have”). We then convert the text to lowercase.

6.3.3.2 SE-customized Word Embeddings

Text in developer chats and other software development communication can differ from regular English text found in Wikipedia, news articles, etc. in terms of vocabulary and semantics. Hence, we trained custom GloVe vectors on the most recent Stack Overflow data dump (as of June, 2020) to more precisely capture word semantics in the context of developer communications. To train GloVe vectors, we performed standard tokenization and preprocessing on each Stack Overflow post's title and text and trimmed extremely rarely occurring words (vocabulary minimum threshold of 100 posts; window size of 15 words). Our word embedding model thus consists of 123,995 words, where each word is represented by a 200-dimensional word vector. We applied this custom word embedding model to each word in each utterance of a conversation.

6.3.3.3 Convolutional Neural Networks

In natural language analysis tasks, a sentence is encoded before it is further processed in neural networks. We leverage the sentence encoding technique from TextCNN [92]. TextCNN, also used for other dialog analysis tasks [151], is a classical technique for sentence modeling which uses a shallow Convolution Neural Network (CNN) that combines n-gram information to model compact text representation. Since an utterance in a chat is typically short (<25 words on average), we take each utterance as a sentence and apply word embedding, multiple convolution, and max-pooling operations.

The input for TextCNN is the distributed representation of an utterance, created by mapping each word index into its pre-trained embeddings. Each utterance is padded to the same length n with zero vectors. Let $z_j \in \mathbb{R}^d$ denote the d-dimensional embedding for the jth word in an utterance. Thus, an utterance of length n can be represented by: $z_{1:n} = z_1 \oplus z_2 \oplus \ldots z_n$, where $\oplus$ is the concatenation operator. To gather local information, convolution is achieved by applying a fixed length sliding window (kernel) $w^m \in \mathbb{R}^{h \times d}$, on each word position i such that $n - h + 1$ convolutional units in the mth layer are generated by: $c_i^m = \sigma\left(w^m \cdot z_{i:i+h} + b^m\right), i = 0, 1, \ldots, n - h + 1,$

where h is the size of convolution kernel, σ is the activation function, and b^m is the bias factor for the mth layer. The convolution layer is followed by a max pooling layer, which can select the most effective information with the highest value. The flattened output vectors for each kernel after max-pooling are concatenated as the final output.

6.3.3.4 Bidirectional LSTM

The task of identifying answers in a conversation requires capturing the context and flow of information among the utterances inside a conversation. Hence, we use Bidirectional Long Short Term Memory (BiLSTM) [72], where the utterances of a conversation are considered as sequential data. The input to our BiLSTM is a sequence of utterance representations created by TextCNN. Variations of LSTM, widely used by researchers for answer extraction tasks [184, 191], are capable of modeling semantic links between continuous text to perform answer sequence learning.

LSTM [80] uses a gate mechanism to filter relevant information and capture long-term dependencies. An LSTM cell comprises of input gate (i), forget gate (f), cell state (c), and output gate (o). The outputs of LSTM at each time step h_t can be computed by the following equations:

$$\begin{bmatrix} i_t \\ f_t \\ \tilde{c}_t \\ o_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b \right)$$

$$c_t = \tilde{c}_t \odot i_t + c_{t-1} \odot f_t$$

$$h_t = o_t \odot \tanh(c_t)$$

where x_t is the ith element in the input sequence; W is the weight matrix of LSTM cells; b is the bias term; σ denotes sigmoid activation function, and $tanh$ denotes hyperbolic tangent activation function; $\odot$ denotes element-wise multiplication.

BiLSTM processes a sequence on two opposite directions (forward and backward), and generates two independent sequences of LSTM output vectors. Hence, the output of a BiLSTM at each time step is the concatenation of the two output vectors from both directions, $h = [\overrightarrow{h} \oplus \overleftarrow{h}]$, where $\overrightarrow{h}$ and $\overleftarrow{h}$ denote the outputs of two LSTM layers respectively, and $\oplus$ is the concatenation operator.

6.4 Evaluation Study

We designed our evaluation to analyze the effectiveness of the pattern-based identification of opinion-asking questions (*RQ1*) and of our answer extraction technique (*RQ2*).

6.4.1 Metrics

We use measures that are widely used for evaluation in machine learning and classification. To analyze whether the automatically identified instances are indeed opinion-asking questions and their answers, we use precision, the ratio of true positives over the sum of true and false positives. To evaluate how often a technique fails to identify an opinion-asking question or its answer, we use recall, the ratio of true positives over the sum of true positives and false negatives. F-measure combines these measures by harmonic mean.

6.4.2 Evaluation Datasets

We established several requirements for dataset creation to reduce bias and threats to validity. To curate a representative analysis dataset, we identified chat groups that primarily discuss software development topics and had a substantial number of participants. To ensure the generalizability of our techniques, we chose two separate chat platforms, Slack and IRC, which are currently the most popular chat platforms used by software developers. We selected six popular programming communities with an active presence on Slack or IRC. We believe the communities are representative of public software-related chats in general; we observed that the structure and intent of conversations are similar across all 6 communities.

To collect conversations on Slack, we downloaded the developer chat conversations dataset released by Chatterjee et al. [34]. To gather conversations on IRC,

Table 6.3: Evaluation Dataset
Samples(OA Question Identification) created from *Chat Logs(#Conv)*, *Samples(Answer Extraction)* created from *Chat Logs(#OAConv)*

Source	**Duration**	**Chat Logs**		**Samples**					
				OA Question Identification			**Answer Extraction**		
		#Conv	#OAConv	#Conv	#Utt	#Users	#Conv	#Utt	#Users
angularjs (IRC)	Apr2014-Jun2020	181662	9175	80	1812	90	891	11218	1109
c++ (IRC)	Oct2018-Jun2020	95548	841	60	630	62	127	2017	181
opengl (IRC)	Jul2005-Jun2020	135536	9198	60	698	54	258	3176	390
pythondev (Slack)	Jul2017-Jun2019	8887	707	80	848	93	160	1604	320
clojurians (Slack)	Jul2017-Jun2019	7918	562	60	570	81	156	1422	314
elmlang (Slack)	Jul2017-Jun2019	22150	1665	60	595	109	409	4535	846
Total		**451701**	**22148**	**400**	**5153**	**489**	**2001**	**23972**	**3160**

we scraped publicly available online chat logs[4]. After disentanglement, we discarded single-utterance conversations, and then created two separate evaluation datasets, one for opinion-asking question identification and a subset with only chats that start with an opinion-asking question, for answer extraction. We created our evaluation datasets by randomly selecting a representative portion of the conversations from each of the six programming communities.

Table 6.3 shows the characteristics of the collected chat logs, and our evaluation datasets, where *#OAConv* gives the number of conversations that we identified as starting with an opinion-asking question using the heuristics described in section 6.3.2, per community.

The question identification evaluation dataset consists of a total of 400 conversations, 5153 utterances, and 489 users. Our question extraction technique is heuristics-based, requiring conversations that do not start with an opinion- asking question. Thus, we randomly chose 400 from our 45K chat logs for a reasonable human-constructed goldset.

The evaluation dataset for answer extraction consists of a total of 2001 conversations, 23,972 utterances, and 3160 users. Our machine-learning-based answer extraction requires conversations starting with a question, and a dataset large enough

[4] https://echelog.com/

for training. Thus, 2001 conversations starting with opinion-asking questions were used.

RQ1. How effective is ChatEO in identifying opinion-asking questions in developer chats?

Gold Set Creation: We recruited 2 human judges with (3+ years) experience in programming and in using both chat platforms (Slack and IRC), but no knowledge of our techniques. They were provided a set of conversations where each utterance includes: the unique conversation id, anonymized name of the speaker, the utterance timestamp, and the utterance text. Using Di Sorbo et al.'s [159] definition of opinion-asking questions i.e., *requiring someone to explicitly express his/her point of view about something (e.g., What do you think about creating a single webpage for all the services?)*, the human judges were asked to annotate only the utterances of the first speaker of each conversation with value '1' for opinion-asking question, or otherwise '0'.

The judges annotated a shared set of 400 conversations, of which they identified 69 instances i.e., utterances of the first speaker of each conversation, containing opinion-asking questions (AngularJS: 10, C++: 10, OpenGL: 5, Python: 15, Clojurians: 12, Elm: 17). We computed Cohen's Kappa inter-rater agreement between the 2 judges, and found an agreement of 0.76, which is considered to be sufficient (> 0.6) [99], while the sample size of 400 conversations is sufficient to compute the agreement measure with high confidence [25]. The two judges then iteratively discussed and resolved all conflicts to create the final gold set.

Comparison Techniques: Researchers have used sentiment analysis techniques [106,177] and lexical patterns [159] to extract opinions from software-related documents. Thus, we selected two different approaches, i.e., pattern-matching approaches and sentiment analysis, as comparison techniques to ChatEO. We evaluated three well-known sentiment analysis techniques SentiStrength-SE [85], CoreNLP [158], and NLTK [84] with their default settings. Since opinions could have positive/negative polarities, for the purpose of evaluation, we consider a leading question in a conversation identified with

Table 6.4: Opinion-Asking Question Identification Results

Technique	**P**	**R**	**F**	**Example FP**	**Example TP**
SentiStrength-SE [85]	0.18	0.31	0.23	*...I'm having a weird issue with my ng-cli based angular app...Is there any potential issue, because my Model is called "Class" and/or the method is called "toClass"?*	*Anyone know a good cross platform GUI library that (preferably) supports CMake? I'd rather not use Qt if I don't have to because I don't want to use the qt MOC*
CoreNLP [158]	0.19	0.73	0.30	*why does '(read-string "07")' return '7', but '(read-string "08")' throws an exception?*	*any suggestions for cmake test running strategies? or how to organize tests expected to succeed or fail?*
NLTK [84]	0.18	0.78	0.30	*Hi, is there a way to expose a class and a function using c style declaration : void foo(MyClass bar)*	*Does anyone know of a good way to "sandbox" the loading of namespaces?*
DECA [159]	1.00	0.01	0.02	-	*..seems lambda with auto argument type provide separate templated methods per used type; am i right?..*
ChatEO	**0.87**	**0.49**	**0.62**	*can someone tell me why does this give an error ?*	*What is the best way in your opinion to convert input (file, XML, etc.) to PDF with precision to 1 mm?*

either positive or negative sentiment as opinion-asking. DECA [159] is a pattern-based technique that uses Natural Language Parsing to classify the content of development emails according to their purpose. We used their tool to investigate the use of their linguistic patterns to identify opinion-asking questions in developer chats. We do not compare with Huang et al.'s CNN-based classifier of intention categories [82], since they merged "opinion asking" with the "information seeking" category.

Results: Table 6.4 presents precision (P), recall (R), F-measure (F), and examples of False Positives (FP) and True Positives (TP) for ChatEO and our comparison techniques for opinion-asking question identification on our evaluation dataset.

ChatEO achieves a precision, recall, and F-measure of 0.87, 0.49 and 0.62, respectively. Results in Table 6.4 indicate that ChatEO achieves an overall better precision (except DECA) and F-measure, compared to all the comparison techniques. With high precision, when ChatEO identifies an opinion-asking question, the chance of it being correct is higher than that identified by other techniques. We aim for higher precision (with possible lower recall) in identifying opinion-asking questions, since that could potentially contribute to the next module of ChatEO i.e., extracting answers to opinion-asking questions.

Some of the opinion-asking instances that ChatEO wasn't able to recognize lacked presence of recurrent linguistic patterns such as *“”How does angular fit in with traditional MVC frameworks like .NET MVC and ruby on rails? Do people generally still use an MVC framework or just write a web api?”*. Some FNs also resulted from instances where the opinion-asking questions were continuation of a separate question such as *“Is there a canvas library where I can use getImageData to work with the typed array data? Or is this where I should use ports?”*.

We observe that the sentiment analysis tools show a high recall at the expense of low precision, with an exception of SentiStrengthSE, which exhibits lower values for both precision and recall. The *“Example FP”* column in Table 6.4 indicates that sentiment analysis tools are often unable to catch the nuances of SE-specific keywords such as ‘expose’, ‘exception’. Another example, *“What is the preferred way to distribute python programs?”*, which ChatEO is able to correctly identify as opinion-asking, is labelled as neutral by all the sentiment analysis tools. The same happens for the instance *“how do I filter items in a list when displaying them with ngFor? Should I use a filter/pipe or should I use ngIf in the template?”*. ChatEO is able to recognize that this is asking for opinions on what path to take among two options, while the sentiment analysis tools classify this as neutral. Note that this just indicates that these tools are limited in the context of identifying opinion-asking questions, but could be indeed useful for other tasks (e.g., assessing developer emotions).

DECA [159] identified only one instance to be opinion-asking, which is a true positive, hence the precision is 1.00. Apart from this, it was not able to classify any other instance as opinion-asking, hence the low recall (0.01). On analyzing DECA’s classification results, we observe that, out of 69 instances in the gold set, it could not assign any intention category to 17 instances. This is possibly due to the informal communication style in chats, which is considerably different than emails. Since an utterance could contain more than one sentence, DECA often assigned multiple categories (e.g., *information seeking, feature request, problem discovery*) to each instance. The most frequent intention category observed was *information seeking*. During the development phase, we explored additional linguistic patterns, but they yielded more

FPs. This is a limitation of using linguistic patterns, as they are restrictive when expressing words that have different meaning in different contexts.

ChatEO opinion-asking question identification significantly outperforms an existing pattern-based technique that was designed for emails [159], as well as sentiment analysis tools [84, 85, 158] in terms of F-measure.

RQ2. How effective is ChatEO in identifying answer(s) to opinion-asking questions in a developer conversation?

Gold Set Creation: Similar to RQ1, we recruited 2 human judges with (3+ years) experience in programming and in using both chat platforms (Slack and IRC), but no knowledge of our techniques. The gold set creation for answer annotation was conducted in two phases, as follows:

- Phase-1 (Annotation): The human judges were provided a set of conversations with annotation instructions as follows: *Mark each utterance in the conversation that provides information or advice (good or bad) that contributes to addressing the opinion-asking question in a way that is understandable/meaningful/interpretable when read as a standalone response to the marked opinion-asking question (i.e., the answer should provide information that is understandable without reading the entire conversation). Such utterances should not represent answer(s) to follow-up questions in a conversation. An answer to an opinion-asking question could also be a yes/no response. There could be more than one answer provided to the opinion-asking question in a conversation.*

- Phase-2 (Validation): The purpose of Phase-2 was two-fold: (1) measure validity of Phase-1 annotations, and (2) evaluate if an answer would match an opinion-asking question out of conversational context, such that the Q&A pair could be useful as part of a Q&A system. Therefore, for Phase-2 annotations, we ensured that the annotators read only the provided question and answers, and not the entire conversations from which they were extracted. The Phase-1 annotations from the first annotator were used to generate a set of question and answers, which were used for Phase-2 annotations by the second annotator, and vice-versa. For each utterance provided as an answer to an opinion-asking question, the annotators were asked to indicate ("yes/no") if the utterance represents an answer based on the guidelines in Phase-1. Additionally, if the annotation value was "no", the annotators were asked to state the reason.

The judges annotated a total of 2001 conversations, of which they identified a total of 2292 answers to opinion-asking questions (AngularJS: 1001, C++: 133, OpenGL: 263, Python: 165, Clojurians: 197, Elm: 533). We found that the first annotator considered 94.6% of annotations of the second annotator as valid, while the second annotator considered 96.2% annotations of the first annotator as valid. We also noticed that the majority of disagreements were due to the answer utterances containing incomplete or inadequate information to answer the marked opinion-asking question when removed from conversational context (e.g., "and then you can replace your calls to 'f' with 'logArgs2 f' without touching the function..."), and human annotation errors such as marking an utterance as an answer when it just points to other channels.
Comparison Techniques: Since we believe this is the first effort to automatically extract answers to opinion-asking questions from developer chats, we chose to evaluate against heuristic-based and feature-based machine learning classification as well as the original R-CNN deep learning-based technique on which we built ChatEO.

Heuristic-based (HE): Intuitively, the answer to an opinion-asking question might be found based on its location in the conversation, the relation between its content and the question, or the presence of sentiment in the answer. We investigated each of these possibilities separately and in combination.

- *Location:* Based on the intuition that a question might be answered immediately after it is asked during a conversation, we compare against the naive approach of identifying the next utterance after the leading opinion-asking question as an answer.

- *Content:* Traditional Q&A systems have often aimed to extract answers based on semantic matching between question and answers [168, 189]. Thus, to model content-based semantic relation between question and answer, we compare the average word embedding of the question and answer texts. Using our word embedding model described in 6.3.3.2, we extract utterances with considerable similarity (≥ 0.5) to the opinion-asking question as answers.

- *Sentiment:* Previous researchers have leveraged sentiment analysis to extract relevant answers in non-factoid Q&A systems [62, 95, 111, 196]. Thus, based on the intuition that the answer to an opinion-asking question might exhibit sentiment, we use CoreNLP [158] to extract utterances bearing sentiment (positive or negative) as answers. We explored other sentiment analysis tools (e.g., SentiStrength-SE [85], NLTK [84]); however, we do not discuss them, since they yielded inferior results.

Machine Learning-based (ML): We combine location, content, sentiment attributes as features of a machine learning (ML)-based classifier. We explored several popular ML algorithms (e.g., Support Vector Machines (SVM), Random Forest) using the Weka toolkit [76], and observed that they yielded nearly similar results. We report the results for SVM.

Deep Learning-based (DL): We present the results for both R-CNN and ChatEO implemented as follows.

- *RCNN:* We implemented R-CNN [204] for developer chats using open-source neural-network library Keras [42]. R-CNN used word embeddings pre-trained on their corpus. Similarly, we trained custom GloVe vectors on our chat corpus for our comparison.

- *ChatEO:* We also implemented ChatEO using Keras [42]. We used grid-search [20] to perform hyper-parameter tuning. First, to obtain sufficient semantic information at the utterance level, we use three convolution filters of size 2, 3, and 4, with 50 (twice the average length of an utterance) feature maps for each filter. The pool sizes of convolution are (2,1), (2,1), (3,1), respectively. Then, a BiLSTM layer with 400 units (200 for each direction) is used to capture the contextual information in a conversation. Finally, we use a linear layer with sigmoid activation function to predict the probability scores of binary classes (answer and non-answer). We use binary-cross-entropy as the loss function, and Adam optimization algorithm for gradient descent.

To avoid over-fitting, we apply a dropout [160] of 0.5 on the TextCNN embeddings, i.e., 50% units will be randomly omitted to prevent complex co-adaptations on the training data. Additionally, we use a recurrent dropout of 0.1 in the LSTM units. We also use early stopping [132], i.e., if the performance of the classifier did not improve for 10 epochs, the training process is stopped. ChatEO answer extraction takes approximately 36 minutes to train and test 2001 conversations on a system with 2.5 GHz Intel Core i5 processor and 8GB DDR3 RAM.

Evaluation Process: Evaluation of ChatEO and the comparison techniques was conducted using the evaluation dataset of 2000 conversations, described in Table 6.3. We created a test set of 400 conversations (adhering to commonly recognized train-test ratio of 80-20%). We ensured that it contains similar number of instances from each programming community: 140 (70*2) conversations from Angular JS and Python, 260 (65*4) conversations from C++, OpenGL, Clojurians, and Elm.

Results: Table 6.5 presents precision (P), recall (R), and F-measure (F) for ChatEO and our comparison techniques for automatic answer extraction on our held-out test set of 400 conversations, which contains a total of 499 answers. The best results for P, R, F across all techniques are highlighted in bold.

Overall, Table 6.5 shows that ChatEO achieves the highest overall precision, recall, and F-Measure of all techniques, with 0.77, 0.59, and 0.67, respectively. Overall, ChatEO identified 370 answers in the test set, out of which 285 are true positives.

Table 6.5: Answer Extraction Results on Held-out Test Set
HE: Heuristic-based, ML: Machine Learning-based, DL: Deep Learning-based

Technique		**AngularJS**			**C++**			**OpenGL**			**Python**			**Clojurians**			**Elm**			**Overall**		
		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
HE	Location(L)	0.66	0.47	0.55	0.68	**0.77**	0.72	0.69	**0.74**	0.71	0.59	0.62	0.60	0.52	0.40	0.46	0.65	0.45	0.53	0.63	0.55	0.59
	Content(C)	0.03	0.06	0.04	0.05	0.14	0.07	0.06	0.18	0.09	0.11	0.18	0.14	0.09	0.11	0.10	0.14	0.15	0.14	0.07	0.13	0.09
	Sentiment(S)	0.06	0.16	0.09	0.05	0.23	0.08	0.09	0.31	0.13	0.11	0.24	0.15	0.07	0.14	0.09	0.10	0.18	0.13	0.07	0.20	0.11
ML	L+C+S	0.70	0.47	0.56	0.78	**0.77**	**0.77**	0.70	0.73	0.71	0.62	0.62	0.62	0.60	0.41	0.49	0.75	0.47	0.58	0.69	0.55	0.61
DL	R-CNN	0.70	0.46	0.55	0.78	0.74	0.76	0.72	0.70	0.71	0.64	0.61	0.62	0.62	0.38	0.48	0.73	0.39	0.51	0.70	0.52	0.60
	ChatEO	**0.80**	**0.53**	**0.64**	**0.81**	0.72	**0.77**	**0.77**	0.68	**0.72**	**0.76**	**0.65**	**0.70**	**0.66**	**0.42**	**0.51**	**0.81**	**0.56**	**0.66**	**0.77**	**0.59**	**0.67**

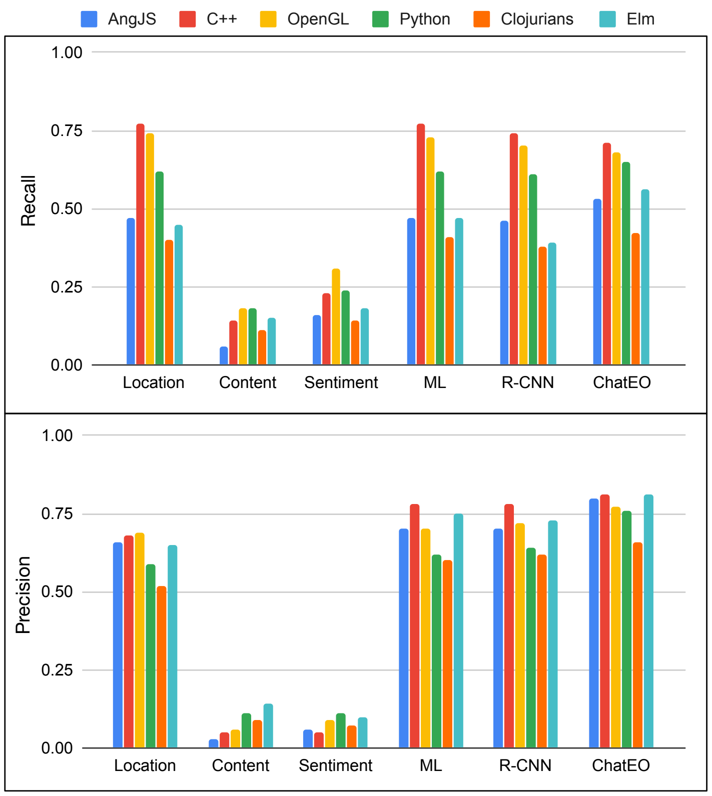


Figure 6.2: Evaluation Measures of Answer Extraction

R-CNN and the ML-based classifier perform next best and similar to each other in precision and F-measure. The better performance of ChatEO suggests that capturing contextual information through BiLSTM can benefit answer extraction in chat messages, and that using a domain-specific word embedding model (trained on software-related texts) accompanied with hyper-parameter tuning, is essential to adapting deep learning models for software-related applications. Figure 6.2 shows that the performance of ChatEO is consistent (76-81% precision) across all communities, except Clojure. One possible reason for this is, answers are often provided in this chat community in the form of code snippets along with little to no natural language text, which makes it difficult for ChatEO to model the semantic links.

ChatEO's overall recall of 0.59 indicates that it is difficult to identify all relevant answers in chats even with complex dialog modeling. In fact, the recall of ChatEO is lower than the heuristic-based technique *location* for C++ and OpenGL. Upon deeper analysis, we observed that these two communities contain less answers (one answer per opinion-asking question on average) compared to the other communities, and the answer often resides in the utterance occurring immediately after the first speaker.

The *location* heuristic exhibits significantly better performance than content or sentiment heuristics. Of the 400 conversations, 278 have at least one answer occurring immediately after the utterances of the first speaker. Neither *content* or *sentiment* is a strong indicator of answers, with precision of 0.07 and F-measure of 0.09 and 0.11, respectively. These heuristics cannot distinguish the intent of a response. Consider, "***Q**: Hi, Clojure intermediate here. What is the best way to read big endian binary file?; **R:** do you need to make something that parses the binary contents, does it suffice to just get the bytes in an array?*". Both *content* and *sentiment* marked this response as an answer, without being able to understand that this is a follow-up question.

Combining the heuristics as features to SVM, the precision (and thus F-Measure) improved slightly over the *location* heuristic with .06 increase in precision and .02 in F-measure. As expected, *location* as a feature shows the highest information gain. We investigated several classifier parameters (e.g., kernal and regularization parameters), but 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). Since our dataset is imbalanced, with considerably low ratio of *answers* to other utterances, we explored over-sampling (SMOTE) techniques. No significant improvements occurred. ML-based classification may be improved with more features and feature engineering.

ChatEO answer extraction shows improvement over heuristics-based, ML-based, and existing deep learning-based [204] techniques in terms of precision, recall, and F-measure.

6.5 Threats to Validity

Construct Validity: A threat to construct validity might arise from the manual annotations for creating the gold sets. To limit this threat, we ensured that our annotators have considerable experience in programming and in using both chat platforms and that they followed a consistent annotation procedure piloted in advance. We also observed high values of Cohen's Kappa coefficient, which measures the inter-rater agreement for opinion-asking questions. For answer annotations, we conducted a two-phase annotation procedure to ensure the validity of the selected answers.

Internal Validity: Errors in the automatically disentangled conversations could pose a threat to internal validity affecting misclassification. We mitigated this threat by humans, without knowledge of our techniques, manually discarding poorly disentangled conversations from our dataset. In all stages of the pipeline of ChatEO, we aimed for higher precision over recall as the quality of information is more important than missing instances; chat datasets are large with many opinions so our achieved recall is sufficient to extract a significant number of opinion Q&A. Other potential threats could be related to evaluation bias or errors in our scripts. To reduce these threats, we ensured that the instances in our development set do not overlap with our train or test sets. We also wrote unit tests and performed code reviews.

External Validity: To ensure generalizability of our approach, we selected the subjects of our study from the two most popular software developer chat communities, Slack and IRC. We selected statistically representative samples from six active communities which represent a broad set of topics related to each programming language. However, our study's results may not transfer to other chat platforms or developer communications. Scaling to larger datasets might also lead to different evaluation results. Our technique of identifying opinion-asking questions could be made more generalizable by augmenting the set of identified patterns and vocabulary terms.

6.6 Related Work

Mining Opinions in SE. In addition to the related work discussed in Section 6.2, significant work has focused on mining opinions from developer forums. Uddin and Khomh [177] designed Opiner, which uses keyword-matching along with a customized Sentiment Orientation algorithm to summarize API reviews. Lin et al. [106] used patterns to identify and classify opinions on APIs from Stack Overflow. Zhang et al. [199] identifies negative opinions about APIs from forums. Huang et al. [83] proposed an automatic approach to distill and aggregate comparative opinions of comparable technologies from Q&A websites. Ren et al. [142] discovered and summarized controversial (criticized) answers in Stack Overflow, based on judgment, sentiment and opinion. Novielli et al. [120, 121] investigated the role of affective lexicon on the questions posted in Stack Overflow.

Researchers have also analyzed opinions in developer emails, commit logs, and app reviews. Xiong et al. [192] studied assumptions in OSS development mailing lists. Sinha et al. [156] analyzed developer sentiment in Github commit logs. Opinions in app reviews [28, 54, 71, 75] have been mined to help app developers gather information about user requirements, ideas for improvements, and user sentiments about specific features. To the best of our knowledge, our work is the first to extract opinion Q&A from developer chats.

Extracting Q&A from Online Communications. Outside the SE domain, researchers have proposed techniques to identify Q&A pairs in online communications (e.g., Yahoo Answers). Shrestha et al. [154] used machine learning approaches to automatically detect Q&A pairs in emails. Cong et al. [47] detected Q&A pairs from forum threads by using Sequential Pattern Mining to detect questions, and a graph-based propagation method to detect answers in the same thread.

Recently, researchers have focused on answer selection, a major subtask of Q&A extraction, which aims to select the most relevant answers from a candidate answer set. Typical approaches for answer selection model the semantic matching between question and answers [147, 150, 168, 202]. These approaches have the advantage of sharing parameters, thus making the model smaller and easier to train. However, they often fail to capture the semantic correlations embedded in the response sequence of a question. To overcome such drawbacks, Zhou et al. [204] designed a recurrent architecture that models the semantic relations between successive responses, as well as the question and answer. Xiang et al. [191] investigated an attention mechanism and context modeling to aid the learning of deterministic information for answer selection. Wang et al. [184] proposed a bilateral multi-perspective matching model in which Q&A pairs are matched on multiple levels of granularity at each time-step. Our model belongs to the same framework which captures the contextual information of conversations in extracting answers from developer chats.

Most of the these techniques for Q&A extraction were designed for general online communications and not specifically for software forums. Gottipati et al. [70] used Hidden Markov Models to infer semantic tags (e.g., question, answer, clarifying question) of posts in the software forum threads. Henß et al. [79] used topic modeling and text similarity measures to automatically extract FAQs from software development discussions (mailing lists, online forums).

6.7 Summary

In this chapter, we present and evaluate ChatEO, which automatically identifies opinion-asking questions from chats using a pattern-based technique, and extracts participants' answers using a deep learning-based architecture. This research provides a significant contribution to using software developers' public chat forums for building opinion Q&A systems, a specialized instance of virtual assistants and chatbots for software engineers. ChatEO opinion-asking question identification significantly outperforms existing sentiment analysis tools and a pattern-based technique that was designed for emails [159]. ChatEO answer extraction shows improvement over heuristics-based, ML-based, and an existing deep learning-based technique designed for CQA [204]. Our replication package can be used to verify our results [5]. The Q&A pairs extracted using ChatEO could be leveraged to generate FAQs, provide tool support for recommendation systems, and in understanding developer behavior and collaboration (asking and sharing opinions).

[5] https://tinyurl.com/y3qth6s3

Chapter 7
CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Integrated development environments today include sophisticated program modeling and analyses behind the scenes to support the developer in navigating, understanding, and modifying their code. While much can be learned from results of static and dynamic analysis of their source code, developers also look to others for advice and learning. As software development teams are more globally distributed and the open source community has grown, developers rely increasingly on chat communications for help they might have previously obtained through in-person conversations.

The overall goal of my research is to mine information from software developer chats and extract actionable nuggets of knowledge, that can be integrated into the development of new models to enhance software maintenance tools. My dissertation work addresses this goal by first gathering knowledge about potential of developer chats as a mining source for software maintenance and evolution tools, and second by developing several techniques for supporting the software engineering process. This chapter summarizes the contributions of this dissertation and describes opportunities for future work.

7.1 Summary of Contributions

My thesis shows that software developers communications through online chat forums are a valuable resource to mine and valuable knowledge can be automatically mined from this resource towards improving and building new tools to support software engineers. The main contributions of this dissertation are as follows:

1. **An exploratory study to investigate the potential of developer chats as a mining source for software maintenance and evolution tools.** Our

results indicate the prevalence of useful information in software-related chats, including API mentions and code snippets with descriptions, and several hurdles that need to be overcome to automate mining that information. The availability of this information in chats may lead to new mining opportunities for software maintenance tools, such as, supporting IDE recommendation [9, 13, 48, 53, 130, 138], learning and recommendation of APIs [38, 137, 183], automatic generation of comments for source code [134, 186], and in building thesauri and knowledge graphs of software-specific terms and commonly-used terms in software engineering [40, 169].

2. **A supervised machine learning technique customized to automatically disentangle conversations in software developer chats.** The model with our enhancements produced a micro-averaged F-measure of 0.80. This work enables researchers to process additional Slack developer chat transcripts and disentangle them into conversations, for further research purposes.

3. **A public knowledge archive of conversations in developer chat communications, suitable for research beyond this project.** Specifically, we present a dataset of software-related Q&A chat conversations, curated for two years from three open Slack communities (python, clojure, elm). Our dataset consists of 38,955 conversations, 437,893 utterances, contributed by 12,171 users. The data is openly available to be downloaded for further reuse by the community.

4. **A data-driven approach to identify properties of post hoc quality information** or quality metrics for developer chat communications. This work takes the first step to assess quality of information in developer chats, and lays the foundation for using chats as a mining source for building task-based software applications such as API recommendation systems, virtual assistants for programming/debugging help, and FAQ generation.

5. **An approach based on text processing and machine learning to automatically identify quality of developer chats for post hoc use.** Evaluation shows that our approach could achieve a reasonable performance of 0.82 precision and a higher recall of 0.90. This technique can be applied as a quality filter mechanism by the mining tools 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.

6. **A linguistic pattern-based technique to automatically identify opinion-asking questions in chat conversations.** Evaluation shows that this approach achieves a precision and recall of 0.87 and 0.49 recall respectively, and significantly outperforms existing sentiment analysis tools and a pattern-based technique that was designed for emails [159]. Mining of opinions from developer chats could help in reducing developers' effort of manual searches on the web and facilitate information gathering, increasing developer productivity, improving code efficiency [177], and building better recommendation systems [106].

7. **A deep learning-based model to automatically extract participants' answers to opinion-asking questions in public Q&A chats.** This answer extraction model could potentially be leveraged to extract answers to other types of questions in developer chats. Our deep learning approach customized to the software domain achieves a precision and recall of 0.77 and 0.59 respectively, and outperforms heuristics-based, machine-learning-based and deep learning for answer extraction in community question answering. The opinion Q&A corpus generated from developer chats by our technique can be used to build virtual assistants for software engineers, automatic generation of FAQs, provide tool support for building recommendation systems, and in understanding developer behavior and collaboration in sharing opinions.

7.2 Future Work

7.2.1 Extensions

The immediate future work includes:

- **Expanding the disentangled chat datasets.** We released a dataset of disentangled software-related chat conversations from Slack, which is one of the most popular software developer chat communities. As a next step, this dataset could be extended to include conversations from other chat platforms such as Gitter, and investigate the comprehensiveness and generalizability of our disentanglement approach.

- **Improving effectiveness of post hoc quality identification.** The effectiveness of our machine-learning based technique to identify post hoc quality information in chats could be improved by: refining the techniques of question identification (to handle more complex forms such as indirect questions), and assessing the quality of the answers by understanding the conversation context.

- **Designing chatbots to assess quality of developer communications in chat platforms.** We envision improving the overall quality of developer chat 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.

- **Improving effectiveness of ChatEO.** The immediate next steps for improving the effectiveness of ChatEO could focus on investigating machine learning-based techniques for opinion-asking question identification, and attention-based LSTM network for answer extraction.

7.2.2 New Directions

The more future directions include:

1. For several years, researchers have been developing techniques to mine information from Stack Overflow that is related and useful in another context, thus, in some way, establishing links between different contexts. The mined information is used to provide recommendation in Integrated Development Environments (IDE) for programming errors and exceptions [10, 49, 136], link relevant discussions to any source code based on context [14, 26, 131], and augment API documentations [91, 167, 173]. To our knowledge, no research has focused on mining similar information and establishing links from chat communities.

   Borrowing from data triangulation used by qualitative researchers, we envision a system where software developers' social communication channels serve as the multiple sources of evidence to establish quality of information in each channel. Information in one social communication channel that complements or contradicts information in another channel is identified and used to provide feedback within the social communities. The first step towards investigating that research direction would be to conduct a case study to explore this potential by comparing information in Q&A chats with another Q&A-based software artifact, such as Stack Overflow. This study would help understand how information derived from informal conversations overlaps with and differs from information embedded in similar topics discussed on a Q&A forum. The comparison will provide insight into how software artifacts from different sources complement each other, how information from different social communication channels contradict each other, and how they duplicate each other further confirming good quality shared information.

2. Due to its increased popularity, chat is becoming a popular media to disseminate information between software engineers across the globe. Lin et al. [104] have shown that developers use Slack to discover news/information on technological trends. Thus developer chat communications could be studied to identify 'hot' topics of discussion in a programming community [149], and understand common

challenges and misconceptions among developers [16]. The results of these studies would provide guidance to future research in developing software support and maintenance tools.

3. The widespread use of chat communication platforms such as Slack and IRC provides a thriving opportunity to build new conversation-based tools and integrations, such as chatbots. Bots have become increasingly prominent due to the ease of their integration with communication tools and accessibility to various APIs and data sources [100]. Virtual conversational assistants designed specifically for software engineers could have a huge impact on the time it takes for software engineers to get help. Current research efforts are focusing on virtual assistants that support specific software development tasks such as bug repair [188] and pair programming [56].

   One of the major constraints for designing virtual assistants for software engineers is the lack of specialized datasets that are needed to design and train a virtual assistant. To effectively collect data needed to design a task-specific virtual assistant, researchers [56, 188] conducted Wizard of Oz experiments to collect conversations between humans and (simulated) virtual assistants. These experiments could be expensive, often requiring several participants and extensive planning. Conversations in online chat services such as IRC, Slack, and Gitter could potentially provide rich data for building virtual assistants for software engineers, but little research has explored this potential. Thus, one of the potential future research directions is to explore the feasibility of building virtual assistants for software engineers by leveraging the conversations available in online developer chat services such as IRC, Slack, and Gitter.

BIBLIOGRAPHY

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

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

[3] P. H. Adams and C. H. Martell. Topic detection and extraction in chat. In *2008 IEEE International Conference on Semantic Computing*, pages 581–588, Aug 2008.

[4] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. Finding high-quality content in social media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 183–194, New York, NY, USA, 2008. ACM.

[5] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining Duplicate Questions in Stack Overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 402–412, New York, NY, USA, 2016. ACM.

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

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

[8] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge. How do developers discuss rationale? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, volume 00, pages 357–369, March 2018.

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

[10] V. Amintabar, A. Heydarnoori, and M. Ghafari. ExceptionTracer: A Solution Recommender for Exceptions in an Integrated Development Environment. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 299–302, May 2015.

[11] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K. Roy, and Kevin A. Schneider. Answering questions about unanswered questions of stack overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 97–100, Piscataway, NJ, USA, 2013. IEEE Press.

[12] A. Bacchelli, M. D'Ambros, and M. Lanza. Extracting source code from e-mails. In *Proc. 18th Int'l Conf. Program Comprehension*, pages 24–33, June 2010.

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

[14] Alberto Bacchelli, Luca Ponzanelli, and Michele Lanza. Harnessing Stack Overflow for the IDE. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, pages 26–30, Piscataway, NJ, USA, 2012. IEEE Press.

[15] A. Sajedi Badashian, A. Hindle, and E. Stroulia. Crowdsourced bug triaging. In *Proc. IEEE Int'l Conf. on Software Maintenance and Evolution*, pages 506–510, September 2015.

[16] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 112?121, New York, NY, USA, 2014. Association for Computing Machinery.

[17] Alexandra Balahur, Ester Boldrini, Andrés Montoyo, and Patricio Martínez-Barco. Opinion question answering: Towards a unified approach. In *ECAI*, 2010.

[18] Antoaneta Baltadzhieva and Grzegorz Chrupala. Question Quality in Community Question Answering Forums: A Survey. *SIGKDD Explor. Newsl.*, 17(1):8–13, September 2015.

[19] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. Modern code reviews in open-source projects: Which problems do they fix? In *Proc. 11th Working Conf. Mining Software Repositories*, pages 202–211, 2014.

[20] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281?305, February 2012.

[21] S. Bird, E. Loper, and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[22] Steven Bird. 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*, 2002.

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

[24] Wesley Brants, Bonita Sharif, and Alexander Serebrenik. Assessing the meaning of emojis for emotional awareness - a pilot study. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 419?423, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Mohamad Adam Bujang and Nurakmal Baharum. 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.

[26] Brock Angus Campbell and Christoph Treude. Nlp2code: Code snippet content assist via natural language tasks. In *Proceedings of the 2017 International Conference on Software Maintenance and Evolution*, 2017.

[27] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 44:1–44:11, New York, NY, USA, 2012. ACM.

[28] L. V. G. Carreño and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 582–591, 2013.

[29] P. Chatterjee, K. Damevski, N.A. Kraft, and L. Pollock. Automatically Identifying the Quality of Developer Chats for Post Hoc Use. In *Transactions on Software Engineering and Methodology (TOSEM)*, TOSEM '20, 2020.

[30] P. Chatterjee, K. Damevski, and L. Pollock. Automatic Extraction of Opinion-based Q&A from Online Developer Chats. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, 2021.

[31] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N.A. Kraft. 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)*, May 2019.

[32] P. Chatterjee, B. Gause, H. Hedinger, and L. Pollock. Extracting code segments and their descriptions from research articles. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 91–101, May 2017.

[33] Preetha Chatterjee. Extracting Archival-Quality Information from Software-Related Chats. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE '20, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Preetha Chatterjee, Kostadin Damevski, Nicholas A. Kraft, and Lori Pollock. Software-related slack chats with disentangled conversations. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, pages 588–592, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Preetha Chatterjee, Minji Kong, and Lori Pollock. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software*, 159:110454, 2020.

[36] Preetha Chatterjee, Manziba Akanda Nishi, Kostadin Damevski, Vinay Augustine, Lori Pollock, and Nicholas A. Kraft. What information about code snippets is available in different software-related documents? an exploratory study. In *Proc. 24th IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering*, February 2017.

[37] C. Chen, S. Gao, and Z. Xing. Mining Analogical Libraries in Q A Discussions – Incorporating Relational and Categorical Knowledge into Word Embedding. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 338–348, March 2016.

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

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

[40] Chunyang Chen, Zhenchang Xing, and Ximing Wang. Unsupervised software-specific morphological forms inference from informal discussions. In *Proc. 39th Int'l Conf. on Software Engineering*, pages 450–461, 2017.

[41] Chunyang Chen, Zhenchang Xing, and Ximing Wang. Unsupervised Software-specific Morphological Forms Inference from Informal Discussions. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 450–461, Piscataway, NJ, USA, 2017. IEEE Press.

[42] Francois Chollet et al. Keras, 2015.

[43] S. A. Chowdhury and A. Hindle. Mining StackOverflow to Filter Out Off-Topic IRC Discussion. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 422–425, May 2015.

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

[45] Per Christensson. Techterms.com. https://techterms.com/category/software, Accessed: 2020-09-02.

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

[47] Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yueheng Sun. Finding question-answer pairs from online forums. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 467–474, New York, NY, USA, 2008. ACM.

[48] Joel Cordeiro, Bruno Antunes, and Paulo Gomes. Context-based recommendation to support problem solving in software development. In *Proc. 3rd Int'l Wksp. on Recommendation Systems for Software Engineering*, pages 85–89, May 2012.

[49] Joel Cordeiro, Bruno Antunes, and Paulo Gomes. Context-based Recommendation to Support Problem Solving in Software Development. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, pages 85–89, Piscataway, NJ, USA, 2012. IEEE Press.

[50] Denzil Correa and Ashish Sureka. Fit or Unfit: Analysis and Prediction of 'Closed Questions' on Stack Overflow. In *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, pages 201–212, New York, NY, USA, 2013. ACM.

[51] Denzil Correa and Ashish Sureka. Chaff from the Wheat: Characterization and Modeling of Deleted Questions on Stack Overflow. In *Proceedings of the 23rd*

International Conference on World Wide Web, WWW '14, pages 631–642, New York, NY, USA, 2014. ACM.

[52] J. Shane Culpepper, Fernando Diaz, and Mark D. Smucker. Research frontiers in information retrieval: Report from the third strategic workshop on information retrieval in lorne (swirl 2018). *SIGIR Forum*, 52(1):34?90, August 2018.

[53] Lucas B. L. de Souza, Eduardo C. Campos, and Marcelo de A. Maia. Ranking crowd knowledge to assist software development. In *Proc. 22nd Int'l Conf. on Program Comprehension*, pages 72–82, May 2014.

[54] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 499–510, New York, NY, USA, 2016. ACM.

[55] Maarten Duijn, Adam Kučera, and Alberto Bacchelli. Quality Questions Need Quality Code: Classifying Code Fragments on Stack Overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 410–413, Piscataway, NJ, USA, 2015. IEEE Press.

[56] Zachary Eberhart, Aakash Bansal, and Collin McMillan. The apiza corpus: Api usage dialogues with a simulated virtual assistant, 2020.

[57] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. Confusion detection in code reviews. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 549–553, 2017.

[58] Osama. Ehsan, Safwat Hassan, Mariam E. Mezouar, and Ying Zou. An empirical study of developer discussions in the gitter platform. *Transactions on Software Engineering and Methodology (TOSEM)*, July 2020.

[59] Margaret S. Elliott and Walt Scacchi. 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*, GROUP '03, pages 21–30, New York, NY, USA, 2003. ACM.

[60] M. Elsner and E. Charniak. Disentangling chat. *Computational Linguistics*, 36(3):389–409, 2010.

[61] Micha Elsner and Eugene Charniak. You talking to me? a corpus and algorithm for conversation disentanglement. In *Proc. Association of Computational Linguistics: Human Language Technology*, pages 834–842, 2008.

[62] F. Eskandari, H. Shayestehmanesh, and S. Hashemi. Predicting best answer using sentiment analysis in community question answering systems. In *2015 Signal Processing and Intelligent Systems Conference (SPIS)*, pages 53–57, 2015.

[63] Stack Exchange. Stack exchange data dump. https://archive.org/details/stackexchange, 2017. [Online; accessed 2017-08-27].

[64] Pnina Fichman. A comparative assessment of answer quality on four question answering sites. *Journal of Information Science*, 37(5):476–486, 2011.

[65] Rudolph Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):p221 – 233, June 1948.

[66] Stack Overflow for Teams. https://slack.com/apps/ABDN6MGG1-stack-overflow-for-teams, 2018.

[67] D. Ford, J. Smith, P.J. Guo, and C. Parnin. Paradise unplugged: Identifying barriers for female participation on Stack Overflow. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 846–857, 2016.

[68] Neelamadhav Gantayat, Pankaj Dhoolia, Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 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*, MSR '15, pages 406–409, Piscataway, NJ, USA, 2015. IEEE Press.

[69] Qing Gao, Hansheng Zhang, Jie Wang, Yingfei Xiong, Lu Zhang, and Hong Mei. Fixing recurring crash bugs via analyzing q&amp;a sites (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 307–318, Washington, DC, USA, 2015. IEEE Computer Society.

[70] Swapna Gottipati, David Lo, and Jing Jiang. Finding relevant answers in software forums. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 323–332, Washington, DC, USA, 2011. IEEE Computer Society.

[71] M. Goul, O. Marjanovic, S. Baxley, and K. Vizecky. Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering. In *2012 45th Hawaii International Conference on System Sciences*, pages 4168–4177, 2012.

[72] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

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

[74] R. Gunning. The Technique of Clear Writing. *McGraw-Hill*, 1952.

[75] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162, 2014.

[76] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[77] F. Maxwell Harper, Daniel Moy, and Joseph A. Konstan. Facts or friends?: Distinguishing informational and conversational questions in social q&#38;a sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 759–768, New York, NY, USA, 2009. ACM.

[78] F. Maxwell Harper, Daphne Raban, Sheizaf Rafaeli, and Joseph A. Konstan. Predictors of answer quality in online q&amp;a sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 865–874, New York, NY, USA, 2008. ACM.

[79] Stefan Henß, Martin Monperrus, and Mira Mezini. Semi-automatically extracting faqs to improve accessibility of software development knowledge. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 793–803, Piscataway, NJ, USA, 2012. IEEE Press.

[80] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735?1780, November 1997.

[81] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. Automatically Mining Software-Based, Semantically-Similar Words from Comment-Code Mappings. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 377–386, May 2013.

[82] Q. Huang, X. Xia, D. Lo, and G. C. Murphy. Automating intention mining. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[83] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. Tell them apart: Distilling technology differences from crowd-scale comparison discussions. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 214?224, New York, NY, USA, 2018. Association for Computing Machinery.

[84] Clayton J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *ICWSM*, 2014.

[85] Md Rakibul Islam and Minhaz F. Zibran. Leveraging automated sentiment analysis in software engineering. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR ?17, page 203?214. IEEE Press, 2017.

[86] I. Itkin, A. Novikov, and R. Yavorskiy. Development of intelligent virtual assistant for software testing team. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 126–129, 2019.

[87] He Jiang, Jingxuan Zhang, Zhilei Ren, and Tao Zhang. An unsupervised approach for discovering relevant tutorial fragments for apis. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 38–48, Piscataway, NJ, USA, 2017. IEEE Press.

[88] Jyun-Yu Jiang, Francine Chen, Yan-Ying Chen, and Wei Wang. Learning to disentangle interleaved conversational threads with a Siamese hierarchical network and similarity ranking. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1812–1822, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[89] Shafiq Joty, Alberto Barrón-Cedeño, Giovanni Da San Martino, Simone Filice, Lluís Màrquez, Alessandro Moschitti, and Preslav Nakov. Global thread-level inference for comment classification in community question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 573–578, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[90] J. Kim, S. Lee, S.-W. Hwang, and S. Kim. Enriching Documents with Examples: A Corpus Mining Approach. *ACM Trans. Inf. Syst.*, 31(1):1:1–1:27, January 2013.

[91] Jinhan Kim, Sanghoon Lee, Seung-Won Hwang, and Sunghun Kim. Enriching Documents with Examples: A Corpus Mining Approach. *ACM Trans. Inf. Syst.*, 31(1):1:1–1:27, January 2013.

[92] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[93] Vanessa Kitzie, Erik Choi, and Chirag Shah. 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*, ASIST '13, pages 5:1–5:10, Silver Springs, MD, USA, 2013. American Society for Information Science.

[94] Vanessa Kitzie, Erik Choi, and Chirag Shah. 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*, ASIST '13, pages 107:1–107:4, Silver Springs, MD, USA, 2013. American Society for Information Science.

[95] Lun-Wei Ku, Yu-Ting Liang, and Hsin-Hsi Chen. Question analysis and answer passage retrieval for opinion question answering systems. In *International Journal of Computational Linguistics & Chinese Language Processing, Volume 13, Number 3, September 2008: Special Issue on Selected Papers from ROCLING XIX*, pages 307–326, September 2008.

[96] Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph J. Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros Polymenakos, and Walter S. Lasecki. A large-scale corpus for conversation disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3846–3856, July 2019.

[97] Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph J. Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros C Polymenakos, and Walter Lasecki. A large-scale corpus for conversation disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3846–3856, Florence, Italy, July 2019. Association for Computational Linguistics.

[98] S. K. Kuttal, J. Myers, S. Gurka, D. Magar, D. Piorkowski, and R. Bellamy. Towards designing conversational agents for pair programming: Accounting for creativity strategies and conversational styles. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–11, 2020.

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

[100] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. How software developers mitigate collaboration friction with chatbots. In *Proc. 20th ACM Conf. on Computer-Supported Cooperative Work and Social Computing*, 2017.

[101] Carlene Lebeuf, Margaret-Anne D. Storey, and Alexey Zagalsky. How Software Developers Mitigate Collaboration Friction with Chatbots. *CoRR*, abs/1702.07011, 2017.

[102] Baichuan Li, Tan Jin, Michael R. Lyu, Irwin King, and Barley Mak. Analyzing and predicting question quality in community question answering services. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 775–782, New York, NY, USA, 2012. ACM.

[103] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 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*, CSCW '16 Companion, pages 333–336, New York, NY, USA, 2016. ACM.

[104] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. Why developers are slacking off: Understanding how software teams use Slack. In *Proc. 19th ACM Conf. on Computer Supported Cooperative Work and Social Computing Companion*, 2016.

[105] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. Pattern-based mining of opinions in q&a websites. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 548?559. IEEE Press, 2019.

[106] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. Pattern-based mining of opinions in q&a websites. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 548?559. IEEE Press, 2019.

[107] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 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*, pages 285–294, Prague, Czech Republic, September 2015. Association for Computational Linguistics.

[108] Ryan Thomas Lowe, Nissan Pow, Iulian Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *D&D*, 8:31–65, 2017.

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

[110] Shikib Mehri and Giuseppe Carenini. Chat disentanglement: Identifying semantic reply relationships with random forests and recurrent neural networks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 615–623, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[111] S. Moghaddam and M. Ester. Aqa: Aspect-based opinion question answering. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 89–96, 2011.

[112] Piero Molino, Luca Maria Aiello, and Pasquale Lops. Social question answering: Textual, user, and network features for best answer prediction. *ACM Trans. Inf. Syst.*, 35(1):4:1–4:40, September 2016.

[113] S. Morgan. How are programming questions from women received on Stack Overflow? a case study of peer parity. In *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, pages 39–41, 2017.

[114] Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. Exploiting syntactic and shallow semantic kernels for question answer classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 776–783, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

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

[116] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[117] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9(5):771–783, Sept 2016.

[118] N. Novielli, F. Calefato, and F. Lanubile. Towards discovering the role of emotions in Stack Overflow. In *Proceedings of the 6th International Workshop on Social Software Engineering*, pages 33–36, 2014.

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

[120] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. Towards Discovering the Role of Emotions in Stack Overflow. In *Proceedings of the 6th International Workshop on Social Software Engineering*, SSE 2014, pages 33–36, New York, NY, USA, 2014. ACM.

[121] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. The Challenges of Sentiment Detection in the Social Programmer Ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering*, SSE 2015, pages 33–40, New York, NY, USA, 2015. ACM.

[122] Elahe Paikari and André van der Hoek. A framework for understanding chatbots and their future. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '18, pages 13–16, New York, NY, USA, 2018. ACM.

[123] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 63–72, 2012.

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

[125] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. ARdoc: App Reviews Development Oriented Classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 1023–1027, New York, NY, USA, 2016. ACM.

[126] Peng Jiang, Hongping Fu, Chunxia Zhang, and Zhendong Niu. A framework for opinion question answering. In *2010 6th International Conference on Advanced Information Management and Service (IMS)*, pages 424–427, 2010.

[127] Gayane Petrosyan, Martin P. Robillard, and Renato De Mori. Discovering information explaining api types using text classification. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 869–879, Piscataway, NJ, USA, 2015. IEEE Press.

[128] Gayane Petrosyan, Martin P. Robillard, and Renato De Mori. Discovering information explaining api types using text classification. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 869–879, Piscataway, NJ, USA, 2015. IEEE Press.

[129] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving Low Quality Stack Overflow Post Detection. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 541–544, Sept 2014.

[130] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In *Proc. 11th Working Conf. on Mining Software Repositories*, pages 102–111, May 2014.

[131] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter. In *Proceedings of the 11th Working Conference on Mining*

Software Repositories, MSR 2014, pages 102–111, New York, NY, USA, 2014. ACM.

[132] Lutz Prechelt. Early stopping - but when? In *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pages 55–69. Springer-Verlag, 1997.

[133] J. R. Quinlan. Induction of decision trees. *MACH. LEARN*, 1:81–106, 1986.

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

[135] M. M. Rahman, C. K. Roy, and D. Lo. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 349–359, March 2016.

[136] M. M. Rahman, S. Yeasmin, and C. K. Roy. Towards a Context-Aware IDE-Based Meta Search Engine for Recommendation about Programming Errors and Exceptions. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 194–203, Feb 2014.

[137] M.M. Rahman, C.K. Roy, and D. Lo. RACK: Automatic API recommendation using crowdsourced knowledge. In *Proc. IEEE 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering*, pages 349–359, March 2016.

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

[139] Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, Eric King, Kate Bland, Amanda Wartick, Yi Pan, Han Song, Sk Jayadevan, Gene Hwang, and Art Pettigrue. Conversational ai: The science behind the alexa prize, 2018.

[140] Nikitha Rao, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, and Nachiappan Nagappan. Analyzing web search behavior for software engineering tasks, 2020.

[141] Manasa Rath, Long T. Le, and Chirag Shah. Discerning the quality of questions in educational q&#38;ausing textual features. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*, CHIIR '17, pages 329–332, New York, NY, USA, 2017. ACM.

[142] Xiaoxue Ren, Zhenchang Xing, Xin Xia, Guoqiang Li, and Jianling Sun. Discovering, explaining and summarizing controversial discussions in community q&a sites. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE ?19, page 151?162. IEEE Press, 2019.

[143] Ricardo Romero, Esteban Parra, and Sonia Haiduc. Experiences building an answer bot for gitter. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW?20)*, 2020.

[144] L. Richardson. Beautiful soup. https://www.crummy.com/software/BeautifulSoup/, 2017. [Online; accessed 2017-05-07].

[145] Matthieu Riou, Soufian Salim, and Nicolás Borrego Hernández. Using discursive information to disentangle french language chat. 2015.

[146] Ricardo Romero, Sonia Haiduc, and Esteban Parra. Experiences building an answer bot for gitter. In *Proceedings of the 2nd International Workshop on Bots in Software Engineering*, BotSE '20, 2020.

[147] Andreas Rücklé and Iryna Gurevych. Representation learning for answer selection with LSTM-based importance weighting. In *IWCS 2017 — 12th International Conference on Computational Semantics — Short papers*, 2017.

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

[149] A. Sharma, Y. Tian, and D. Lo. What's hot in software engineering twitter space? In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 541–545, Sep. 2015.

[150] Gehui Shen, Yunlun Yang, and Zhi-Hong Deng. Inter-weighted alignment network for sentence pair modeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1179–1189, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[151] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Li Shoubin, and Qing Wang. 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, New York, NY, USA, 2020. ACM.

[152] E. Shihab, Z. M. Jiang, and A. E. Hassan. Studying the Use of Developer IRC Meetings in Open Source Projects. In *2009 IEEE International Conference on Software Maintenance*, pages 147–156, Sept 2009.

[153] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 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, pages 107–110, Washington, DC, USA, 2009. IEEE Computer Society.

[154] Lokesh Shrestha and Kathleen McKeown. Detection of question-answer pairs in email conversations. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[155] Jonathan Sillito, Frank Maurer, Seyed Mehdi Nasehi, and Chris Burns. 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, pages 25–34, Washington, DC, USA, 2012. IEEE Computer Society.

[156] V. Sinha, A. Lazar, and B. Sharif. Analyzing developer sentiment in commit logs. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 520–523, 2016.

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

[158] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[159] Andrea Di Sorbo, Sebastiano Panichella, Corrado A. Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 12–23, Washington, DC, USA, 2015. IEEE Computer Society.

[160] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929?1958, January 2014.

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

[162] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (R) Evolution of Social Media in Software Engineering. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 100–116, New York, NY, USA, 2014. ACM.

[163] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2), 2017.

[164] Viktoria Stray, Nils Brede Moe, and Mehdi Noroozi. Slack me if you can!: Using enterprise social networking tools in virtual agile teams. In *Proceedings of the 14th International Conference on Global Software Engineering*, ICGSE '19, pages 101–111, Piscataway, NJ, USA, 2019. IEEE Press.

[165] X. Su, G. Gao, and Y. Tian. A framework to answer questions of opinion type. In *2010 Seventh Web Information Systems and Applications Conference*, pages 166–169, 2010.

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

[167] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. Live API Documentation. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 643–652, New York, NY, USA, 2014. ACM.

[168] Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *ArXiv*, abs/1511.04108, 2015.

[169] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Proc. IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering*, pages 44–53, February 2014.

[170] Y. Tian, D. Lo, and J. Lawall. 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)*, pages 44–53, Feb 2014.

[171] Rebecca Tiarks and Walid Maalej. How Does a Typical Tutorial for Mobile Development Look Like? In *Proc. 11th Working Conf. Mining Software Repositories*, pages 272–281, 2014.

[172] Slack Development Tools. https://slack.com/apps/category/At0EFRCDNY-developer-tools, 2018.

[173] C. Treude and M. P. Robillard. Augmenting API Documentation with Insights from Stack Overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 392–403, May 2016.

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

[175] Christoph Treude and Martin P. Robillard. Augmenting API documentation with insights from Stack Overflow. In *Proc. 38th Int'l Conf. Software Engineering*, pages 392–403, 2016.

[176] G. Uddin and F. Khomh. Automatic mining of opinions expressed about apis in stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[177] Gias Uddin and Foutse Khomh. Automatic summarization of api reviews. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, page 159?170. IEEE Press, 2017.

[178] David C Uthus and David W Aha. Multiparticipant chat analysis: A survey. *Artificial Intelligence*, 199:106–121, 2013.

[179] Carmine Vassallo, Sebastiano Panichella, Massimiliano Di Penta, and Gerardo Canfora. CODES: Mining source code descriptions from developers discussions. In *Proc. 22nd Int'l Conf. Program Comprehension*, pages 106–109, 2014.

[180] Isabel K. Villanes, Silvia M. Ascate, Josias Gomes, and Arilo Claudio Dias-Neto. What are software engineers asking about Android testing on Stack Overflow? In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, pages 104–113, 2017.

[181] Alexandra Vtyurina, Denis Savenkov, Eugene Agichtein, and Charles L. A. Clarke. Exploring conversational search with humans, assistants, and wizards. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, page 2187?2193, New York, NY, USA, 2017. Association for Computing Machinery.

[182] M. Wan and J. McAuley. Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 489–498, 2016.

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

[184] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI?17, page 4144?4150. AAAI Press, 2017.

[185] E. Wong, Jinqiu Yang, and Lin Tan. AutoComment: Mining question and answer sites for automatic comment generation. In *Proc. 28th Int'l Conf. Automated Software Engineering*, pages 562–567, Nov 2013.

[186] Edmund Wong, Jinqiu Yang, and Lin Tan. AutoComment: Mining question and answer sites for automatic comment generation. In *Proc. 28th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 562–567, 2013.

[187] Edmund Wong, Jinqiu Yang, and Lin Tan. AutoComment: Mining Question and Answer Sites for Automatic Comment Generation. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE'13, pages 562–567, Piscataway, NJ, USA, 2013. IEEE Press.

[188] Andrew Wood, Paige Rodeghero, Ameer Armaly, and Collin McMillan. 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*, ESEC/FSE 2018, page 491?502, New York, NY, USA, 2018. Association for Computing Machinery.

[189] Wei Wu, Xu Sun, and Houfeng Wang. Question condensing networks for answer selection in community question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1746–1755, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[190] X. Xia, D. Lo, D. Correa, A. Sureka, and E. Shihab. 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)*, volume 1, pages 73–82, June 2016.

[191] Y. Xiang, Q. Chen, X. Wang, and Y. Qin. Answer selection in community question answering via attentive neural networks. *IEEE Signal Processing Letters*, 24(4):505–509, 2017.

[192] Z. Xiong, P. Liang, C. Yang, and T. Liu. Assumptions in oss development: An exploratory study through the hibernate developer mailing list. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 455–464, 2018.

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

[194] Di Yang, Aftab Hussain, and Cristina Videira Lopes. From Query to Usable Code: An Analysis of Stack Overflow Code Snippets. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 391–402, New York, NY, USA, 2016. ACM.

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

[196] Q. Ye, K. Misra, H. Devarapalli, and J. T. Rayz. A sentiment based non-factoid question-answering framework. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 372–377, 2019.

[197] Liguo Yu, Srini Ramaswamy, Alok Mishra, and Deepti Mishra. *Communications in Global Software Development: An Empirical Study Using GTK+ OSS Repository*, pages 218–227. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[198] Alexey Zagalsky, Daniel M. German, Margaret-Anne Storey, Carlos Gómez Teshima, and Germán Poo-Caamaño. How the R community creates and curates knowledge: An extended study of Stack Overflow and mailing lists. *Empirical Software Engineering*, 2017.

[199] Y. Zhang and D. Hou. Extracting problematic api features from forum discussions. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 142–151, May 2013.

[200] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. "Multi-Factor Duplicate Question Detection in Stack Overflow". *Journal of Computer Science and Technology*, 30(5):981–997, Sep 2015.

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

[202] Zhou Zhao, Hanqing Lu, Vincent W. Zheng, Deng Cai, Xiaofei He, and Yueting Zhuang. Community-based question answering via asymmetric multi-faceted ranking network learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI?17, page 3532?3538. AAAI Press, 2017.

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

[204] Xiaoqiang Zhou, Baotian Hu, Qingcai Chen, Buzhou Tang, and Xiaolong Wang. Answer sequence learning with neural networks for answer selection in community

question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 713–718, Beijing, China, July 2015. Association for Computational Linguistics.

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

Appendix A

DATASETS AND REPLICATION PACKAGES

This appendix presents the links to the publicly available data sets and replication packages described in Chapters 4, 5, and 6 of this dissertation.

A.1 Chat Disentanglement Dataset and Replication Package (Chapter 4):

`https://zenodo.org/record/3627124#.YD6YyZNKjwc`

A.2 Post Hoc Quality Dataset and Replication Package (Chapter 5):

`https://bit.ly/3dkgA9g`

A.3 ChatEO Dataset and Replication Package (Chapter 6):

`https://tinyurl.com/y3qth6s3`

Appendix B

IRB/HUMAN SUBJECTS APPROVAL

This appendix presents the IRB approval notification documents.

UNIVERSITY OF DELAWARE

RESEARCH OFFICE

210 Hullihen Hall
University of Delaware
Newark, Delaware 19716-1551
Ph: 302/831-2136
Fax: 302/831-2828

DATE:	July 16, 2018
TO:	Lori Pollock
FROM:	University of Delaware IRB
STUDY TITLE:	[1293642-1] Automatically Enhancing Quality of Social Communication Channels to Support Software Developers and Improve Tool Reliability
SUBMISSION TYPE:	New Project
ACTION:	DETERMINATION OF EXEMPT STATUS
DECISION DATE:	July 16, 2018
REVIEW CATEGORY:	Exemption category #2

Thank you for your submission of New Project materials for this research study. The University of Delaware IRB has determined this project is EXEMPT according to federal regulations (45 CFR 690.101(b)(2)).

We will put a copy of this correspondence on file in our office. Please remember to notify us if you make any substantial changes to the project.

If you have any questions, please contact Maria Palazuelos at (302) 831-8619 or mariapj@udel.edu. Please include your study title and reference number in all correspondence with this office.

University of Delaware
Informed Consent Form

Title of Project: Automatically Enhancing Quality of Social Communication Channels to Support Software Developers and Improve Tool Reliability

Principal Investigator (s): Lori Pollock

Other Investigators: Kostadin Dameski, Virginia Commonwealth University

You are being asked to participate in a research study. This form tells you about the study including its purpose, what you will be asked to do if you decide to participate, and any risks and benefits of being in the study. Please read the information below and ask the research team questions about anything we have not made clear before you decide whether to participate. Your participation is voluntary and you can refuse to participate or withdraw at any time without penalty or loss of benefits to which you are otherwise entitled. If you decide to participate, you will be asked to sign this form and a copy will be given to you to keep for your reference.

WHAT IS THE PURPOSE OF THIS STUDY?

The purpose of this study is to evaluate the accuracy and usefulness of several of our techniques for automatically determining and improving the quality of developers' interactive

Communications on social media sites with regard to quality concerns including poor software properties, obsolescence, contradictions, unreliable sources, and duplication. We need human judges such as yourself who have programming experience to help us build gold sets of ground truth for evaluating our automatic system against what a human would expect.

You are being asked to take part in this study because you have the programming experience and background to provide a programmer's perspective. Participants must be 18 years of age or older, and have programming experience adequate for this kind of judgements. You do not need to meet any additional criteria beyond the ability to read and comprehend software and communications by software developers. Your identity will be kept with your data; however, results will be reported without identifying you directly and typically in aggregate in research papers. We will have about 6-20 participants in this study.

WHAT WILL YOU BE ASKED TO DO?

You will be asked to do one or some of the following procedures.

1. Manually judge pre-recorded conversations by our quality bot or other quality assessment information provided by our system
2. Complete inlined surveys within Slack channels about the behavior of a bot participating in the conversation

Participant Initials ________

3. Manually interpret and provide feedback about a bot-posted answers
4. Use our quality-assessment system and provide feedback on the usefulness of the quality assessments.
5. Participate in an in-person or remote interview with one of the researchers in this project to answer questions about the usefulness and accuracy of the quality feedback.

Participation in a study will be framed to take less than an hour. It will take place either in Smith Hall lab or remotely with instructions given via skype.

WHAT ARE THE POSSIBLE RISKS AND DISCOMFORTS?

There are no known risks or direct benefits to you if you participate in this experiment. If you refuse to participate, you will not be penalized.

WHAT ARE THE POTENTIAL BENEFITS?

Participants will learn about software maintenance tools they may not have been aware of previously, and could use in their own work environment. Society will benefit from the tools we are developing, with the results and feedback on their outputs by these human judges. Other researchers will benefit from our construction of gold sets of ground truth data for use in their evaluations, with identifying information removed.

HOW WILL CONFIDENTIALITY BE MAINTAINED?

We will make every effort to keep all research records that identify you confidential to the extent permitted by law. In the event of any publication or presentation resulting from the research, no personally identifiable information will be shared outside the lab.

Your research records may be viewed by the University of Delaware Institutional Review Board, but the confidentiality of your records will be protected to the extent permitted by law.

This research is sponsored by the National Science Foundation.

WILL THERE BE ANY COSTS RELATED TO THE RESEARCH?

There are no costs involved in participation.

DO YOU HAVE TO TAKE PART IN THIS STUDY?

Participation in this study is voluntary. It is your decision as to whether you want to participate. If you refuse to participate, you will not be penalized. If at any point during the experiment, you change your mind and wish to withdraw, you may do so. If you request to have the data collected from you destroyed, we will do so.

As a student, if you decide not to take part in this research, your choice will have no effect on your academic status or your grade in a class.

Participant Initials ________

WHO SHOULD YOU CALL IF YOU HAVE QUESTIONS OR CONCERNS?

If you have any questions about this study, please contact the Principal Investigator, Lori Pollock at 302 831-1953 or email Pollock@udel.edu.

If you have any questions or concerns about your rights as a research participant, you may contact the University of Delaware Institutional Review Board at 302-831-2137.

Your signature below indicates that you are voluntarily agreeing to take part in this research study. You have been informed about the study's purpose, procedures, possible risks and benefits. You have been given the opportunity to ask questions about the research and those questions have been answered. You will be given a copy of this consent form to keep.

______________________________	____________
Signature of Participant	Date
______________________________	
Printed Name of Participant	

Participant Initials ________

HUMAN SUBJECTS PROTOCOL
University of Delaware

Protocol Title: Automatically Enhancing Quality of Social Communication Channels to Support Software Developers and Improve Tool Reliability

Principal Investigator
Name: Lori Pollock
Department/Center: Computer and Information Sciences
Contact Phone Number: 484 678 7969 (cell) or 302 831-1953 (office – less available)
Email Address: Pollock@udel.edu

Advisor (if student PI):
Name:
Contact Phone Number:
Email Address:

Other Investigators: Kostadin Dameveski, Virginia Commonwealth University

Investigator Assurance:

By submitting this protocol, I acknowledge that this project will be conducted in strict accordance with the procedures described. I will not make any modifications to this protocol without prior approval by the IRB. Should any unanticipated problems involving risk to subjects occur during this project, including breaches of guaranteed confidentiality or departures from any procedures specified in approved study documents, I will report such events to the Chair, Institutional Review Board immediately.

1. **Is this project externally funded?** [✓ YES] ☐ NO

If so, please list the funding source:

2. Research Site(s)

[✓] University of Delaware

[✓] Other (please list external study sites) Virginia Commonwealth University

Is UD the study lead? [✓ YES] ☐ NO (If no, list the institution that is serving as the study lead)

3. **Project Staff**
Please list all personnel, including students, who will be working with human subjects on this protocol (insert additional rows as needed):

NAME	**ROLE**	**HS TRAINING COMPLETE?**
Lori Pollock	PI and co-advisor	yes
Kostadin Dameveski	Collaborator at VCU	
No students hired yet		

4. **Special Populations**
Does this project involve any of the following:

Research on Children? no

Research with Prisoners? no

If yes, complete the Prisoners in Research Form and upload to IRBNet as supporting documentation

Research with Pregnant Women?no

Research with any other vulnerable population (e.g. cognitively impaired, economically disadvantaged, etc.)? please describe no

5. **RESEARCH ABSTRACT** Please provide a brief description in LAY language (understandable to an 8th grade student) of the aims of this project.

Social communication channels (e.g., Stack Overflow, Slack, and GitHub) play an increasingly important role in software developer productivity, as developers
use these channels to collaborate and coordinate activities. Researchers have shown additional value in mining
information from these sources to develop and improve recommendation systems, software integrated development environments, and other tools. Both developers and researchers complain about the
varying and often poor quality of the information shared on these channels despite the intrinsic mechanisms for maintaining
quality (e.g., voting, accepted answers). This project aims to address the need for automation in curating for increased quality of the information
shared in developer communication channels.
The envisioned system uses multiple software developers' social communication channels as parallel sources of evidence to establish quality of
information in each channel.

This project will contribute to the state of the art by tackling three major challenges to bring

automated curation of developer communication into
practical use. First, we will develop analyses for automatically determining and improving the quality of developers' interactive communications on social media
sites with regard to quality concerns including poor software properties, obsolescence, contradictions, unreliable sources, and duplication. Second, we will
explore different feedback mechanisms to indicate quality findings back to the communities. Third, we will develop techniques to customize quality measurement
and feedback to a particular developer's context. The novel approach to analyses, the resulting tools, data sets, and experimental infrastructure developed
as part of the project will be released, enabling other researchers and practitioners to build on the project's results and ultimately advance the quality of
the modern software development ecosystem.

6. **PROCEDURES** Describe all procedures involving human subjects for this protocol. Include copies of all surveys and research measures.

Evaluation will focus on the effectiveness of quality improvement, specifically
addressing the questions: (1) How well does our approach identify relationships (mapping of specific topic and information relationship) between Q&A posts and chat conversations? (2) Is the information we supplement to a channel actually improving quality? (3) What kind of recall of quality-related information do we achieve? (4) In terms of providing information to a chat, how timely is the information for a current conversation? (5) What are the costs involved in information triangulation across social communication channels? Similar to our exploratory study methodologies, we will collect a large sample from a variety of Slack developer communities and overall
similarly tagged Stack Overflow posts. We will expand our current Slack and Stack Overflow data sets significantly for these studies, and develop a gold set manually using human annotators outside the project team.

We will recruit people with computer programming experience to provide us with their judgements on various results output by our developed software system. They will examine outputs that we provide, and provide judgements of the quality and usefulness of feedback, without knowing it came from our system.

To evaluate the quality of feedback provided by the proposed quality improvement bots, it is necessary to investigate how humans respond to the bot messages. Specifically for
the chat bot, evaluation can be performed by: 1) asking humans to manually judge pre-recorded conversations by the bot; 2) inlining surveys of the bot behavior from the participants of a conver-sation; or 3) automatic sentiment analysis of human responses to the bot recorded from numerous conversations. For the forum bot, evaluation would involve: 1) manually or automatically (via sen-timent analysis) interpreting comments to bot posted answers; and 2) measuring trends in voting influenced by the bot, by comparing activity before and after bot involvement.

The key characteristic that we intend to measure is the utility of the additional
context embedded in the communication channel. This can be directly observed in a few ways: 1) reduction of follow-up questions on Slack; 2) reduction of edits by the moderators and improvement of the time-to-answer for a question on Stack Overflow. Short surveys can be conducted within each communication medium intended to gauge the value of a specific context sentence. Longer surveys

and interviews can provide a summative view of the bots utility to a channel. Both surveys can be conducted by contacting participants directly in the public Stack Overflow or Slack channels.

7. **STUDY POPULATION AND RECRUITMENT**

Describe who and how many subjects will be invited to participate. Include age, gender and other pertinent information.

Human Subjects:

We will involve graduate and undergraduate students as well as full-time professional Java developers as subjects. The subjects will be volunteers not currently in any course involving the PIs, but instead from local companies and in the graduate or undergraduate program in computer science or related field at UD, with programming experience.

Attach all recruitment fliers, letters, or other recruitment materials to be used. If verbal recruitment will be used, please attach a script.

Describe what exclusionary criteria, if any will be applied.

Describe what (if any) conditions will result in PI termination of subject participation.

Termination of subject participation would be based upon whether the subject is taking the time to be careful about their judgement tasks to provide data that is reflective of a real user of the tool.

8. **RISKS AND BENEFITS**

List all potential physical, psychological, social, financial or legal risks to subjects (risks listed here should be included on the consent form).

In your opinion, are risks listed above minimal* or more than minimal? If more than minimal, please justify why risks are reasonable in relation to anticipated direct or future benefits.

There are no risks to a participant using these software maintenance tools and providing their judgements of our outputs, or answering our surveys or interview questions about their use of these communication channels and the feedback provided by our tool.

*(*Minimal risk means the probability and magnitude of harm or discomfort anticipated in the research are not greater than those ordinarily encountered in daily life or during the performance of routine physical or psychological examinations or tests)*

What steps will be taken to minimize risks?

The data per participant is kept within our own machines and anonymized before any public data for use by others outside the evaluation studies of our work are provided.

Describe any potential direct benefits to participants.

Participants will learn about software maintenance tools they may not have been aware of previously, and could use in their own work environment.

Describe any potential future benefits to this class of participants, others, or society.

Society will benefit from the tools we are developing, with the results and feedback on their outputs by these human judges. Other researchers will benefit from our construction of gold sets of ground truth data for use in their evaluations (anonymized).

If there is a Data Monitoring Committee (DMC) in place for this project, please describe when and how often it meets.

9. **COMPENSATION**

Will participants be compensated for participation?

No, they will be volunteers.
If so, please include details.

10. **DATA**

Will subjects be anonymous to the researcher?

If subjects are identifiable, will their identities be kept confidential? (If yes, please specify how)
Yes, we will keep the data with names inhouse in our lab.
The data will also be anonymized by taking all identifying information about the participants off the data completely before making the data public for other researchers.

How will data be stored and kept secure (specify data storage plans for both paper and electronic files. For guidance see http://www.udel.edu/research/preparing/datastorage.html)

How long will data be stored?

The data will be stored on the secure research lab server for as long as the data is needed.
The data with all participant information taken out will be made public for other researchers on a web page from our lab.

Will data be destroyed? ☐ YES [✓ NO] (if yes, please specify how the data will be destroyed)

Will the data be shared with anyone outside of the research team? [✓ YES] ☐ NO (if yes, please list the person(s), organization(s) and/or institution(s) and specify plans for secure data transfer)

Only without the participant information taken off, and shared on a web page for data sharing for others to perform replicated experiments.

How will data be analyzed and reported?

Data is analyzed by the PIs and students working on the project, and reported in aggregate in research publications, with participant names removed.

11. **CONFIDENTIALITY**

Will participants be audiotaped, photographed or videotaped during this study?

No

How will subject identity be protected?
The subject identity is known only to the PIs and students working on the project.

Is there a Certificate of Confidentiality in place for this project? (If so, please provide a copy).
no

12. **CONFLICT OF INTEREST**

(For information on disclosure reporting see: http://www.udel.edu/research/preparing/conflict.html)

Do you have a current conflict of interest disclosure form on file through UD Web forms? Yes, for PI with university

Does this project involve a potential conflict of interest*? no

* As defined in the University of Delaware's Policies and Procedures ,a potential conflict of interest (COI) occurs when there is a divergence between an individual's private interests and his or her professional obligations, such that an independent observer might reasonably question whether the individual's professional judgment, commitment, actions, or decisions could be influenced by considerations of personal gain, financial or otherwise.

If yes, please describe the nature of the interest:

13. **CONSENT and ASSENT**

____x Consent forms will be used and are attached for review (see Consent Template under Forms and Templates in IRBNet)

____ Additionally, child assent forms will be used and are attached.

____ Waiver of Documentation of Consent (attach a consent script/information sheet with the signature block removed).

____ Waiver of Consent (Justify request for waiver)

14. **Other IRB Approval**
Has this protocol been submitted to any other IRBs? no

If so, please list along with protocol title, number, and expiration date.

15. **Supporting Documentation**
Please list all additional documents uploaded to IRBNet in support of this application.

Appendix C
PERMISSIONS

This appendix presents the "Right links" from the corresponding journals/conferences that provide permissions to reuse these articles in this dissertation.

ACM Publications (Chapters 4 and 5):

Rights link (https://authors.acm.org/author-resources/author-rights)

IEEE Publications (Chapters 3 and 6):

Rights link (https://www.ieee.org/publications/rights/rights-link.html)

Copyright Clearance Center RightsLink®

Home | Help | Email Support | Sign in | Create Account

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

IEEE Requesting permission to reuse content from an IEEE publication

Conference Proceedings:
2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Author: Preetha Chatterjee

Publisher: IEEE

Date: May 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW