The Journal of Systems and Software 159 (2020) 110454

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts

Preetha Chatterjee*, Minji Kong, Lori Pollock

Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA

ARTICLE INFO

Article history:
Received 30 July 2019
Revised 22 October 2019
Accepted 28 October 2019
Available online 30 October 2019

Keywords:
Empirical study
Stack overflow
Mining software repositories
Software developer behavior

ABSTRACT

Monthly, 50 million users visit Stack Overflow, a popular Q&A forum used by software developers, to share and gather knowledge and help with coding problems. Although Q&A forums serve as a good resource for seeking help from developers beyond the local team, the abundance of information can cause developers, especially novice software engineers, to spend considerable time in identifying relevant answers and suitable suggested fixes.

This exploratory study aims to understand how novice software engineers direct their efforts and what kinds of information they focus on within a post selected from the results returned in response to a search query on Stack Overflow. The results can be leveraged to improve the Q&A forum interface, guide tools for mining forums, and potentially improve granularity of traceability mappings involving forum posts. We qualitatively analyze the novice software engineers' perceptions from a survey as well as their annotations of a set of Stack Overflow posts. Our results indicate that novice software engineers pay attention to only 27% of code and 15–21% of text in a Stack Overflow post to understand and determine how to apply the relevant information to their context. Our results also discern the kinds of information prominent in that focus.

1. Introduction

Modern software development communities are becoming increasingly social by contributing to crowdsourced content and using various social tools, including public code repositories, developer question and answer (Q&A) forums, and online chat communities. Monthly, about 50 million people visit the Stack Overflow Q&A forum to learn and share information (S.O.D.S., 2018). Stack Overflow provides a vast resource for software developers to get help from other developers beyond their local developer community. Stack Overflow users include those who post questions, answer questions, and search existing Q&A posts seeking help. In this paper, we focus on Stack Overflow from the perspective of an individual seeking help with programming errors.

An information seeker typically formulates a search query based on the task to be accomplished, (in this case, the programming task and error they are experiencing), uses that query to search online using a search engine or Q&A forum search, and retrieves a list of results related to the query. Even after their search query returns a set of relevant posts from Stack Overflow (and possibly other forums), users seeking answers are faced with a set of posts that require time to identify the relevant part of the post that provides an appropriate solution for their context. Q&A forums often contain long threads of discussions. Our analysis shows that on average, each sample Stack Overflow post (consisting of just the question and at most two answers) in our data set, contains approximately 27 sentences and 23 lines of code. The median length of the subjects of our study is 23 sentences and 15 lines of code, and the maximum number of sentences and lines of code could be as high as 187 and 240 respectively.

We have observed that for posts related to object-oriented programming languages, developers include entire methods and even whole classes because they are not sure where the problem lies, and to provide context for others to help them identify their problem. Code segments in questions are typically accompanied by detailed descriptions of the questioner's understanding of the problem and errors/exceptions that they encountered.

A number of researchers have studied the problem of finding relevant information from Q&A forums (Gottipati et al., 2011; Zou et al., 2015b; Xu et al., 2017). Xu et al. (2017) conducted a survey to understand the developers' difficulties of extracting relevant information from Q&A forums. Some excerpts of their survey responses are: "Google will return a number of 'relevant' links for a query, and I have to click into these links, and read a number of

* Corresponding author.
E-mail address: preethac@udel.edu (P. Chatterjee).

https://doi.org/10.1016/j.jss.2019.110454
0164-1212/

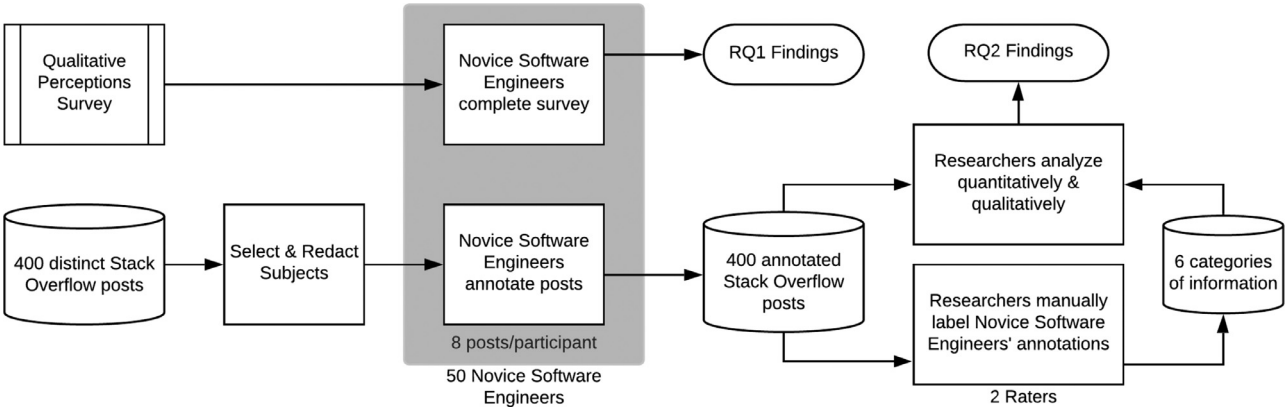

Fig. 1. Overview of exploratory study methodology.

paragraphs... It is really time-consuming... ", "... Some questions received too many long answers, and many of these answers have redundant content." Xu et al. also illustrated an example of how information overload can be detrimental to a developer. They used Google to search for Stack Overflow posts to understand the differences between HashMap and HashTable in Java. The top 5 ranked Stack Overflow questions returned by Google, consist of 51 answers which have 6771 words in total. Reading these answers at an average reading speed of 200 words per minute could take about 30 min. The authors also mentioned that *"Even just reading the best answers (i.e., the accepted answers) or top-voted answers may still take some time*."

Researchers have developed techniques and tools to help information seekers on Stack Overflow, including reformulation of queries (Nie et al., 2016; Li et al., 2016) and automatic recommendation of tags (Saha et al., 2013; Beyer and Pinzger, 2015) both targeted at retrieving better search results. However, to our knowledge, no one has investigated how information seekers on Stack Overflow focus their efforts in identifying their needed information from a selected post result after they have narrowed down their search to a relevant post in response to a search query.

In this paper, we report on an exploratory study that we conducted with the goal of understanding novice software engineers' challenges in pinpointing their needed information from a selected post result, and their recommendations for helping to reduce their time in locating the specific information. Finding help is especially difficult for novice software engineers since they do not have sufficient expertise and prior knowledge of the subject to quickly skim through a post to find the relevant information. Our study focuses on the common case of Stack Overflow use, where a novice software engineer is seeking help to fix errors and exceptions in their application (Treude et al., 2011).

Specifically, we conducted an exploratory case study to answer the following research questions:

- **RQ1** *Challenges* - Assuming that the relevant post in Stack Overflow is already identified, what slows a novice software engineer down in identifying the solution most appropriate for their problem?
- **RQ2** *Focus* - Which parts of a Stack Overflow post would these novice software engineers recommend highlighting to focus attention quickly to reduce their time in locating information within a post?

The results of our study could be leveraged for: (1) designing a user interface of Stack Overflow that draws the reader's attention to the relevant parts of the thread to help them to quickly choose a suitable solution, (2) guiding tools for mining Stack Overflow for knowledge gathering to the most salient parts of the question and answers, and (3) improving granularity of traceability mappings to Stack Overflow from other software artifacts and tools.

2. Methodology

2.1. Overview

Fig. 1 shows the major steps of our exploratory study. To answer research question 1 (RQ1), we developed and distributed a survey to novice software engineers to gather their perceptions of what slows them down in their Stack Overflow usage. The second research question (RQ2) is answered by novice software engineers examining and annotating a random sample of Stack Overflow posts to indicate what they would recommend to highlight to focus attention. We contacted 50 novice software engineers to participate in our study and selected 400 unique Stack Overflow posts. Two authors then manually analyzed the annotations, labeling the different kinds of information highlighted by the novice software engineers. This manual labeling resulted in identification of six major categories of highlighted information in the Stack Overflow posts. Using the identified labels, the researchers qualitatively and quantitatively analyzed the annotations again to answer RQ2, and gain more insight into where they focus their attention.

2.2. Novice software engineers

We conducted our study during lab sessions of an undergraduate software engineering course. Limiting the study to students enrolled in a specific course helps us in identifying participants with similar experience and skill level. We received responses from 50 participants in total, all with prior programming experience of at least 2 years in Java and/or C++. Each participant was asked to indicate their Stack Overflow usage as either (1) *Never (don't know what it is)*, (2) *Seldom (at least once in six months)*, (3) *Periodically (at least once in a month)*, or (4) *Frequently (at least once a week)*. The responses indicate that 62% use Stack Overflow frequently, 32% use it periodically, and 6% use it seldom. None of the participants indicated 'Never'. Thus, at least 94% use Stack Overflow at least monthly. Since this is a study of novice software engineers, we do not expect lots of heavy users of Stack Overflow. Each participant was also asked to indicate their experience as a software engineer outside school (e.g., summer intern, part-time job). 51.1% participants selected "yes", and 48.9% selected "no" to this question.

2.3. Pilot study

To finalize the survey and annotation task, we first performed a pilot study with 25 developers. We assigned 4 posts to each of

Q1: When you are using Stack Overflow to help you in your coding, and once you have identified a particular post, what do you believe is the major obstacle(s) that slows you down in reading through the post to identify the solution you believe will be most appropriate/applicable for your problem? You may select more than one. *

- ☐ Too much text containing unnecessary details
- ☐ Too much code containing unnecessary details
- ☐ Determining if code segment in the post is similar to my code
- ☐ Identifying the cause of error/exception
- ☐ Identifying the suggested fix
- ☐ Absence of accepted answers
- ☐ Absence of suggested code in answers
- ☐ Platform dependencies and deprecated APIs
- ☐ Other

Q2: When there is significant amounts of code in the post, which code components in the question/answer/comment would you recommend highlighting to focus your attention quickly? You may select more than one. *

- ☐ Only the problematic code (code responsible for error/exception) in the question
- ☐ Problematic code in the question and its context (e.g. 5 lines before and after the problematic code)
- ☐ Entire solution code as suggested in the answer/comment
- ☐ Only the necessary code modifications as suggested in the answer/comment
- ☐ Only the suggested code in verified/accepted answers
- ☐ Other

Q3: Which natural language text in the question/answer/comment do you think highlighting would help to focus your attention quickly? You may select more than one. *

- ☐ Text in the question describing the developer's intended outcome
- ☐ Text in the question describing the error/exception
- ☐ Text in the answer/comment describing the cause of the error/exception
- ☐ Text in the answer/comment describing the suggested solution
- ☐ Text in the answer/comment describing why the suggested solution works
- ☐ Text clues in the answer/comment pointing to suggested solution (e.g. "following code", "below code")
- ☐ A comment that reaffirm a solution's correctness (e.g. "Thanks, this code works.")
- ☐ Programming environment dependencies
- ☐ Coding best practices
- ☐ Other

Fig. 2. Survey form for novice software engineers' perceptions.

our participants, which took them approximately 20 min to annotate. In total, the participants submitted annotations on 100 Stack Overflow posts (2 to 6 different opinions on 30 unique posts). The questions asked in the pilot study were:

PQ1: Assuming that the Stack Overflow post provided to you in this study is relevant to your question, what slows you down in identifying the solution you believe will be most appropriate/applicable for your problem?

PQ2: Which code in the question/answer would you recommend highlighting to focus attention quickly?

PQ3: Which natural language text in the question/answer do you think highlighting would help to focus attention quickly? Since the purpose of the pilot study was to understand the common slowdowns faced by novice software engineers in identifying information relevant to a problem they are experiencing, we framed the first question as a leading question. However, to allow flexibility of opinions, we made the answer in the form of a short descriptive text. We summarized the responses that we received from the pilot study by clustering the related responses into themes. These themes were used as the answer options to each question along with the option to enter a new answer in the survey of novice software engineers' perceptions, which we describe in the next subsection. Since the 30 unique posts used in the pilot study are not representative of the large number of error/exception related posts on Stack Overflow, we expanded the sample size for our survey of novice software engineers to 400 unique posts, to obtain a confidence level of 95% and margin of error of 5%.

2.4. Survey of novice software engineers' perceptions

Fig. 2 shows the survey questions that the novice software engineers were asked. The answers to the first question were used to

provide insight into research question RQ1. The remaining questions in the survey form were used to collect the novice software engineers' perceptions on research question RQ2. To gain perceptions in the form of reflective feedback after looking at concrete examples, novice software engineers completed all the survey questions after they had completed their annotation task.

2.5. Novice software engineer annotation study

To collect data from participants actually reading concrete example posts in addition to providing their perceptions on a survey, each novice software engineer in our study was asked to take on the role of an information seeker for a sample of Stack Overflow posts. This section describes the details of the Stack Overflow post selection and redaction, manual highlighting task by participants, and labeling and analysis of the annotations by the researchers.

2.5.1. Stack overflow post selection and redaction

We selected a set of 400 Stack Overflow posts related to errors and exceptions in the commonly used Java and C++ programming languages, using Stack Exchange Data Explorer (Explorer, 2019). Our data set comes from a wide time range from August 2008 until March 2019. We retrieved posts that contain the tag 'Java' and/or 'C++', since all our novice software engineers have considerable programming experience in these two programming languages.

Previous studies have categorized Stack Overflow questions into several categories such as "how-to", "conceptual", "discrepancy", and "error" (Treude et al., 2011; Rosen and Shihab, 2015; Beyer et al., 2018). Since, the focus of this study is to understand novice software engineer behavior on finding help with programming errors, we specifically selected posts where the title contains one or more of the terms "error", "exception", "bug". It should be noted that the "errors" category is in the top 4 most frequently occurring question categories.

To ensure selection of good quality posts, we chose posts with a total vote count of 3 or higher. We chose posts in which the question contains at least one code segment. Including code segments in questions helps the asker to clarify or specify the information need she is seeking for Baltadzhieva and Chrupala (2015), and also provides the Stack Overflow audience more context to understand and/or answer the question. Program-specific questions such as asking about how to fix a bug are difficult to answer if no code fragment is included in the question (Asaduzzaman et al., 2013).

We excluded posts that did not have at least two answers. If any of the selected posts had more than two answers, we chose the top two answers (the "accepted/best voted answer" and the "second answer"), and redacted the remaining answers to ensure that the participants did not spend undue time on each post, preventing them from annotating more posts in a reasonable time. For posts that did not have an accepted answer, we chose the top 2 best voted answers. In some posts, the accepted answer is also the best voted answer, so we chose the "accepted/best voted answer" and the second best voted answer as the "second answer". For posts where the accepted answer was not the same as the best voted answer, we chose the accepted answer as the "accepted/best voted answer" and the best voted answer as the "second answer". The redaction helped to ensure that the study took no longer than 60 min for each participant. The documents contained all attributes of the original posts such as date, timestamp, comments, vote count, tags etc. The final data set consisted of 400 unique questions, and 800 unique answers.

To gain insight into the overall quality and popularity of the posts selected in our data set, we give some statistics as follows:

(a) The minimum vote count for the questions in our data set is 3; the maximum is 1795 (mean = 32).

(b) The minimum answer count (number of answers to a question in a post) is 2; the maximum is 32 (mean = 6).

(c) 324 out of 400 (~81%) posts contain an accepted answer.

(d) The minimum view count of the posts is 263; the maximum is 1,104,490 (mean = 35,415).

2.5.2. Novice software engineers' manual highlighting

Each participant completed the concrete task of annotating a set of posts following the instructions: *"For each of the posts provided as pdf, highlight the minimum amount of code and phrases/sentences in the question and answer, that you believe would be most helpful in making you more efficient in understanding your problem and the suggested solution."* To ensure that the participants understood the instructions, we first conducted a pilot study with 25 developers and 100 Stack Overflow posts (details in Section 2.3). We provided the participants with opportunities to ask for clarification of the instructions during the pilot study. We analyzed the annotations, and based on the responses from the pilot study we revised the instructions for the main survey.

Fig. 3 shows an example manual annotation by a participant. This example post is longer in reality, we removed parts of the post from this figure since it is difficult to fit the entire post in one figure. Overall, we have removed 15 lines of code/stack traces and 5 sentences in this figure. Also, note that we are showing only the question and the accepted answer, whereas the study document contained the question, the two best answers, and the comments to the question and answers.

We assigned 8 posts to each of our novice software engineer participants, which together took approximately 40 min to annotate. In total, the participants submitted annotations on 400 unique Stack Overflow posts, which corresponds to 400 unique questions and 800 unique answers (two answers selected for each question as a part of document selection).

2.5.3. Researchers' manual labeling of annotations

Examination of the annotations indicates that the annotations consist of different types of information, which are important to understand for building automated systems to help novice software engineers on Stack Overflow, as well as to improve tools that mine Stack Overflow for information. Therefore, we followed an inductive approach (Runeson et al., 2012) to qualitatively analyze the annotations on the Stack Overflow posts. Two of the authors performed the manual labeling. We created a code book to define the labels and memos to facilitate the process of analysis. For labelling each annotation, we wrote memos to record comments and reflections by the researcher, and additional information (if applicable).

The analysis was performed in an iterative approach comprised of multiple sessions, which helped in generalizing the hypotheses and revising the definitions of the labels. To calculate inter-rater agreement, we used Cohen Kappa coefficient (Stemler, 2004). Section 3 describes the labels, which constitute our categories of information, and the inter-rater agreement findings.

2.5.4. Quantitative and qualitative analysis of labeled annotations

After deriving categories of information from the manual labeling of Stack Overflow highlights from the participants, we returned to the annotations to perform a deeper analysis of the highlighted text, code, and stack traces. To better understand how the portions of code, text, and stack traces in a Stack Overflow post are typically helpful for novice software engineers to understand their problem and select a potentially suitable solution, we manually counted the number of sentences highlighted in the natural language text and the number of lines of code and stack traces highlighted in each post. We compared them to the total number of sentences and lines of code/stack traces per post. Since, in multiple cases, participants highlighted phrases instead of complete sentences, we also

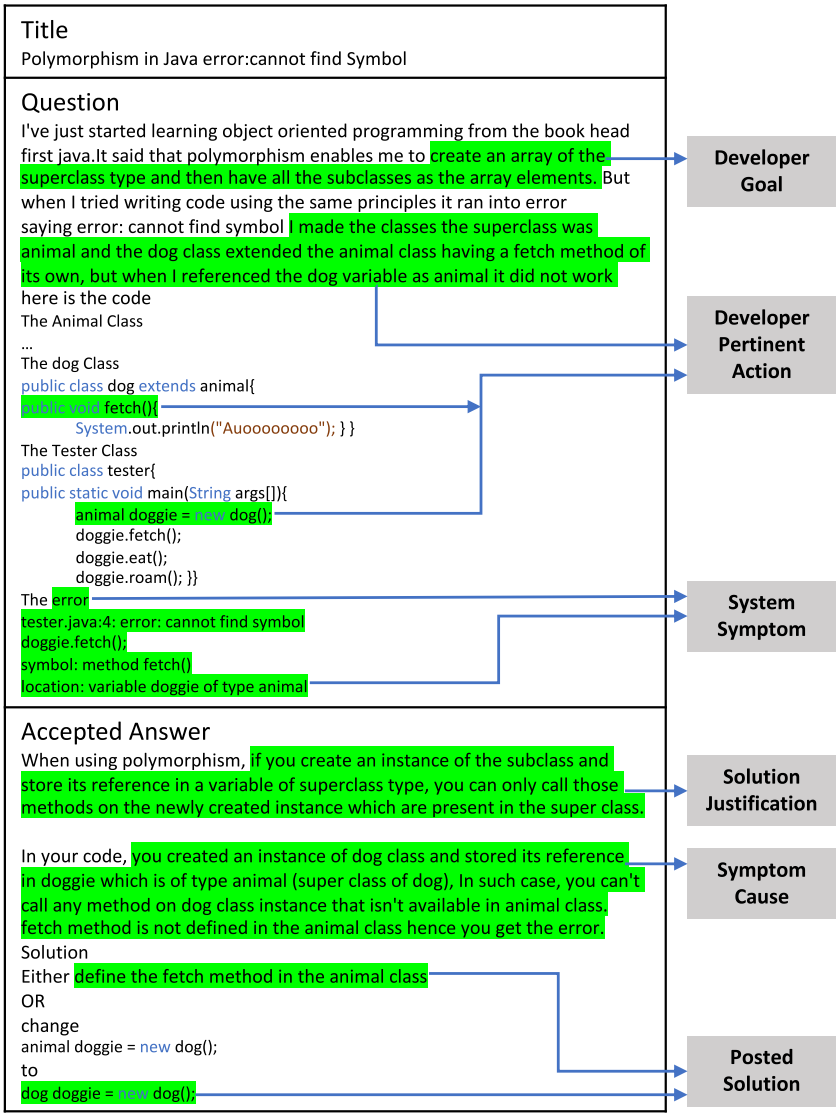

Fig. 3. Example of labelled highlights in an annotated post.

calculated the percentage of words highlighted in each post. We also analyzed the location of the highlighted portions with regard to Stack Overflow question and title, accepted or second answers, and comments that occur in a question or either of the two answers. These quantities are separately reported by text, code, and stack traces.

To identify the prevalence of each type of information available in a post, we first computed their frequency and percentage of occurrence across all the posts. To gain an insight into how important different information is to novice software engineers, we then calculated the frequency and percentage of posts where participants highlighted code/text related to each type of information. We separated these results by Stack Overflow question and answers.

Lastly, we collected the responses from the last two questions in the perceptions survey (in Fig. 2) which the participants took after completing their annotation task. We summarize those results.

2.6. Threats to validity

Here, we describe the potential threats to validity of our study and how we designed our study to minimize the threats, organized by kinds of threats (Runeson et al., 2012).

Construct validity. As with any case study based on human participants, the results of the study are subjective to human judgement, and based on individuals' opinions. To minimize this threat, we ensured that all the participants had considerable programming and Stack Overflow experience. Labeling of annotations in our study was performed by researchers, and therefore is dependent on human judgement. To limit this threat, two researchers with experience in qualitative data analysis performed the labeling. We calculated the inter-rater agreement using Cohen Kappa coefficient, and iteratively discussed and resolved conflicts, resulting in coefficient of 0.8.

Internal validity. It is possible that participants highlighted portions of posts based on factors other than the information that is actually highlighted, whereas the researchers analyzed the highlights based on content. To minimize this threat, we also collected the perceptions of the participants after they completed their concrete task. Our study examines the focus of novice software engineers after they have narrowed down to a single post. The results might vary if we considered the case where no post contains a fix for their error. Additionally, if any of the selected posts had more than two answers, we chose the top two answers and redacted the remaining to ensure that the participants did not

spend undue time on each post. In a real environment novice software engineers might spend a lot of time reading through those additional answers. Analyzing that scenario is beyond the scope of this study. Since the study participants were provided with sample Stack Overflow posts to analyze, they were not experiencing the errors/exceptions that triggered them to search for solutions on Stack Overflow. Therefore, the participants could have possibly been unaware of the context of the problem that information seekers in similar situations would have. Hence as a future work, we plan to expand the study by providing specific aims for the novice software engineers and letting them solve a programming task using the information from the annotated post.

External validity. Various kinds of questions with specific developer intents are asked on Stack Overflow, including "how-to", "decision help", "error". Treude et al. (2011). Our study focus is on questions related to errors/exceptions. The scope of our case study was intentionally designed to learn about the behavior of the novice software engineers who are seeking information about errors and exceptions. The programming language of the selected posts are Java and C++. Thus, the results may not apply to posts with other programming languages. However, we chose posts discussing various types of errors and exceptions as the subjects of the study, and since Stack Overflow and other Q&A forums have abundant threads on fixing/avoiding errors and exceptions, we believe our results would be applicable to those types of posts. More research is necessary to investigate the generalization of our findings to all types of questions in Stack Overflow. The kinds of information are highly dependent on the developer discussion format and hence we do not claim the generalizability of the results of this study across other datasets such as Github issue trackers, etc. The results of the study are based on 50 responses on 400 distinct Stack Overflow posts. It is possible that scaling to more perceptions might lead to different observations. Also, the targeted participants in this study are undergraduate students who are enrolled in introductory software engineering course, and are therefore novice software engineers. Expanding this study to a wider audience with different programming skills and experience might yield different results.

Reliability. To ensure that the questions and instructions in our study were clear and easy to understand, we performed a pilot study with 25 developers prior to distributing the task and survey to our participants. Based on their feedback, we revised and incorporated necessary changes before conducting the study.

3. Findings

We present our findings by research question.

RQ1: Once a novice software engineer has identified a Stack Overflow post relevant to their question, what slows them down in identifying the solution most appropriate for their problem?

Table 1 shows the percentage of the 50 participant responses in each category of the question "What do you believe is the major obstacle(s) that slows you down in reading through the post to identify the solution you believe will be most appropriate/applicable for your problem?" The majority of participants answered *"Too much text containing unnecessary details"*, followed by *"Determining embedded code segment relevance"*, and *"Too much code containing unnecessary details"*. Only 4% participants selected the "Other" option, in which a participant stated *"Often, the answers might be too complicated for me to understand"*.

Table 1
Participant Responses on Question1 of survey .

Perceived cause of slowdown	%Participant
Too much text containing unnecessary details	69
Determining embedded code segment relevance	51
Too much code containing unnecessary details	39
Absence of accepted answers	36
Absence of suggested code in answers	35
Identifying the suggested fix	28
Identifying the cause of error/exception	27
Platform dependencies and deprecated APIs	26

RQ2: Which parts of a Stack Overflow post would novice software engineers recommend highlighting to focus attention quickly to reduce their time in locating information within a post?

We first present the findings from the annotation study, followed by the participant perceptions from the survey.

Quantity and Location of Participant Highlights: Table 2 presents the results on how much and where the participants highlighted natural language text in the different components of the Stack Overflow posts in our study. The first column indicates the different *Components* in each post: *Question & Title, Answers* and *Comments*. Typically, the title contains only one sentence, so we combined the title and the question body into one component to calculate the measures. We further split answers into 'Accepted/Best Voted' answers and 'Second" answers'. Since comments can be written for both questions and answers, we separated our findings for comments accordingly. Out of 400 posts in total, comments were present for 225 questions, 299 accepted/best voted answers, and 250 second answers.

To provide context, the *Total #Snts* and *Total #Words* columns report the total number of sentences and words, respectively, present in all posts in our data set. The *#Hghlt Snts* and *#Hghlt Words* columns show the total number of sentences and words respectively, that were highlighted by the participants. In the *%Hghlt Snts* and *%Hghlt Words* columns, we present the percentage of highlighted sentences and words across our data set, respectively. For example, using Table 2, we observe that 716 of 2899 sentences (24.7%), and 9695 of 52,391 words (18.5%) were highlighted in the titles and questions. When participants highlighted phrases instead of entire sentences, we counted the entire sentence containing potentially multiple highlighted phrases as one sentence when calculating the number of highlighted sentences. The remaining columns in Table 2 help us to understand the proportions of each component that are highlighted by the participants. Specifically, we report the mean number of sentences and words present in each post per component in the *Mean #Snts/ Comp* and *Mean #Words/ Comp* columns, respectively. The *Mean #Hghlt Snts/Comp* and *Mean #Hghlt Words/Comp* columns are the mean number of sentences and words highlighted in each post per component.

Analysis of Table 2 provides several key observations:

- In total, only 21% of sentences and 15% of words were highlighted. These results suggest that very little text needs to be highlighted to focus a novice software engineer's attention to speed up their use of Stack Overflow. This implies that there may be a much text in posts that may not be necessary for novice software engineers to find and reuse a solution to their problem.
- Novice software engineers focus their attention primarily on the accepted answer, highlighting 42% of the sentences, and then almost equally on 'Question & Title' and 'Second answer' with 25% highlighted sentences in each of these components. Upon analyzing the posts where the second answers were highlighted more than the accepted/best voted answers, we noticed that the comments in those posts indicated the second answer to be a better answer than what was best voted and/or accepted, and hence some participants chose to highlight more information in the second answer. Our qualitative analysis of these cases suggests that novice software engineers may look

Table 2
Quantity and Location of participant Highlighted Natural language text (Sentences and Words) in Stack Overflow Post [Sentences(Snts), Highlighted (Hghlt), Component(Comp)] .

Component		Total #Snts	#Hghlt Snts	%Hghlt Snts	Mean #Snts/ Comp	Mean #Hghlt Snts/Comp	Total #Words	#Hghlt Words	%Hghlt Words	Mean #Words/ Comp	Mean #Hghlt Words/Comp
Question & Title		2899	716	24.7	7.2	1.8	52,391	9695	18.5	131.0	24.2
Answers	Accepted/ Best Voted	2104	877	41.7	5.3	2.2	42,244	13,547	32.1	105.6	33.9
	Second	1766	445	25.2	4.4	1.1	36,050	6604	18.3	90.1	16.5
	Question	885	42	4.8	3.9	0.2	16,979	479	2.8	75.5	2.1
	Accepted/ Best Voted Answer	1939	96	4.9	6.5	0.3	39,558	1382	3.5	132.3	4.6
Comments	Second Answer	1061	20	1.9	4.2	0.1	19,888	228	1.1	79.6	0.9
Total		10,654	2196	20.6	26.6	5.5	207,110	31,935	15.4	517.8	79.8

beyond accepted/best voted answer when either the accepted answer did not have enough information/explanation about the suggested solution, or other readers indicated that the suggested solution is inefficient/wrong.
- Very little text is highlighted in comments, only 5% sentences for comments in question and accepted answer, and 2% for comments in second answer text. Our participants do not find the comments to be a key source of information in their usage of Stack Overflow.

Table 3 presents our results on how much and where the participants highlighted the code and stack traces in the different components of a Stack Overflow post. Again, the first column shows the different *Components* in each post. The *Total #LOC* and *Total #LST* columns report the total number of lines of code (LOC) and lines of stack trace (LST), respectively, appearing across all posts in our data set. The *#Hghlt LOC* and *#Hghlt LST* columns provide the total number of lines of code and stack traces, respectively, highlighted by the participants. In the *%Hghlt LOC* and *%Hghlt LST* columns, we report the percentage of highlighted lines of code and stack traces. We show the mean number of lines of code and stack traces present in each post per component in the *Mean #LOC/Comp* and *Mean #LST/Comp* columns, respectively. The *Mean #Hghlt LOC/Comp* and *Mean #Hghlt LST/Comp* columns present the mean number of lines of code and stack traces highlighted in each post per component.

From Table 3, we make several key observations:

- In total, only 27% of code and 23% of stack traces are highlighted. Similar to the natural language text in the posts, this reflects the prevalence of much longer code and stack traces than needed for novice software engineers to identify and reuse a solution to their problem.
- Similar to novice software engineers focusing primarily on the natural language text of posts in the accepted answer, they also focus primarily on the code in the accepted answer with 38% highlighted code. Unlike the natural language text, they focus next on the code in the second answer with 30% highlighted code, more than the question and title with 22% highlighted code. This suggests that novice software engineers are looking for suggested code examples to reuse as opposed to studying the buggy code included in the question. Also, the total number of lines of code in Question & Title is much higher (5k LOC) than in the answers (2k LOC), and not all of the additional code may be necessary for novice software engineers to determine if the problem in the question matches with the problem in their context.
- The percentage of highlighted lines of stack traces in the answers (30 to 38%) is higher than the question (20%), primarily because the total lines of stack traces present in the answers across all the posts is low, which can skew the overall percentage. Stack traces are rarely found in answers, and are mostly present only in the questions (1.2k LST) to show more details about the encountered errors or exceptions.
- Overall, 20% code in comments is highlighted compared to 68% of answers highlighted, indicating that novice software engineers do not focus much on code in comments compared to answers. Although stack traces are useful for developers to locate bugs in their code, both code and stack traces are rarely present in comments. Comments to the Question mostly contain additional follow-up or clarification questions, and comments to the Answer mostly contain additional information related to the solution provided. Out of 400 posts, 41 posts have code in comments, and only 2 posts have stack traces in comments.

Categories of highlighted information: Through our two researchers' qualitative analysis of the highlights, we identified six main types

Table 3
Quantity and Location of participant Highlighted Code and Stack trace in Stack Overflow Post [Lines of Code (LOC), Lines of Stack trace (LST), Highlighted (Hghlt), Component (Comp)] .

Component		Total #LOC	#Hghlt LOC	%Hghlt LOC	Mean #LOC/ Comp	Mean #Hghlt LOC/Comp	Total # LST	#Hghlt LST	%Hghlt LST	Mean #LST/ Comp	Mean #Hghlt LST/Comp
Questions & Title		5152	1114	21.6	12.9	2.8	1280	248	19.4	3.2	0.6
Answers	Accepted/ Best Voted	2063	774	37.5	5.2	1.9	253	95	37.5	0.6	0.2
	Second	1895	564	29.8	4.7	1.4	63	19	30.2	0.2	0
Comments	Question	15	3	20.0	1.9	0.4	1	0	0.0	1.0	0.0
	Accepted/ Best Voted Answer	32	7	21.9	1.4	0.3	0	-	-	-	-
	Second Answer	16	3	18.8	1.3	0.3	1	0	0.0	1.0	0.0
Total		9173	2465	26.9	22.9	6.2	1598	362	22.7	4	0.9

Table 4
Presence of information category in participants' highlights .

Component	Category	Location		
		Text	Code	Stack Trace
Question & Title	Developer's Goal	✓	-	-
	Developer's Pertinent Action	✓	✓	-
	System Symptom	✓	-	✓
Answer	Symptom Cause	✓	-	-
	Posted Solution	✓	✓	-
	Solution Justification	✓	-	-
Comments	Developer's Pertinent Action	✓	✓	-
	System Symptom	✓	-	✓
	Symptom Cause	✓	-	-
	Posted Solution	✓	✓	-
	Solution Justification	✓	-	-

of information in a Stack Overflow post that were prevalent across the participants' highlights.

Developers' Goal: The software developer's objective or intended outcome.

Developer's Pertinent Action: The software developer's action that led to the symptom that they are experiencing. This category can be further divided into three subcategories:

Specific problematic code segment: The exact lines of code that need to be fixed, typically identified by an information helper in an answer, but occurring (usually embedded in a larger code segment) in the developer's question.

Code surrounding the problematic code: The extra code included beyond the problematic code which the questioner posted because they know the problem lies somewhere in the whole code segment and possibly want to provide more context for others to help identify the problematic code.

Text: A description of (all or) part of the action that the developer took that led to the problematic code. The questioner will often describe a set of actions they took, but only some of them led to the problematic code. The text describing the action that actually led to problematic code can be identified using the information helper's indication of the problematic code.

System Symptom: Observed system response to the developer's action that does not meet the developer's intended outcome.

Symptom Cause: Reason why the developer action resulted in the System Symptom.

Posted Solution: Suggested alternate approach to meet the developer's intended outcome and avoid/fix the System Symptom.

Solution Justification: Reason why the Posted Solution achieves the intended outcome.

Figure 3 shows an example highlighted Stack Overflow post[1] with labelled categories. Table 4 indicates the post components in which each category of information typically occurs in our data set. Developer's Goal, Developer's Pertinent Action, and System Symptom are related to the situation and problem faced by the questioner, and are therefore most frequently in *Question & Title*. The remaining categories Symptom Cause, Posted Solution, and Solution Justification pertain to the suggested fix as proposed by other contributors. They could be present in either answer or comments. Comments rarely contain Developer's Pertinent Action, or System Symptom, but when present, they occur in comments mostly when the original questioner adds information as comments. Typically, comments seldom contain Solution Justification, as comments are rarely long enough to accommodate such information. *Qualitative categorization of participant highlights:* Tables 5 and 6 present the results from qualitative analysis of the categories of information

[1] https://stackoverflow.com/questions/54266858/polymorphism-in-java-error cannot-find-symbol

Table 5
Frequency of information category in participants' highlights of question and title.

Question/title (400 Question instances)	Dev's goal	Dev's pertinent action				System symptom		
		Final	Prob. code	Surr. code	Text	Final	Stack trace	Text
#Instances	375	380	212	247	312	320	131	317
#Highlighted instances	163	201	97	73	124	184	69	158
%Highlighted instances	43.5	52.9	45.8	29.6	39.7	57.5	52.7	49.8

Table 6
Frequency of information category in participants' highlights of answers .

Answers		Symptom Cause	Posted Solution			Solution Justification
			Final	Code	Text	
Accepted/ Best Voted Answer (400 Answer Instances)	#Instances	189	362	353	157	173
	#Highlighted instances	138	276	245	103	80
	%Highlighted instances	73	76.2	69.4	65.6	46.2
Second Answer (400 Answer Instances)	# Instances	177	351	333	157	158
	#Highlighted instances	67	175	137	68	42
	%Highlighted instances	37.9	49.9	41.1	43.3	26.6

in each of the participants' highlights. Before qualitatively labeling the participant highlights with the category of information, two of the authors separately coded a set of highlights and compared labeling to insure adequate agreement to label separately. The coefficients for each type of information were as follows: Developer's Goal (0.80), Developer's Pertinent Action (0.87), System Symptom (0.88), Symptom Cause (0.88), Posted Solution (1.00), Solution Justification (0.62). These are all above the indicated 0.6 coefficient values needed to suggest substantial results (Landis and Koch, 1977). However, to resolve conflicts, we initiated a discussion and refined codes if applicable. Given the level of agreement, after this process, all the data from one of the authors was used for labeling.

Table 5 shows the frequency and percentages of each information category in the *Question & Title* posts. The *#Instances* row gives the total number of posts where each category of information occurs in the posts, the *#Highlighted instances* and *%Highlighted instances* rows give the total number of instances and percentages of instances highlighted for each category of information across the data set. Each column indicates a category of information available in *Question & Title* posts. We split *Developer's (Dev's) Pertinent Action* into three columns: *Problematic (Prob.) code, Surrounding (Surr.) code*, and *Text*. For each instance, if any of these subcategories is highlighted, we add one to the final count of the information category (*Developer's Pertinent Action*). Similarly, we split *System Symptom* into two subcategories: *Stack trace* and *Text*, and if any one of these subcategories is highlighted, we add one to the count for *System Symptom*. For example, the first row indicates that *Developer's (Dev's) Goal* occurs in 375 of 400 (94%) question and title instances, and at least one of the subcategories of *Developer's Pertinent Action* were available in 380 (95%) instances, and at least one of the subcategories of *System Symptom* were available in 320 (80%) instances.

Analysis of Table 5 provides several key observations:

- While Developer's Goal and Developer's Pertinent Action categories of information appear in almost all of the question (95%) instances, the System Symptom category appears in 80% post instances. This is because some questioners were seeking information on how to possibly avoid certain errors/exceptions and did not explicitly mention their system response to an error/exception.
- All three categories of information that occur in the question component of a post (Developer's Goal, Developer's Pertinent Action, and System Symptom) were highlighted in about half of the instances; however, Developer's Pertinent Action and System Symptom were highlighted more frequently (53% and 58%, respectively) than Developer's Goal (44%). This is partly due to user preference, since we observed that 18 of our 50 participants consistently never highlighted Developer's Goal. These results indicate that our participants focus more on descriptions of the Developer's Pertinent Action and System Symptom when reading Stack Overflow posts.
- When the participants highlighted the Developer's Pertinent Action, they highlighted the Problematic Code most often, then the Text, and lastly the Surrounding code. These results indicate that the Problematic Code is considered most relevant by novice software engineers to understand the specific developer action that led to the error/exception, but they also believe that it is somewhat important to focus on the surrounding code and corresponding text description.
- Participants who highlighted the System Symptom highlighted both stack trace and text description of the System Symptom about the same frequency, both the majority of the time. This suggests that novice software engineers focus on stack trace information almost as much as the textual descriptions of symptoms.

Similar to Table 5, each row of Table 6 shows the frequency and percentages of information category in the *Answer* instances. We show separate counts for 'Accepted/Best voted' answers and 'Second' answers. The *Symptom Cause, Posted Solution*, and *Solution Justification* columns correspond to each category available in both types of *Answer* instances. We further split *Posted Solution* into two subcategories as *Code*, and *Text*. For each instance, if any of the subcategories is highlighted, we add one to the final count for the category (*Posted Solution*). For example, the first row for 'Accepted/Best voted Answers' shows that of 400 answer instances, *Symptom Cause* is available in 189 instances (47%), and either of the subcategories of *Posted Solution* is available in 362 instances (91%), and *Solution Justification* is present in 173 instances (43%).

Analysis of Table 6 provides several key observations:

- As expected, Posted Solution information occurs in almost all (91%) of the accepted answer instances. In the few cases that there is no Posted Solution, Symptom Cause information is offered instead. Information about the Symptom Cause occurs much less frequently (47%) and Solution Justification information even less frequently (43%) in accepted answer instances.

- In both accepted and second answer instances, the participants focused their attention primarily on the Posted Solution information with 76% and 50% highlights, respectively. The accepted answer highlights are much more frequent most likely because the participants have more confidence in answers that are indicated as helpful by the original questioner and other users.
- When participants highlighted Posted Solution information, they focused almost equally on code (69%) and text describing the Posted Solution (66%) in the accepted answer instances. There is a similar pattern for second answer instances with more equal focus between code and text descriptions. This is expected since novice software engineers are typically looking for some description to understand the code provided in the answers.
- Beyond the Posted Solution information, the participants focused more on the Symptom Cause than the Solution Justification, almost twice as often in the accepted answer instances and only slightly more often focus on Symptom Cause in the second answer instances.
- Descriptions of Solution Justification are not frequently highlighted, only 27 to 46% of the occurrences. Similar to Developer's Goal information, this is partly because of user preference, since we observed that 14 of 50 participants consistently never highlighted this information.

We also computed the frequency of each category in participants' highlights of comments in questions and answers. Comments to the 'Accepted/Best Voted Answers' are most frequently highlighted. The most frequent categories highlighted in comments to 'Accepted/Best Voted Answers' are Posted Solution, Symptom Cause and Solution Justification with 9 to 14% (20 to 32 of 229) instances highlighted for each category. Each of the remaining categories was highlighted in less than 10% (16 of 165) of the instances containing the information.

There were only a few other kinds of information in the participants' highlights, which we did not classify into one of the six categories. In the answers, this included links to helpful tutorials/developer blogs, text describing the assumptions and limitations of suggested solutions and phrases indicating whether the proposed solution works. In comments, they sometimes highlighted insightful follow-up questions. Any additional information was rarely highlighted in the questions. *Novice Software Engineer Reflections after Annotating:* The last data that we analyzed were the responses to the last two questions in the perceptions survey (Fig. 2) completed after the annotation task. Out of 50 participants, Table 7 shows the percentage of participant recommendations for each sub-question of RQ2. These categories are not necessarily the same as the categories of highlighted information. For *RQ2(a): When there is significant amounts of code in the post, which code components in the question/answer/comment would you recommend highlighting to focus your attention quickly?*, the most prevalent answers were problematic code in question and its context. For *RQ2(b): Which natural language text in the question/answer/comment do you think highlighting would help to focus your attention quickly?*, participants ranked the text in the question describing the error/exception, the text in the answer/comment describing the cause of the error/exception, and the text in answer/comment describing the suggested solution highest. None of the participants selected "Other" for any of these two questions.

4. Implications and recommendations

Based on our results, we provide implications for both improving Q&A forums for software engineers and potential improvements to Q&A mining-based tools to help software engineers. Our findings suggest that, while novice software engineers could read the entire Stack Overflow posts and answers that they are provided in the study, they highlighted only 27% of code and 16 to 21% of natural language text that could potentially help them to understand and determine how to apply the relevant information to their context. The small percentage of highlighting of relevant information suggests that designing a browser extension or a highlighting interface tool could be useful to help novice software engineers on these forums. Results also indicate some design considerations for such a tool. For example, the highlighting tool could ignore comments and second answers and set priority on other components of a post. Automatically classifying and highlighting each category of information by color would provide the novice software engineers with the option of adjusting their focus on specific types of information. Additionally, the categories of information identified by our qualitative analysis could be leveraged to improve search results on Stack Overflow. For example, if posts are clustered and ranked based on information category, it might be easier for novice software engineers to quickly browse through more posts and identify the most relevant one.

The small percentage of recommended highlights and the various categories could also potentially be leveraged by tools that mine Stack Overflow for various tasks. Depending on the mining tool's objective, it could filter out irrelevant content and mine only specific parts of a post based on our categories of information. For example, tools investigating common coding problems such as API

Table 7
Participant Reflections after annotation task .

Recommended Highlight	%Participant
Code	
Only the problematic code in question	36
Problematic code in question and its context	50
Entire solution code suggested in answer/comment	28
Only necessary code modifications suggested in answer/comment	39
Only suggested code in verified/accepted answers	39
Text	
Text in question describing developer's intended outcome	51
Text in question describing the error/exception	66
Text in answer/comment describing cause of the error/exception	63
Text in answer/comment describing suggested solution	63
Text in answer/comment describing why suggested solution works	35
Text clues in answer/comment pointing to suggested solution	25
A comment that reaffirm a solution's correctness	41
Programming environment dependencies	11
Coding best practices	15

misuse on Stack Overflow could extract and analyze code in 'Developer's Pertinent Action'.

5. Related work

To our knowledge, this is the first study on how to increase the efficiency of novice software engineers on Stack Overflow as they read a single post. Researchers have conducted studies of questions, answers, and the code examples embedded in Stack Overflow posts towards determining their quality in terms of helpfulness (Sillito et al., 2012), good and bad quality characteristics (Duijn et al., 2015; Baltadzhieva and Chrupala, 2015; Correa and Sureka, 2013; 2014), code correctness (Zhang et al., 2018), and understandability of code examples (Treude and Robillard, 2017). Others have studied Stack Overflow to determine developer behaviors (Wang et al., 2013; Xia et al., 2017; Greco et al., 2018; Treude et al., 2011) and topics of discussion (Rosen and Shihab, 2015; Ahmed and Bagherzadeh, 2018), to compare the kinds of information on Stack Overflow against information available in other kinds of artifacts (Chatterjee et al., 2017; 2019; Zagalsky et al., 2016), to gather opinions (Lin et al., 2018), and to determine what triggers questions (Aghajani et al., 2018). Others have applied topic analysis and mining of domain-specific information (Zou et al., 2015a; Villanes et al., 2017), exploring gender bias (Lin and Serebrenik, 2016; Morgan, 2017; Ford et al., 2016), and emotions (Novielli et al., 2014; 2015). We specifically study the behavior of novice software engineers on Stack Overflow, and investigate how they use their time and focus in identifying relevant information from the question and the two best answers provided to them in the study. We assume that the relevant post is already identified.

There have been several advances to improve Stack Overflow for the users, including recommending tags (Saha et al., 2013; Wang et al., 2014; Zhou et al., 2017), grouping semantically related tags (Wang et al., 2012; Tian et al., 2014; Chen et al., 2017; Beyer and Pinzger, 2015), adding and replacing improved embedded code snippets (Tavakoli et al., 2016), detecting duplicate questions (Zhang et al., 2015; Murgia et al., 2016; Ahasanuzzaman et al., 2016), improving quality post detection (Ponzanelli et al., 2014; Yao et al., 2015; Ponzanelli et al., 2014), predicting questions as they are posed as being quality that would result in deletion (Xia et al., 2016), and helping users reformulate queries for better search (Rahman and Roy, 2018). de Souza et al. (2014) developed a technique to help developers in searching content on Stack Overflow, by recommending a ranked list of question and answer pairs (instead of entire Q&A threads) based on their query. Recently, Phan et al. proposed an approach using statistical machine translation to resolve the fully qualified names (FQNs) for API elements in Stack Overflow code snippets (Phan et al., 2018). Zhang and Hou (2013) analyzed 200 threads from the Java Swing Forum and identified eight categories of sentences in a discussion thread ("Design-Goal", "How-to", "Question-of-code", "Neutral-Action", "Claims", "Neutral-Behavior", "Question-of-Behavior", and "Negative-Behavior"), which are related to the six information categories ("Developer's Goal", "Developer's Pertinent Action", "System Symptom", "Symptom Cause", "Posted Solution", and "Solution Justification") we identified in Stack Overflow posts. After identifying the categories of sentences, Zhang et al. also designed a technique to extract problematic API features from online developer discussions (Zhang and Hou, 2013), which in turn could potentially help to identify "hot topics" in a forum, highlight the negative sentiment sentence and its neighbors in a discussion to guide a reader's attention and speed up reading, and help form better search queries. Similar to how Zhang et al.'s work could be used to design a browser extension, our results can direct the design of a browser extension or a highlighting interface tool to speed up reading individual forum posts, and thus improve the experience of novice software engineers seeking help on Stack Overflow.

In context of analyzing error related posts, previous studies identified broader categorization of the content of forum posts such as specific challenges that programmers had about a software framework (Hou and Li, 2011). Mining information related to programming errors and exceptions from code examples on Stack Overflow is common. Among others, researchers have identified common error patterns (Nagy and Cleve, 2015), and API usage obstacles (Wang and Godfrey, 2013). A key mechanism for providing such mined information is through Integrated Development Environments (IDE) recommendations (Amintabar et al., 2015; Rahman et al., 2014; Cordeiro et al., 2012). Categories of information proposed in our paper are based on Stack Overflow posts related to errors and exceptions, and thus could possibly be leveraged by tools that mine Stack Overflow for related tasks.

Researchers have proposed several techniques to extract and summarize relevant information from software artifacts. Treude and Robillard (2016) designed an approach to automatically augment API documentation with insightful sentences from Stack Overflow. Wong et al. (2013), designed an automatic comment generation tool that extracts code segments along with their descriptions from Stack Overflow and leverages this information to automatically generate descriptive comments for similar code segments in open-source projects. Lotufo et al. (2012) proposed a bug report summarization technique that estimates a user's attention on different sentences in a bug report when pressed with time. Rastkar et al. (2014) proposed an extractive approach for automatic bug report summarization that uses a binary classifier to determine whether a comment should be selected or not. Xu et al. (2017) designed an approach to generate multi-answer-posts summary for a given technical question in Q&A forum. Di Sorbo et al. (2016) developed an approach to create summaries of mobile app reviews. The authors classify and group sentences in a user review to one of the user-intention categories ("Information Giving", "Information Seeking", "Feature Request", "Problem Discovery", and "Other"), and then use a sentence selection and scoring mechanism to generate the user review summary. Similar to Di Sorbo et al.'s user-intention categories, our proposed categories of information in Stack Overflow posts could possibly be leveraged for summarization or information extraction related tasks.

Beyond software artifacts, there has been significant contribution in summarizing various types of documents such as email, chat and meeting conversations (Mehdad et al., 2014; Zhou and Hovy, 2005; Wang and Cardie, 2013; Gillick et al., 2009). Marcu (1997) developed a discourse-based approach for text summarization by using Rhetorical Structure Theory (RST). Radev et al. (2000) proposed a multi-document summarization technique for news articles by using cluster centroids produced by a topic detection system. Rambow et al. (2004) worked on summarizing email threads by leveraging the dialogic structure of email communication. Carenini et al. (2008) summarized email conversations based on the conversational cohesion and the subjective opinions. Zhou and Hovy (2005) proposed an approach for summarizing IRC logs. They cluster the discussions according to subtopic structure and then generate summaries for each of the clusters by extracting adjacent pairs of user response through machine learning methods. Murray et al. (2010) summarized meeting documents according to a meeting ontology representing phenomena such as decisions, action items and sentiment.

6. Conclusions and future work

This paper presents results from an exploratory study to understand Stack Overflow novice software engineers' challenges in pinpointing their needed information within a selected post, and

their perceptions of the parts of the post, highlighting of which would help to focus attention to increase their efficiency. Our findings suggest that, while novice software engineers could read the entire Stack Overflow posts and answers that they are provided in the study, they highlighted only 27% of code and 16 to 21% of natural language text that could potentially help them to understand and determine how to apply the relevant information to their context. Further, the relevant (highlighted) parts on which they focus attention can be categorized as primarily the System Symptom and Developer's Pertinent Action, and then the Developer's Goal within the question component, and in the answer, mainly the Posted Solution and Symptom Cause and to a lesser extent, the Solution Justification.

Since this study is focused on posts related to errors and exceptions in Java/C++, we plan to investigate the generalizability of the proposed categories of information to other kinds of posts on Stack Overflow. We also plan to explore the feasibility of automatically identifying and classifying the categories of information in a Stack Overflow post, and conduct a case study of information seekers using a highlighting tool based on these results.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Science Foundation [grant numbers 1813253, 1422184].

References

Aghajani, E., Nagy, C., Bavota, G., Lanza, M., 2018. A large-scale empirical study on linguistic antipatterns affecting apis. In: 34th International Conference on Software Maintenance and Evolution (ICSME'18).

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K., Schneider, K.A., 2016. Mining duplicate questions in stack overflow. In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, New York, NY, USA, pp. 402–412. doi:10.1145/2901739.2901770.

Ahmed, S., Bagherzadeh, M., 2018. What do concurrency developers ask about?: alarge-scale study using stack overflow. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, New York, NY, USA, pp. 30:1–30:10. doi:10.1145/3239235.3239524.

Amintabar, V., Heydarnoori, A., Ghafari, M., 2015. ExceptionTracer: a solution recommender for exceptions in an integrated development environment. In: Proc. IEEE 23rd Int'l Conf. on Program Comprehension, pp. 299–302. doi:10.1109/ICPC.2015.45.

Asaduzzaman, M., Mashiyat, A.S., Roy, C.K., Schneider, K.A., 2013. Answering questions about unanswered questions of stack overflow. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 97–100.

Baltadzhieva, A., Chrupala, G., 2015. Question quality in community question answering forums: a Survey. SIGKDD Explor. Newsl. 17 (1), 8–13. doi:10.1145/2830544.2830547.

Beyer, S., Macho, C., Pinzger, M., Di Penta, M., 2018. Automatically classifying posts into question categories on stack overflow. In: Proceedings of the 26th Conference on Program Comprehension. ACM, New York, NY, USA, pp. 211–221. doi:10.1145/3196321.3196333.

Beyer, S., Pinzger, M., 2015. Synonym Suggestion for Tags on Stack Overflow. In: 2015 IEEE 23rd International Conference on Program Comprehension, pp. 94–103. doi:10.1109/ICPC.2015.18.

Carenini, G., Ng, R.T., Zhou, X., 2008. Summarizing emails with conversational cohesion and subjectivity. In: Proceedings of ACL-08: HLT. Association for Computational Linguistics, Columbus, Ohio, pp. 353–361.

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

Chatterjee, P., Nishi, M.A., Damevski, K., Augustine, V., Pollock, L., Kraft, N.A., 2017. What information about code snippets is available in different software-related documents? an exploratory study. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 382–386. doi:10.1109/SANER.2017.7884638.

Chen, C., Xing, Z., Wang, X., 2017. Unsupervised Software-specific Morphological Forms Inference from Informal Discussions. In: Proceedings of the 39th International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 450–461. doi:10.1109/ICSE.2017.48.

Cordeiro, J., Antunes, B., Gomes, P., 2012. Context-based Recommendation to Support Problem Solving in Software Development. In: Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 85–89.

Correa, D., Sureka, A., 2013. Fit or Unfit: Analysis and Prediction of 'Closed Questions' on Stack Overflow. In: Proceedings of the First ACM Conference on Online Social Networks. ACM, New York, NY, USA, pp. 201–212. doi:10.1145/2512938.2512954.

Correa, D., Sureka, A., 2014. Chaff from the Wheat: Characterization and Modeling of Deleted Questions on Stack Overflow. In: Proceedings of the 23rd International Conference on World Wide Web. ACM, New York, NY, USA, pp. 631–642. doi:10.1145/2566486.2568036.

Di Sorbo, A., Panichella, S., Alexandru, C.V., Shimagaki, J., Visaggio, C.A., Canfora, G., Gall, H.C., 2016. 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. ACM, New York, NY, USA, pp. 499–510. doi:10.1145/2950290.2950299.

Duijn, M., Kučera, A., Bacchelli, A., 2015. Quality questions need quality code: classifying code fragments on stack overflow. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 410–413.

Explorer, S. E. D., 2019. https://data.stackexchange.com//.

Ford, D., Smith, J., Guo, P.J., Parnin, C., 2016. 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. ACM, New York, NY, USA, pp. 846–857. doi:10.1145/2950290.2950331.

Gillick, D., Riedhammer, K., Favre, B., Hakkani-Tur, D., 2009. A global optimization framework for meeting summarization. In: 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 4769–4772. doi:10.1109/ICASSP.2009.4960697.

Gottipati, S., Lo, D., Jiang, J., 2011. Finding relevant answers in software forums. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, Washington, DC, USA, pp. 323–332. doi:10.1109/ASE.2011.6100069.

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

Hou, D., Li, L., 2011. Obstacles in using frameworks and apis: an exploratory study of programmers' newsgroup discussions. In: 2011 IEEE 19th International Conference on Program Comprehension, pp. 91–100. doi:10.1109/ICPC.2011.21.

Landis, J., Koch, G., 1977. The measurement of observer agreement for categorical data. Biometrics 33 (1), 159–174. doi:10.2307/2529310.

Li, Z., Wang, T., Zhang, Y., Zhan, Y., Yin, G., 2016. Query reformulation by leveraging crowd wisdom for scenario-based software search. In: Proceedings of the 8th Asia-Pacific Symposium on Internetware. ACM, New York, NY, USA, pp. 36–44. doi:10.1145/2993717.2993723.

Lin, B., Serebrenik, A., 2016. Recognizing gender of stack overflow users. In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, New York, NY, USA, pp. 425–429. doi:10.1145/2901739.2901777.

Lin, B., Zampetti, F., Bavota, G., Di Penta, M., Lanza, M., Oliveto, R., 2018. Sentiment analysis for software engineering: how far can we go? In: Proceedings of the 40th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 94–104. doi:10.1145/3180155.3180195.

Lotufo, R., Malik, Z., Czarnecki, K., 2012. Modelling the ?hurried? bug report reading process to summarize bug reports. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 430–439. doi:10.1109/ICSM.2012.6405303.

Marcu, D., 1997. The rhetorical parsing of natural language texts. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 96–103. doi:10.3115/976909.979630.

Mehdad, Y., Carenini, G., Ng, R.T., 2014. Abstractive summarization of spoken and written conversations based on phrasal queries. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Baltimore, Maryland, pp. 1220–1230. doi:10.3115/v1/P14-1115.

Morgan, S., 2017. 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. ACM, New York, NY, USA, pp. 39–41. doi:10.1145/3135932.3135952.

Murgia, A., Janssens, D., Demeyer, S., Vasilescu, B., 2016. Among the machines: human-bot interaction on social q&a websites. In: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems. ACM, New York, NY, USA, pp. 1272–1279. doi:10.1145/2851581.2892311.

Murray, G., Carenini, G., Ng, R., 2010. Generating and validating abstracts of meeting conversations: A user study. In: Proceedings of the 6th International Natural Language Generation Conference. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 105–113.

Nagy, C., Cleve, A., 2015. Mining stack overflow for discovering error patterns in SQL queries. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 516–520. doi:10.1109/ICSM.2015.7332505.

Nie, L., Jiang, H., Ren, Z., Sun, Z., Li, X., 2016. Query expansion based on crowd knowledge for code search. IEEE Trans. Serv. Comput. 9 (5), 771–783. doi:10.1109/TSC.2016.2560165.

Novielli, N., Calefato, F., Lanubile, F., 2014. Towards discovering the role of emotions in stack overflow. In: Proceedings of the 6th International Workshop on Social Software Engineering. ACM, New York, NY, USA, pp. 33–36. doi:10.1145/2661685.2661689.

Novielli, N., Calefato, F., Lanubile, F., 2015. The challenges of sentiment detection in the social programmer ecosystem. In: Proceedings of the 7th International Workshop on Social Software Engineering. ACM, New York, NY, USA, pp. 33–40. doi:10.1145/2804381.2804387.

Phan, H., Nguyen, H.A., Tran, N.M., Truong, L.H., Nguyen, A.T., Nguyen, T.N., 2018. Statistical learning of api fully qualified names in code snippets of online forums. In: Proceedings of the 40th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 632–642. doi:10.1145/3180155.3180230.

Ponzanelli, L., Mocci, A., Bacchelli, A., Lanza, M., Fullerton, D., 2014. Improving low quality stack overflow post detection. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 541–544. doi:10.1109/ICSME.2014.90.

Radev, D.R., Jing, H., Budzikowska, M., 2000. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. NAACL-ANLP 2000 Workshop: Automatic Summarization.

Rahman, M., Yeasmin, S., Roy, C., 2014. Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. In: Proc. IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering, pp. 194–203. doi:10.1109/CSMR-WCRE.2014.6747170.

Rahman, M.M., Roy, C.K., 2018. Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In: 34th International Conference on Software Maintenance and Evolution (ICSME'18).

Rambow, O., Shrestha, L., Chen, J., Lauridsen, C., 2004. Summarizing email threads. In: Proceedings of HLT-NAACL 2004: Short Papers. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 105–108.

Rastkar, S., Murphy, G.C., Murray, G., 2014. Automatic summarization of bug reports. IEEE Trans. Softw. Eng. 40 (4), 366–380. doi:10.1109/TSE.2013.2297712.

Rosen, C., Shihab, E., 2015. What are mobile developers asking about? a large scale study using stack overflow. Empir. Softw. Eng. 21, 1192–1223. doi:10.1007/s10664-015-9379-3.

Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples, 1st Wiley Publishing.

Saha, A.K., Saha, R.K., Schneider, K.A., 2013. A discriminative model approach for suggesting tags automatically for stack overflow questions. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 73–76. doi:10.1109/MSR.2013.6624009.

Sillito, J., Maurer, F., Nasehi, S.M., Burns, C., 2012. What makes a good code example?: a study of programming Q&A in StackOverflow. In: Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington, DC, USA, pp. 25–34. doi:10.1109/ICSM.2012.6405249.

S.O.D.S., 2018. https://insights.stackoverflow.com/survey/2018.

de Souza, L.B.L., Campos, E.C., Maia, M.d.A., 2014. Ranking crowd knowledge to assist software development. In: Proceedings of the 22nd International Conference on Program Comprehension. ACM, New York, NY, USA, pp. 72–82. doi:10.1145/2597008.2597146.

Stemler, S. E., 2004. A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability.

Tavakoli, M., Heydarnoori, A., Ghafari, M., 2016. Improving the quality of code snippets in stack overflow. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1492–1497. doi:10.1145/2851613.2851789.

Tian, Y., Lo, D., Lawall, J., 2014. Automated construction of a software-specific word similarity database. In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), pp. 44–53. doi:10.1109/CSMR-WCRE.2014.6747213.

Treude, C., Barzilay, O., Storey, M.-A., 2011. How do programmers ask and answer questions on the web? (nier track). In: Proceedings of the 33rd International Conference on Software Engineering. ACM, New York, NY, USA, pp. 804–807. doi:10.1145/1985793.1985907.

Treude, C., Robillard, M.P., 2016. Augmenting API documentation with insights from stack overflow. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 392–403. doi:10.1145/2884781.2884800.

Treude, C., Robillard, M.P., 2017. Understanding stack overflow code fragments. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 509–513. doi:10.1109/ICSME.2017.24.

Villanes, I.K., Ascate, S.M., Gomes, J., Dias-Neto, A.C., 2017. What are software engineers asking about android testing on stack overflow? In: Proceedings of the 31st Brazilian Symposium on Software Engineering. ACM, New York, NY, USA, pp. 104–113. doi:10.1145/3131151.3131157.

Wang, L., Cardie, C., 2013. Domain-independent abstract generation for focused meeting summarization. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Sofia, Bulgaria, pp. 1395–1405.

Wang, S., Lo, D., Jiang, L., 2012. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 604–607. doi:10.1109/ICSM.2012.6405332.

Wang, S., Lo, D., Jiang, L., 2013. An empirical study on developer interactions in stackoverflow. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, New York, NY, USA, pp. 1019–1024. doi:10.1145/2480362.2480557.

Wang, S., Lo, D., Vasilescu, B., Serebrenik, A., 2014. EnTagRec: an enhanced tag recommendation system for software information sites. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 291–300. doi:10.1109/ICSME.2014.51.

Wang, W., Godfrey, M.W., 2013. Detecting API usage obstacles: a study of iOS and android developer questions. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 61–64.

Wong, E., Yang, J., Tan, L., 2013. AutoComment: mining question and answer sites for automatic comment generation. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 562–567. doi:10.1109/ASE.2013.6693113.

Xia, X., Bao, L., Lo, D., Kochhar, P.S., Hassan, A.E., Xing, Z., 2017. What do developers search for on the web? Empir. Softw. Engg. 22 (6), 3149–3185. doi:10.1007/s10664-017-9514-4.

Xia, X., Lo, D., Correa, D., Sureka, A., Shihab, E., 2016. It takes two to tango: deleted stack overflow question prediction with text and meta features. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 1, pp. 73–82. doi:10.1109/COMPSAC.2016.145.

Xu, B., Xing, Z., Xia, X., Lo, D., 2017. Answerbot: automated generation of answer summary to developersź technical questions. In: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 706–716.

Yao, Y., Tong, H., Xie, T., Akoglu, L., Xu, F., Lu, J., 2015. Detecting high-quality posts in community question answering sites. Inf. Sci. 302 (C), 70–82. doi:10.1016/j.ins.2014.12.038.

Zagalsky, A., Teshima, C.G., German, D.M., Storey, M., Poo-Caamaño, G., 2016. How the R community creates and curates knowledge: a comparative study of stack overflow and mailing lists. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 441–451. doi:10.1109/MSR.2016.052.

Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., Kim, M., 2018. Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow. In: Proceedings of the 40th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 886–896. doi:10.1145/3180155.3180260.

Zhang, Y., Hou, D., 2013. Extracting problematic api features from forum discussions. In: 2013 21st International Conference on Program Comprehension (ICPC), pp. 142–151. doi:10.1109/ICPC.2013.6613842.

Zhang, Y., Lo, D., Xia, X., Sun, J.-L., 2015. "Multi-Factor duplicate question detection in stack overflow". J. Comput. Sci. Technol. 30 (5), 981–997. doi:10.1007/s11390-015-1576-4.

Zhou, L., Hovy, E., 2005. Digesting virtual "geek" culture: the summarization of technical Internet relay chats. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05). Association for Computational Linguistics, Ann Arbor, Michigan, pp. 298–305. doi:10.3115/1219840.1219877.

Zhou, P., Liu, J., Yang, Z., Zhou, G., 2017. Scalable tag recommendation for software information sites. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 272–282. doi:10.1109/SANER.2017.7884628.

Zou, J., Xu, L., Guo, W., Yan, M., Yang, D., Zhang, X., 2015. An empirical study on stack overflow using topic analysis. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 446–449.

Zou, Y., Ye, T., Lu, Y., Mylopoulos, J., Zhang, L., 2015. Learning to rank for question-oriented software text retrieval. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 1–11. doi:10.1109/ASE.2015.24.

Preetha Chatterjee is a Ph.D. student in the Department of Computer and Information Sciences at the University of Delaware. Her research interest is in improving software engineers development environments through mining and text analysis of software artifacts to gain information from other developers experiences. https://sites.udel.edu/preethac/

Minji Kong is an undergraduate student pursuing a Computer Science honors degree at the University of Delaware with minors in Interactive Media and Art. Her current interests include Software Analysis, Data Mining, Human-Computer Interaction, and Computer Science education. https://www.linkedin.com/in/minjikong/

Lori Pollock is Alumni Distinguished Professor in the Department of Computer and Information Sciences at the University of Delaware. Her research currently focuses on program analysis for building better software maintenance tools, software testing, energy efficiency of software, and computer science education. https://www.eecis.udel.edu/~pollock/