What Help Do Developers Seek, When and How?

Hongwei Li[1], Zhenchang Xing[2], Xin Peng[1], Wenyun Zhao[1]
[1]School of Computer Science, Fudan University, Shanghai, China
[2]School of Computer Engineering, Nanyang Technological University, Singapore
{lihongwei, pengxin, wyzhao}@fudan.edu.cn
zcxing@ntu.edu.sg

***Abstract*—Software development often requires knowledge beyond what developers already possess. In such cases, developers have to seek help from different sources of information. As a metacognitive skill, help seeking influences software developers' efficiency and success in many situations. However, there has been little research to provide a systematic investigation of the general process of help seeking activities in software engineering and human and system factors affecting help seeking. This paper reports our empirical study aiming to fill this gap. Our study includes two human experiments, involving 24 developers and two typical software development tasks. Our study gathers empirical data that allows us to provide an in-depth analysis of help-seeking task structures, task strategies, information sources, process model, and developers' information needs and behaviors in seeking and using help information and in managing information during help seeking. Our study provides a detailed understanding of help seeking activities in software engineering, the challenges that software developers face, and the limitations of existing tool support. This can lead to the design and development of more efficient and usable help seeking support that helps developers become better help seekers.**

***Index Terms*—Help Seeking, Human Study, Process Model**

I. INTRODUCTION

Software engineering is a knowledge-intensive activity. Today's software developers must use diverse technologies and complex frameworks to ensure a high quality of their work products and to maximize their productivity. As such, software development often requires knowledge beyond what developers already possess. To obtain the knowledge necessary to complete the task at hand, software developers have to seek help from co-workers, project documentations, books, manuals, and digital information resources.

Help seeking is an important metacognitive skill (i.e., knowing about knowing) [22]. It includes knowledge about when and how to use particular strategies for learning or for problem solving. Before the widespread use of the Internet, software developers' help seeking activity was significantly dependent on communication with co-workers [12]. The prevalence of online resources has fundamentally changed the way software developers find and use help.

The availability of open source projects (e.g., sourceforge.net, github) and the popularization of online technical manuals (e.g., MSDN, Java tutorial), technical blogs (e.g., codeproject.com, iteye.com), and socio-professional media (e.g., StackOverflow) contribute to a fast-growing body of online knowledge for software development, which software developers have perceived as their "key information resource". The ability to search, understand, and use this online knowledge is one of the key abilities affecting software engineers' efficiency and success [26].

Human's help seeking behavior, such as process of help seeking and human and system factors affecting help seeking, has been extensively studied in social contexts, such as classrooms and interactive learning environments [1][10]. In the context of software engineering, a significant amount of literature exists aiming to understand software developers' information behavior and needs [15][29][30], cognitive process [16][23][8], work habit [9] in various types of software engineering tasks (e.g., debugging, program exploration, reengineering). When it comes to research on help seeking the focus has been on code search engines [3] and recommender systems [14][20]. However, there has been little research to provide a systematic investigation of software developers' help seeking behavior in terms of what help developers seek, when and where they seek help and how.

This paper reports our empirical study aiming to fill this gap. Our study includes two human experiments. The first experiment involves 11 developers and the task is to develop a socket-based peer-to-peer chat software from scratch. The second experiment involves 13 developers and the task is to maintain an existing Eclipse plugin (including two subtasks: fixing two bugs in the plugin and extending the plugin with additional features).

Our study gathers empirical data to answer the following questions regarding help-seeking activities in software engineering:

- In which situation do developers seek help? What do they look for? What strategies do they use?
- Where do developers search? Do they have preferences in different situations?
- Are there distinct phases in help seeking? How do developers start, proceed, and finish the task? What factors affect the process of help seeking?
- What strategies do developers use for managing information during help seeking process? What are the challenges they face?

Our study provides a detailed understanding of both static and dynamic perspectives of help seeking activities in software engineering. The static perspective includes help-seeking task structure, task strategies, and information sources. The dynamic perspective includes process model and developers' information needs and behavior in finding and

using help information as well as in managing information during help seeking. It also identifies the challenges that software developers face and the limitations of existing tool support. Our study results can lead to design and development of more human-centric, context-sensitive, and holistic help-seeking solutions for finding and using online knowledge.

The remainder of the paper is organized as follows. Section II reviews related work. Section III describes the design of our study. Section IV reports our study results. Section V discusses insights and lessons learned from our results. Section VI discusses threats to our study. Finally, we conclude and outline our future plan.

II. Related Work

Help seeking has been extensively studied in pedagogy and instructional design [22][10][1]. It has been recognized as an important metacognitive skill and self-regulated behavior. A variety of learner characteristics (e.g., priori knowledge [31], self-assessment [22]) influence help seeking individually or in combination. Furthermore, a variety of system factors (e.g., context sensitivity, domain-specific representation, quality of text and multimedia information) may cause different types of help seeking activities and result in different learning outcomes [1]. These sociological studies provide a general framework for the design and analysis of our study.

Researchers have investigated software developers' information behavior and needs [15][21], cognitive process [8][23][16], work habit [9] in writing, changing and debugging software. Their goal is to elicit requirements for more efficient software development environments and tools. Umarji et al. [27] investigate developers' code searching practices using anecdotal online questionnaire. These studies focus mainly on Internet-scale code search and reuse practice. In contrast, our study investigates help seeking activities in software engineering which include but are not limited to code reuse. Furthermore, existing studies examine only the static perspective of code reuse practice, such as search motivations, information sources, and size of reused code. Our study provides an in-depth quantitative and qualitative analysis of both static and dynamic perspectives of help seeking in software engineering.

Code search engines [3] have been proposed to provide domain-specific search services for developers. Recommender systems [14][20] have been proposed to address the information overloading problem in software engineering. To leverage prevalent online knowledge for software development (e.g., Stack Overflow), several tools [2][3] have been proposed to integrate code search and crowdsourced knowledge. While useful in some ways, existing tools support mainly search activities in software development. Our study shows that developers spend much more time on selecting, integrating and verifying help information than searching. However, select, integration, and verification activities are under studied and under supported.

Existing work studies and supports search, select, integration and verification individually. Holmes et al. [13] proposed that end-to-end support is needed for reusing code examples. Our study confirms their observation. In addition, our study reveals the very dynamic and iterative nature of help seeking processes. This dynamic and iterative nature calls for more holistic and flexible support for help seeking than a simple pipeline-style tool chain.

III. Experiment Design

We now describe the design of our human experiments, participants, tasks, and development environments. Note that our experiment design was based on the survey of existing literature and the discussion of a focus group that includes three authors and three developers with different background, experience and skills. This literature survey and focus group discussion also provided guideline for the analysis of empirical data collected in our experiments.

A. Overview

Table 1 summarizes our study. Our literature survey and focus group discussion identified two typical software development tasks that may differ in two important ways: developing new software versus maintaining existing software, and reusing library APIs versus framework-based plugin development. Thus, we designed two experiments in our study, involving 11 and 13 developers (recruited from our graduate school in Fudan University) respectively. The Experiment1 task is to develop a chat software based on socket APIs, while the Experiment2 task is to maintain an existing Eclipse editor plugin.

Table 1 An Overview of Our Study

	Experiment1	**Experiment2**
Pre-Survey	Yes	Yes
Tutorial	Yes	Yes
#Participants	11	13
Experiment Task	Develop a P2P Chat Software	Maintain an Eclipse editor plugin
Length	2 hours	2 hours
Help-seeking task types	Feature, API, error (maybe)	Feature, API, error, configure
Post-Ques.	Yes	Yes

Furthermore, our literature survey and focus group discussion identified four types of help seeking tasks:1) feature-oriented for implementing new features or extending existing features; 2) API-oriented for reusing library or framework APIs; 3) configuration-oriented for configure system or framework environments; and 4) error-oriented for fixing compilation or runtime errors. The Experiment2 task covers all the four types of identified help-seeking tasks, while the Experiment1 task covers feature- and API-oriented and (potentially) error-oriented help seeking.

Before each experiment, we conducted a pre-study survey of developers' development experiences. Furthermore, we offered a short tutorial to explain task requirements and some technical details such as socket port range, Eclipse plugin development basics.

As the objective of our study is to investigate help-seeking behavior and process in software engineering, we put less emphasis on the delivery of working software in our study. Thus, we asked participants to spend up to two hours on each

task in Experiment1 and Experiment2. They were allowed to submit their results anytime no matter whether or not they completed the task. We believe this helps to elicit "true" help seeking behavior rather than help seeking behavior biased by the concern of completing the assigned task on time.

It is important to note that during the experiments we had no constraints on what kinds of help developers can seek, when and where they can seek help and how. This is because we want to be more opportunistic and let interesting help seeking behavior emerge through the experiments, rather than limit our study to some preconceived areas of focus.

Because we need to analyze behavior and process of each participant in completing the assigned task, we required the participants to run a full-screen recorder once they started working on the tasks. The screen recorder recorded all screen activities of participants' working computers and discussions between participants (sound quality varies depending on the distance to the participants' working computers).

We also encouraged participants to "thinking aloud" [28] during the experiments for important actions and strategies Furthermore, after experiment participants were asked to fill out a questionnaire immediately as their memory was still fresh. This post-experiment questionnaire asked information sources where developers seek help, and important actions, intentions and strategies during their help seeking process. This questionnaire provides a qualitative overview of developers' task completion process. Think-aloud hints and post-experiment questionnaires are important to understand developers' behavior, especially their intentions and strategies, when we analyze and transcribe task videos.

B. Experiment Tasks

Our study included two software development tasks. Our design goal was to elicit all four types of help seeking behaviors in our experiments. To that end, we designed software development and maintenance tasks that are likely to incur different types of help seeking.

The first task we designed is to develop a P2P chat software. The software should support the following features: 1) set and display user name, 2) connect to a given IP address, 3) after establishing connection, send text messages to and receive text messages from the given IP address, 4) record and display chat history. To implement these features, developers may need to perform feature-oriented help seeking. Furthermore, developers have to use thread APIs, socket APIs, and use certain GUI framework (e.g., Eclipse SWT or Java Swing) in their implementation. This may incur API-oriented help seeking. Developers also need to determine and choose proper socket ports for client-server communications. Because socket ports below 8000 are often used by other programs, choosing an already-in-use ports will result in a runtime error, which may in turn incur error-oriented help seeking.

The second task is to maintain an existing Eclipse editor plugin. This task includes two subtasks. The first subtask is to fix two bugs in the existing implementation. First, the plugin's configuration file contains an error. The class attribute of the editor extension point is incorrect (i.e., using a wrong class name). Due to this bug, when the user attempts to open a new editor, nothing happens. After fixing this bug, when the user attempts to open a new editor, the plugin throws an IllegalArgumentException. Fixing these two bugs incurs not only API-oriented help seeking but also error-oriented and configuration-oriented help seeking. After developers fix bugs and open a new editor successfully, the second subtask asks developers to extend an existing editor plugin with file open/save/close features and file content statistics (e.g., word count) feature. Implementing these new features requires developers use proper Eclipse interfaces and extension points (e.g., view or status bar). As such, the second subtask will incur feature-oriented, API-oriented, and configuration-oriented help seeking.

C. Participants

In this study, we recruited 24 graduate students from our school in Fudan University. 11 students participated in Experiment1, another 13 different students participated in Experiment2. We used students as participants because they can represent junior developers who need help the most during software development. Based on our pre-study survey, we rated participants' programming experience at three levels in terms of lines of code they have developed and their a priori knowledge of specific libraries or frameworks that need to be used in the task.

In Experiment1, two developers had developed over 50K lines of code (LOC), and they had experience in socket programming. We considered them as A-level experience. Eight developers had developed 5K-50K LOC. None of them had socket-programming experience. We considered them as B-level experience. One developer had developed less than 5K LOC and he had no socket-programming experience. We considered him as C-level experience.

In Experiment2, two developers had developed over 50,000 LOC. One of them had Eclipse plugin development experience, while the other did not. We considered these two developers as A-level experience. Eight developers had developed 5,000-50,000 LOC. We considered them as B-level experience. The rest three developers had developed less than 5,000 LOC. We considered them as C-level experience. Three B-level developers had understood the basics and five B-level and all C-level developers had no experience in Eclipse plugin development.

D. Experiment Environment

Our experiments were conducted in our software engineering lab. Participants were allowed to use their preferred working computer (laptop or PC). All computers used Windows 7 or newer. They all installed full screen recorder software. Their development environments were Eclipse3.6 (or newer) or MyEclipse8.0 (or newer), and JDK1.6.30 (or newer). During the experiment, we allowed participants to discuss with other participants. The goal is to simulate a more natural working environment in which developers can discuss with their co-workers. Note that the screen recorder software we used can record participants' discussions when they are close enough to their working computers through sound recording.

IV. RESULTS

In this section, we discuss both quantitatively and qualitatively the empirical data that we collected in our study. Our literature survey and focus group discussion suggest that we should analyze help seeking activities from two perspectives, i.e., static perspective including help seeking tasks and information sources, and dynamic perspective including help seeking process and developers' information needs and behavior in the process.

Our analysis is based on about 35 hours of task videos and 24 post-experiment questionnaires collected in the two experiments. In Experiment1, the 11 developers spent on average 92±19 minutes on the development task. Two developers delivered a working chat software that supports all required features by the experiment task description, while the rest nine developers delivered a partially-featured chat software. In Experiment2, the 13 developers spent 84±24 minutes on the maintenance task. Three developers delivered a fully-featured editor plugin as required by the experiment task description; eight developers fixed bugs in the plugin and delivered a partially-featured editor plugin; two developers failed to fix bugs in the plugin and thus did not implement additional plugin features.

We watched each developer's task video and transcribed it into a sequence of working sessions. For each working session, we recorded session duration, help-seeking tasks in the session (feature-, API-, error- or configuration-oriented), information sources being explored (e.g., technical blogs, API documents), and important actions and intentions of these actions (e.g., search on Google, select useful online resources, integrate online examples, verify resulting program). We used recorded discussions, think-aloud hints and questionnaires responses to assist the analysis of task videos. We stored video transcripts into a database. We analyzed these video transcripts to answer questions regarding help seeking activities in software engineering (see Section I for a list of questions that we investigated).

A. Help Seeking Tasks and Information Sources

We first discuss our analysis from static perspective of help seeking, i.e., help seeking task structure, task strategies, and information sources.

1) Task Structure: Our analysis of task videos of the two experiments suggests that different types of help-seeking tasks are not independent of one another. Instead, they have rich semantic relations that correspond well to work breakdown structure [25] of a software development task. We model help-seeking tasks and their relations using a metamodel (see Figure 1).

Users usually described software system with features which are observable to the users [7]. A composite feature can comprise some subfeatures. Thus, a feature-oriented help seeking may be composed of help seeking activities for subfeatures. To implement a feature, developers usually need to use libraries and frameworks. For example, in Experiment1, to specify an IP address, developers may use GUI framework components such as Java Swing's JTextField. To connect to the given IP address, they need to use socket APIs. In Experiment2, developers need to implement the abstract class EditorPart for saving file edits, and extend either view or status bar extension points for showing file statistics. As such, developers may have to seek help in programing with library/framework APIs with proper parameters and/or configuration (i.e., API-oriented and configuration-oriented help seeking).

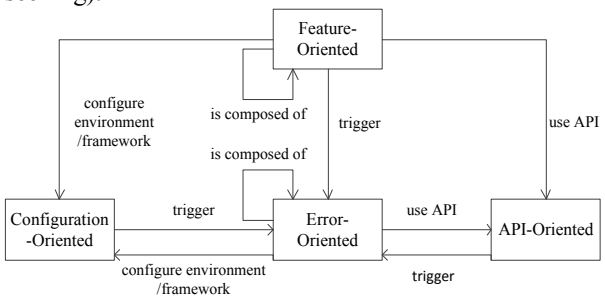

Figure 1 Help Seeking Task Metamodel

TABLE 2 PREFERENCES OF HELP SEEKING STRATEGIES

	Use online crowdsourced knowledge	*Use formal technical documentations*	*Solve based on previous experience*	*Consult experts*
Experiment1				
Configuration-oriented	27.27%	45.45%	9.09%	18.18%
Feature-oriented	63.64%	18.18%	9.09%	9.09%
Error-oriented	45.45%	0.00%	45.45%	9.09%
API-oriented	27.27%	63.64%	9.09%	0.00%
Experiment2				
Configuration-oriented	53.85%	46.15%	0.00%	0.00%
Feature-oriented	69.23%	15.38%	15.38%	0.00%
Error-oriented	46.15%	0.00%	46.15%	7.69%
API-oriented	15.38%	76.92%	7.69%	0.00%

Due to misuse of APIs or misconfiguration of frameworks, developers may encounter errors. For example, in Experiment1, a TCP port has already been used. In addition to errors generated in development process, errors may also be reported by users. For example, in Experiment2, a new editor cannot be successfully opened using Eclipse editor plugin. These errors will trigger error-oriented help seeking. Note that an error-oriented help seeking may be composed of several help seeking activities for suberrors. For example, the cannot-open-new-editor error in Eclipse editor plugin is caused by two bugs in the plugin implementation. To fix an error, developers may have to seek help in programing with library/framework APIs with proper parameters and/or configuration. For example, to fix the two bugs in Eclipse editor plugin, developers need to learn how to properly configure and use Eclipse editor extension points and relevant APIs.

Our analysis shows that help seeking is an integral part of software development. In the course of completing a software development task, developers may need to seek help to achieve the objective of the task when they do not possess knowledge necessary to complete the task.

2) Task Strategy: Help seeking has been considered as a metacognitive skill [22] for knowing when and how to use particular strategies for learning or for problem solving. To understand developers' help seeking strategy for different types of help seeking tasks, we analyzed post-experiment questionnaires for recurring strategies using open coding [18] and a grounded theory approach [24].

Our analysis identified four types of help-seeking strategies. Table 2 summarizes the percentage of participants in the two experiments who adopted different help-seeking strategies for different types of help-seeking tasks. Online crowdsourced knowledge refers to online code examples, technical blogs, forum discussions, and Q&A posts. Our further task video analysis suggests that developers usually use search engine (e.g., Google, Baidu) to find online crowdsourced knowledge that may satisfy their learning needs (see Section IV.A.3).

Formal technical documentations refer to API specifications, books, technical manual. Note that most of such technical documentations are available online nowadays. But technical documentations are different from online crowdsourced knowledge in that technical documentations are formally produced by software producers, while crowdsourced knowledge is implicitly generated by a community of users.

Our analysis shows that developers may solve problems based on their previous experience and knowledge. Questionnaire responses indicate that these developers still need to seek help from previous projects and online resources. But our further task video analysis suggests that their goal is to recall and confirm technical details. As such, help-seeking behavior of developers who solve problems based on previous knowledge is different from that of developers learning unfamiliar knowledge (see Section IV.B.1).

Developers still consult experts for different types of help. Our questionnaire responses and task video analysis suggest that the way that developers consult experts become more virtual. For example, developers may contact experts using social media (such as Skype). Furthermore, experts also become more virtual; they may no longer be somebody that developers know of. For example, developers can post a question to a community of developers using online Q&A service such as Stack Overflow. Both questions and replies will become online crowdsourced knowledge that can be further used by many others in communities of practice [4].

In Table 2, our questionnaire data suggests that using online crowdsourced knowledge is the most commonly adopted strategy for feature-oriented help seeking. For API-oriented help seeking, developers prefer the most formal technical documentation. To fix errors, developers prefer to use online crowdsourced knowledge or solve the problems based on their previous experience. None of our participants seek help in formal technical documentations for fixing errors. Technical documentations are usually designed for correct usage scenarios; they may offer little help in dealing with erroneous scenarios. For configuration-oriented help seeking, developers prefer to use crowdsourced knowledge and technical documentations.

3) Information Sources: We analyzed participants' task videos to identify specific information sources where developers find useful information or knowledge for completing the assigned tasks. Our analysis identified 10 types of useful information sources. Table 3 summarizes how many times developers in the two experiments find useful information in specific information sources. The values in brackets are average times per developer.

TABLE 3 STATISTICS OF USEFUL INFORMATION SOURCES

	Experiment1	Experiment2
Technical Blogs	32 (2.9)	65 (5)
doc sharing sites	14 (1.3)	20 (1.54)
Q&A web sites	14 (1.3)	6 (0.5)
API spec	4 (0.36)	17 (1.3)
Forum discussions	7 (0.64)	9 (0.69)
Code example sites	7 (0.64)	4 (0.31)
Expert	4 (0.36)	2 (0.15)
Previous projects	5 (0.46)	0
Online tutorials	0	4 (0.31)
Code hosting sites	0	2 (0.15)

Overall, technical blogs (e.g., iteye.com, blog.163.com) are the most useful information sources for both tasks in the two experiments. Next in the list, document sharing web sites (e.g., wenku.baidu.com, www.360doc.com) and Q&A sites (such as zhidao.baidu.com) are useful information sources for the development of the chat software in Experiment1, while document sharing web sites and API specifications are useful information sources for the maintenance of Eclipse editor plugin in Experiment2.

In our two experiments, developers find code hosting and code example web sites less useful than other information sources. In Experiment1, none of the eleven developers used code hosting web sites, and one developer used code example web site seven times. In Experiment2, two of the 13 developers used cost hosting web site in total twice, and another two developers used code example web sites in total four times. Our interviews with participants suggest that they found those code hosting or code example web sites provide mostly code-oriented information. However, when they seek help for a software engineering task, they often need more knowledge-oriented information that can help them understand techniques in code examples.

Our task video analysis reveals that developers heavily rely on search engines (such as Google, Baidu, and Bing) to find useful technical blogs, forum discussions, and Q&A posts. Only one participant in Experiment1 did not use search engine. This developer had developed a similar communication project before. He completed the task by reusing code from his previous project. One participant in Experiment2 used search engine only once and opened two technical blogs. This developer is very experienced in Eclipse plugin development. He quickly visited the two technical blogs to recall and confirm some technical details. Except these two participants, developers in Experiment1 used search engines 4.31±2.10 times, while developers in Experiment2 used search engines 6.60±2.92 times.

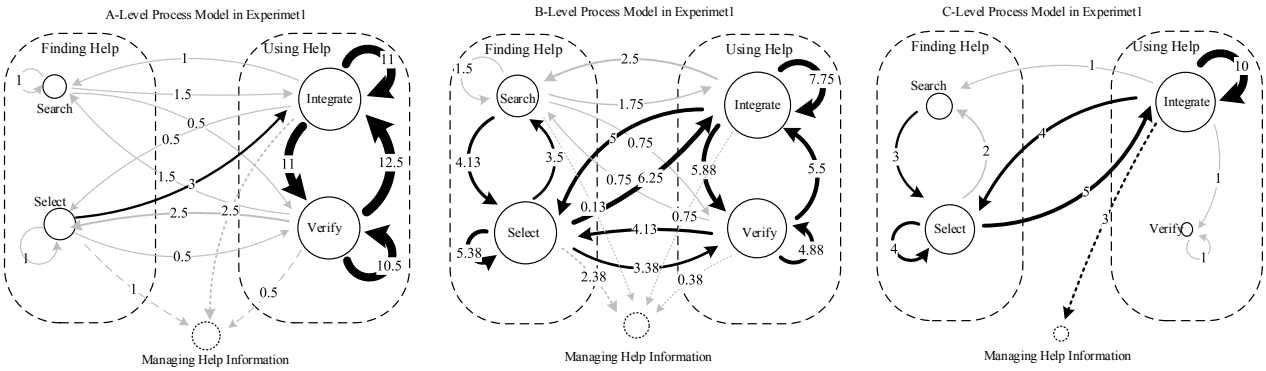

Figure 2 Process Models of Different Levels of Developers in Experiment1

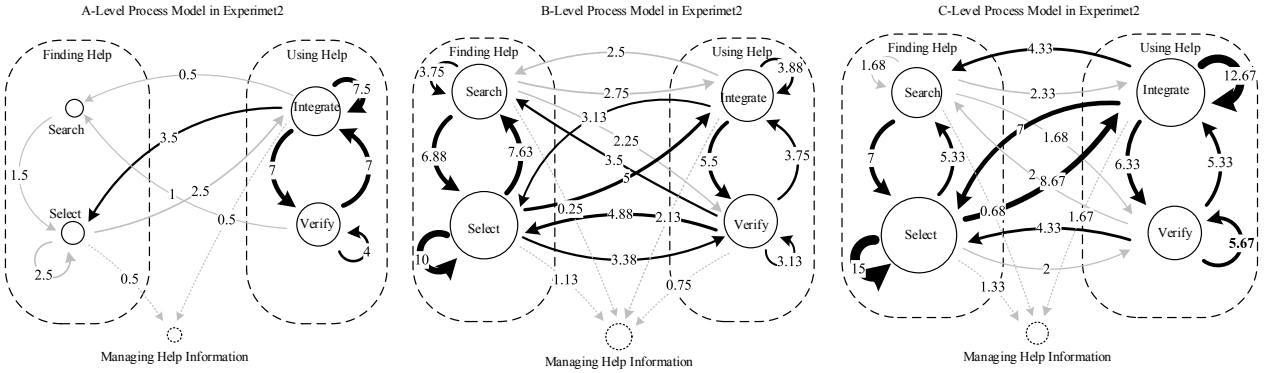

Figure 3 Process Models of Different Levels of Developers in Experiment2

In our experiments, a developer may seek help from various information sources. Each developer on average used 4.82±1.59 (means ± standard deviation) information sources in Experiment1 and 4.69±1.77 information sources in Experiment2. Our analysis suggests that B-level developers use information sources most diversely (5.38±1.11 in Experiment1 and 5.25±1.56 in Experiment2 information sources), followed by C-level developers (4±0 in Experiment1 and 4.67±1.70 in Experiment2), and A-level developers (3±2 in Experiment1 and 2.5±0.5 in Experiment2). A plausible explanation could be A-level developers already possess the necessary knowledge, while C-level developers do not know of as many types of information sources as B-level developers.

B. Help Seeking Process

To understand the dynamic perspective of help seeking in software engineering, we analyzed task video transcripts for distinct phases using open coding [18] and a grounded theory approach [24]. We identified five distinct phases, i.e., search help information, select potentially useful information, integrate help information, verify help information or integration results, and manage help information. In this section, we first discuss the help seeking process model. We then elaborate on the five phases and their transitions.

1) Process Model: Figure 2 and Figure 3 show process models of A-level, B-level, and C-level developers in the two experiments respectively. Graph nodes represent distinct phases that we identified. The area of a node represents average times that the corresponding phase occurs in the course of completing the assigned task for a particular level of developers. Graph edges represent transitions from one phase to another. The thickness and grayscale of an edge represents the average number of corresponding transitions that happened in the course of completing the assigned task for a particular level of developers.

It is clear that help seeking does not follow a sequential or waterfall process. In contrast, it is iterative and dynamic by nature. The Search and select phase is to find useful help information to solve the problem. The Integration and verification phase is to actually use help information to solve the problem. The most common transitions from finding help to using help are from the select phase to the integration phase. However, transitions between other phases are also frequently present (see Section IV.B.2)). All these transitions constitute an iterative and dynamic process of help seeking.

Note that we had only one C-level developer in Experiment1. This developer performed 4 times of search on Google. He found a high-quality web page about socket programming. Thus, he spent only 12% of his working time on searching and selecting help information. Then, he spent most (88%) of his working time on learning this web page and integrating the code example in the page into his project. He performed some simple verification at the end of integration. This developer is not representative for C-level developers. But his task completion process also shows the importance of good search engine and high-quality information sources. If a less experienced developer can find such high-quality

information sources quickly, he may complete relevant software development task smoothly.

Overall, A-level developers spent most of their efforts on integrating and verifying help information. They performed more integration and verification actions than search and select actions. B-level developers spent similar efforts on finding help information and using the information. C-level developers spent most of their efforts on selecting and integrating help information.

B-level developers had the most dynamic and iterative help seeking process. They may transit from one phase to any other three phases during help seeking. A-level developers performed more iterative integration and verification. C-level developers transit back and forth mainly between select and integration phases.

It is interesting to note that developers may transit to the manage-information phase from all four other phases. This is especially the case for B-level developers as they tend to use more information sources than other levels of developers (see Section IV.A.3)).

2) Finding Help: Searching and Selecting: The process models in Figure 2 and Figure 3 show that developers of different experience levels show different information needs and behavior in finding help information.

Because A-level developers know what information they need and where to find it, they can often quickly find the needed information and determine its relevance. For example, an A-level developer in Experiment1 imported a similar communication project that he has developed before and quickly found reusable parts that can be integrated in the assigned task. In Experiment2, an A-level developer entered a precise query on search engine and then opened two highly relevant technical blogs. He quickly read through the blogs to refresh his memory and then started fixing plugin bugs. As such, the help seeking process of A-level developers shows very few numbers of search and select phases and very few search-select iterations. These experienced developers spent only about 11% in Experiment1 and 16% in Experiment2 of their working time on searching and selecting help information.

Compared with A-level developers, B-level and C-level developers in our experiment relied heavily on crowdsourced knowledge and search engines. They performed more search-select iterations to search, compare, assess and learn online knowledge. As such, B-level and C-level developers spent more time on finding help information, about 24% in Experiment1 and 36% in Experiment2 of working time of B-level developers, and about 12% in Experiment1 and 42% in Experiment2 of working time of C-level developers.

After finding useful help information, developers in our Experiments proceeded to use the found information in several ways. The most common way (about 55% in Experiment1 and 46% in Experiment2 of all transitions from Find Help to Use Help) is to integrate the selected information (i.e., Select→Integrate) to solve the problem at hand. Developers may also integrate search results directly (i.e., Search→Integrate, about 15% in Experiment1 and 19% in Experiment2 of all Find→Use transitions). This usually happens in API-oriented help seeking. Finally, Developers may not directly integrate the found information. In contrast, they may verify the information in a separate project (i.e., Search→Verify, Select→Verify, about 31% in Experiment1 and 36% in Experiment2 of all Find→Use transitions) before they integrate it into their solution. This verification step can be considered as extended select or pre-integration to better determine the quality of help information.

Note that searching help information is not always as straightforward as expected, especially for error-oriented or configuration-oriented help seeking, because the system may not give enough hints about what went wrong or what needs to be done. For example, the buggy Eclipse editor plugin in Experiment2 cannot open a new editor properly, but it does not give any clear error message about the incorrect configuration value. This makes it very difficult to start help seeking as developers were clueless about what went wrong.

3) Using Help: Integrating and Verifying: Developers of different experience levels also show different information needs and behaviors in using help information. A-level developers spent 84% in Experiment1 and 89% in Experiment2 of their working time on integrating and verifying help information, while B-level developers spent about 76% in Experiment1 and 64% in Experiment2 of their working time on integration and verification, C-level developers spent 88% in Experiment1 and 58% in Experiment2 of their working time on integration and verification. Furthermore, A-level developers performed more iterative integration-verification cycles than B-level and C-level developers.

Developers adopt different strategies for integrating help information. In both Experiment1 and Experiment2, one developer is very familiar with the assigned tasks. Thus, they directly write and modify the code of the assigned tasks with no or little search and selection of help information. Basically, they completed the tasks based mainly on the knowledge they possess. The rest 22 developers in our experiments adopt copy-paste-modify practices to reuse code or solutions found in help information. If code or solutions in help information is similar to what developers need but cannot be directly reused, developers will develop their solutions by referencing the found help information. In our experiments, we found two developers who performed refactoring after integrating help information in order to improve code maintainability and achieve better design.

The process models in Figure 2 and Figure 3 show that while developers integrate and verify help information, they often go back to search and/or select phases. First, developers may perform incremental integration. They need to go back to search or select more information. This accounts for 59% in Experiment1 and 50% in Experiment2 of all transitions from Use Help to Find Help. Second, developers may encounter errors during integration or verification of help information. For example, if developers use an already-in-use socket port for socket connection, they will encounter runtime error of port collision. To fix such errors, developers need to go back

to the original help information to investigate what he may miss. Or they may start a new search and/or select alternative solutions. Or they may initiate a new error-oriented help seeking process for the port collision error.

4) Managing Help Information: Our empirical data suggest that help seeking is not only about finding and using help information. It is also about managing a large amount of help information during help seeking. We identified two places where effective information management strategies are needed.

First, the quality of help information, especially online crowdsourced knowledge varies greatly. Search engines may return seemingly different but essentially duplicated information (e.g., reposted or similar answers). During the select phase, developers need to manage such low-quality, irrelevant, or repetitive information. We observed two strategies for managing help information during the select phase. Some developers in our experiments threw away such information immediately. They opened a web page, quickly read through its content, and closed the page if they consided the page content useless or duplicate. We observed 28 times such open-immediate-close actions by 10 developers. Other developers tended to open multiple web pages at the same time. They then inspected opened web pages one by one and closed useless or duplicated pages after inspection. We observed 40 times such actions by 12 developers.

Second, after developers integrate help information and verify integration results, they may close relevant web pages, documentations and/or source files. Or developers may verify the quality of online code examples in a separate project. After they confirm the usefulness of those code examples, they may close or even delete this experimental project.

We found that managing help information goes beyond closing opened webpages or files. Developers also need effective screen management during integration of help information into their projects. Help information that developers find is usually separated from IDEs. As a result, developers have to work with multiple windows. In our experiments, developers usually opened about 3-5 task-related windows, which often include IDE, task description editor, and browsers in the course of completing the task. Thus, keeping only the most relevant information on the screen affects efficiency of integration.

Furthermore, developers also need to keep track of help information he or she finds or uses during a software engineering task. We observed 17 times re-finding or reopening closed web pages, API documentations, or source files by nine developers. For example, one developer in Experiment2 closed the web page about Eclipse plug-in configuration after he or she integrated a code example in the web page into his project. Later on he found he still needs this webpage for file open/save/close features, especially for save feature. Then, he had to re-find this webpage using search engine again.

V. DISCUSSION

Conducting this study has given us interesting insights into the challenges that software developers face during help seeking activities and the limitations of existing tool support. We discuss them in this section. We also propose how help seeking in software engineering can be better supported.

A. Understanding, Monitoring and Using Working Context

Our study shows that help seeking is an integral part of software development tasks and thus should be studied and supported in the larger context of software development. However, existing tools for finding and using help information are largely agnostic of developers' working context. For example, although developers rely heavily on search engines and online resources, current search tools are separate from software development environments and thus cannot make use of developers' working context. This separation also incurs the difficulties in information and screen management during help seeking.

Several recent research efforts [3][2] propose to embed search tools in the development environments. They can leverage the developers' working context to reduce the cost of constructing a good query, which can improve search result quality. These solutions are limited because they use only simple program context. However, help seeking happens in a more complex application, environment, and user context. For example, existing solutions can offer little help in finding useful solution for the configuration error in Experiment2 task. Furthermore, existing solutions consider context as a static set of facts providing the background for search. However, our analysis of help seeking process suggests that context is dynamically changing as the current context induces actions which give rise to further contexts in the iterative help seeking process.

Furthermore, our study also shows that help seeking goes far beyond a search task. Help seeking requires browsing through several alternatives, comparing and assessing them, and possibly selecting and integrating solutions from several information sources in a very dynamic and iterative process. Unfortunately, contextual information has not been exploited to support help seeking phases other than search.

Our study results call for more holistic understanding, monitoring and utilization of developers' working context in which help seeking occurs. We need to better understand types, scope and observability of contextual information as well as their utility in meeting developers' information needs in seeking and using help information during software development. This better understanding will allow us to better monitor and react to developers' information needs and working status as they work. For example, we may perform implicit search to proactively find and deliver the information developers need when they need it as developers integrate information from multiple sources.

In addition to assisting search, we may also use contextual information to better augment the presentation of search results to allow developers to evaluate results more rapidly, reducing the cost of selecting a good result. Furthermore, monitoring developers' working context allows us to keep track of developers' help seeking history. This history can help to refind useful help information. More important, it provides an opportunity for more effective information and

screen management by implicitly linking or grouping help information immediately relevant to developers' task.

B. Supporting Peer-to-Peer Architecture for Help Seeking

Our study shows that the help seeking process is dynamic and iterative. However, this dynamic and iterative nature has not been widely appreciated. Existing tools usually support each phase individually. Holmes et al. [13] proposed that end-to-end support is needed for reusing code examples. Our study suggests that a pipeline style tool chain may not work effectively for help seeking. First, phases may not be easily separated from each other, for example search-select or integrate-verify. Second, help seeking rarely goes through an ideal sequence of phases, from search, select, integrate to verify. Instead, developers can go to any other phases from a given phase depending on feedbacks collected in the phase.

An effective help seeking tool must take into account all phases as a whole and also feedback loops among phases. A peer-to-peer architecture in which each phase is both service requestor and provider can better support iterative process of help seeking. This peer-to-peer architecture allows phases to be integrated more flexibly.

For example, our study shows that search engines often return links to duplicated contents that waste time to examine and provide no new information. Furthermore, developers have to click through a list of web pages to compare and assess similar ones. A better search-select integration could automatically analyze search results using clustering and program analysis (e.g., clone analysis, software differencing) techniques. This can provide a high-level structure over search results and also highlight their commonalities and differences, reducing the cost of selecting a good result. Furthermore, we may also monitor developers' selections of information sources and issues they encounter during integration and verification phases. Such contextual information may be used as feedbacks to enable search engine to return results of increasing utility.

As another example, our study shows that developers spend more time on integration and verification than on searching and selecting. One particular reason is that developers have to reconcile many inconsistent names in code snippets. A smart integration tool should support auto-complete and undo/redo mechanism so that developers can easily embed online code snippets in a new context while maintaining parameter names and styling conventions. This smart auto-complete mechanism may also be used to speculatively integrate search result [19] to identify results with least integration problems. This allows developers to better select help information based on not only relevance but also potential integration efforts needed. It also allows developers to be more experimental in selecting alternative solutions and to spend more time on verifying the resulting programs instead of fixing tons of compilation errors.

C. Helping Develpoers Become Better Help Seekers

Our study shows that developers frequently used online crowdsourced knowledge for completing software development tasks. Several recent research tools [2] start integrating online crowdsourced knowledge (e.g., Stack Overflow) into the development environments. Others enable developers to share their work (e.g., navigation data [6], execution trace [11], modification of copied online code snippets [17]) directly from the development environment.

These tools allow developers to leverage the body of knowledge on socio-professional media as well as create opportunities for others. They make it easier for developers to share and find useful software artifacts such as code examples and bug fixes. However, software artifacts provide only a finished view. When it comes to learning in working the very act of doing a job provides real opportunities to learn and develop. This may explain why in our experiments developers prefer technical blogs over code hosting web sites. High-quality blogs often provide detailed background knowledge and step-by-step instructions. This makes it easier to select and integrate help information.

This result suggests that we need a new paradigm that allows developers to share not only software artifacts but also their behavior information, i.e., how they produce these software artifacts. Cloud-based IDEs such as Cloud9 IDE [5] may provide infrastructure to easily gather not only artifacts but also behavior information of developers. Wisdoms of crowds may then be mined from mass artifact and behavior data. This cloud-based infrastructure can also support unobtrusive monitoring of developers' working context.

This new paradigm can support communities-of-practice [4] that allow developers to pick up not just information ("know what") but also invaluable "know now" for solving a specific problem. Furthermore, it provides opportunities to help developers become better help seekers. As a metacognitive skill [1], our study shows that help seeking is not just about solving a specific problem, it is also about learning when and how to use a particular strategy to find useful information for problem solving. This new paradigm may help to record how a community of developers seek and use help information for similar problems. Such help seeking metaknowledge may provide guided navigation that leads developers to the information they need as quick as possible.

VI. THREATS TO VALIDITY

A major threat to the internal validity of our empirical study is that many findings in this study were based on our subjective interpretations of participants' task videos and post-experiment questionnaires. The complexity of developers' actions during help seeking may incur errors in our analysis, especially for quantitative analysis of developer's help seeking process and their information behavior and needs in finding and using help. The other threat to the internal validity of the study is the evaluation of developers' program experience. This may threaten the analysis of the impacts of human factors on help seeking.

A major threat to the external validity of the study is that we studied developers' work in controlled experiments instead of a real-world context. Although we designed two realistic tasks, the complexity of the software to be developed or maintained may limit the generalizability of our findings.

Furthermore, due to the limited number of developers and their limited diversity, our study is anecdotal and exploratory by nature. Further studies are required to generalize our findings with professional developers, industrial systems and real software engineering tasks.

VII. Conclusion and Future Work

This paper reported our empirical study of help seeking activities in software engineering. Our study investigated both static and dynamic perspectives of help seeking in software development and maintenance. Our study showed that 1) help seeking is an integral part of software development. It must be understood and supported in the larger context of software development; 2) help seeking involves dynamic and iterative searching, selecting, integrating, verifying, and managing help information. An effective help seeking tool must consider all these activities as a whole and also feedback loops among these activities; 3) help seeking is not just about useful information (i.e., know what"). It is also about invaluable "know-how" for solving the problems. A cloud-based communities-of-practice may provide opportunities to help developers become better help seekers. Inspired by the study results, we will investigate cloud-based, human-centric, context-sensitive environments and tools for supporting help seeking in software engineering.

Acknowledgement

This work is supported by National High Technology Development 863 Program of China under Grant No.2013AA01A605, and by NTU SUG M4081029.020.

References

[1] V. Aleven, E. Stahl, S. Schworm, et al., "Help seeking and help design in interactive learning environments," Review of Educational Research, vol. 73, no. 3, 2003, pp. 277-320.

[2] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the IDE," RSSE, 2012, pp. 26-30.

[3] J. Brandt, M. Dontcheva, M. Weskamp et al., "Example-centric programming: Integrating web search into the development environment," CHI, 2010, pp. 513-522.

[4] J. S. Brown and P. Duguid, "Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation," Organization Science, vol. 2, no. 1, 1991,pp. 40-57.

[5] Cloud9 IDE, https://c9.io.

[6] R. DeLine, M. Czerwinski, and G. Robertson, "Easing program comprehension by sharing navigation data," VLHCC, 2005, pp.241-248.

[7] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," Journal of Software: Evolution and Process, vol. 25, no. 1, 2013, pp. 53-95.

[8] S. D. Fleming, C. Scaffidi, D. Piorkowski, et al., "An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks" ACM Transactions on Software Engineering and Methodology, vol. 22, no. 2, 2013,pp. 14:1-14:41.

[9] L. Freund, E. G. Toms, and J. Waterhouse, "Modeling the information behaviour of software engineers using a work-task framework," ASIST, vol. 42, no. 1, 2005.

[10] S. N. Gall, "Help-seeking behavior in learning," Review of Research in Education, vol. 12, 1985, pp. 55-90.

[11] Z. Gu, E. T. Barr, D. Schleck, and Z. Su, "Reusing debugging knowledge via trace-based bug search," OOPSLA, 2012, pp. 927-942.

[12] M. Hertzum and A. M. Pejtersen, "The information-seeking practices of engineers: Searching for documents as well as for people," Information Processing & Management, vol.36, no.5, 2000, pp.761-778.

[13] R. Holmes, R. J. Walker, and J. Denzinger, "The end-to-end use of source code examples: An exploratory study," ICSM, 2009, pp.555-558.

[14] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," ESEC/FSE-13, 2005, pp. 237-240.

[15] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," ICSE, 2007, pp. 344-353.

[16] S. Letovsky, "Cognitive processes in program comprehension," Journal of Systems and Software, vol. 7, no. 4, 1987, pp. 325-339.

[17] T. Lieber and R. Miller, "Programming with everybody: Tightening the copy-modify-publish feedback loop," UIST Adjunct, 2012, pp. 101-102.

[18] M. B. Miles and A. M. Huberman, Qualitative Data Analysis: An Expanded Sourcebook, Sage Publications, Inc., 1994.

[19] K. Muşlu, Y. Brun, R. Holmes, et al., "Speculative analysis of integrated development environment recommendations," OOPSLA, 2012, pp. 669-682.

[20] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, et al., "Graph-based pattern-oriented, context-sensitive source code completion," ICSE 2012, pp. 69-79.

[21] C. Parnin and S. Rugaber, "Programmer information needs after memory failure," ICPC, 2012, pp. 123-132.

[22] M. Puustinen, "Help-seeking behavior in a problem-solving situation: Development of self-regulation," European Journal of Psychology of Education, vol. 13, no. 2, 1998, pp. 271-282.

[23] M. Storey, F. D. Fracchia, and H. A. Müller, "Cognitive design elements to support the construction of a mental model during software exploration," Journal of Systems and Software, vol. 44, no. 3, 1999, pp. 171-185.

[24] A. C. Strauss and J. M. Corbin, Basics of Qualitative Research: Grounded Theory Procedures and Techniques, SAGE Publications, Inc., 1990.

[25] R. C. Tausworthe, "The work breakdown structure in software project management," Journal of Systems and Software, vol. 1, 1980, pp. 181-186.

[26] C. Tenopir and D. W. King, Communication Patterns of Engineers, Wiley-IEEE Press, 2004.

[27] M. Umarji, S. E. Sim, and C. Lopes, "Archetypal internet-scale source code searching," Open Source Development, Communities and Quality, vol. 275, 2008, pp. 257-263.

[28] M. W. Van Someren, Y. F. Barnard, and J. A. C. Sandberg, The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes, Academic Press London, 1994.

[29] J. Wang, X. Peng, Z. Xing, and W. Zhao, "An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions," ICSM, 2011, pp. 213-222.

[30] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving feature location practice with multi-faceted interactive exploration," ICSE, 2013, pp. 762-771.

[31] H. Wood and D. Wood, "Help seeking, learning and contingent tutoring," Computers & Education, vol.33,no.2,1999, pp.153-169.