

When Programmers Don't Ask

Sukanya Ratanotayanon
Department of Informatics
University of California, Irvine
sratanot@uci.edu

Susan Elliott Sim
Department of Informatics
University of California, Irvine
ses@ics.uci.edu

Abstract

Throughout the software development process, participants of the project need to collaborate in order to exchange the knowledge required to complete the project. Exchanging and obtaining knowledge is often done through asking and answering questions. We present an initial study aimed at understanding question-asking behavior during knowledge exchange in software development. We found that this seemingly simple activity is often not performed well, nor as frequently as required. Novices do particularly poorly as they are not aware of their knowledge needs. Experts also asked few questions but focused on different kinds of knowledge. In addition, they sometimes, ask questions although they have ability to obtain information themselves. We speculate on the causes of failures in question asking and the rationale behind experts' questions.

1. Introduction

Software development is a knowledge-intensive activity. Various stakeholders who are involved in the project need to collaborate and communicate in order to exchange and transfer their knowledge. However, collaboration and knowledge exchange in software projects are often not performed effectively. As reported by Curtis [1], the most common and severe issues in software development projects are the thin spread of domain knowledge, and communication and coordination breakdowns. These issues need to be addressed in order to improve overall development process performance.

A common way to exchange knowledge is through asking and answering questions. However, in order to effectively perform this seemingly simple activity, the participants need to be aware of what knowledge others have and their needs for knowledge. In addition, as observed in other fields such as education, it is not

unusual that a person does not ask questions because he lacks the information or does not understand the given explanation. If this behavior is also present in software development, it could decrease the effectiveness of knowledge exchange.

In order to support knowledge collaboration, we need to understand the question-asking behavior: i) how and why people ask questions and ii) what kind of information is needed, including its importance to the inquirer's knowledge needs. The answer to these questions can provide guidance for designing collaboration tools, such as what kinds of information is needed but is not requested, or support for novices to ask the right questions.

In this paper, we report on an initial exploratory study in a laboratory setting. The data reported here comes from a multi-part study in which novice and expert software engineers are required to collaborate on a change request for a moderate-sized web application. Our goal is to explore the question-asking behavior in knowledge transfer. We observed how software developers ask questions in a collaboration session to increase their understanding of the application and to complete an assigned task.

We found breakdowns in question asking. Our results showed that developers do not ask questions well, nor do they ask as often as they should. Novices did especially poorly as were not aware of their knowledge needs and didn't ask questions when they really should have. Experts also asked few questions, but focused on knowledge from different levels of abstraction. In addition, they sometimes, asked questions that they have ability to answer themselves.

This paper is organized as follows. Results from related research are reviewed in Section 2. The laboratory procedure used in the study is described in Section 3. The results are presented in Section 4. The discussion of the causes of the failures in question asking is presented in Section 5. Section 6 presents our future work. We conclude our paper in Section 7.

2. Related Work

Previous studies in question-asking fall into two main categories: psychological studies and field studies of question-asking in software development and engineering.

In psychology, questions tend to be used as an indicator of cognitive activity. Questions are viewed as a means to obtain knowledge that is important for the inquirer to reach a certain goal [2, 3]. Ram suggested that questions are crucial and central to learning [3]. The question formulation process is the process of identifying what the learner needs to learn, and asking the right questions allows the learner to focus on relevant issues by pursuing the questions. Therefore, the depth of understanding of the learner depends on the questions asked. However, as shown in experiment performed by Miyake and Norman [4] a prerequisite for asking questions about a new topic is an appropriate level of knowledge with which to formulate the question and to interpret the response. The number of questions a person asked when learning new material depended on two variables: i) the existence of a proper knowledge structure and ii) the level of completeness of those structures regarding new material.

Although, the work discussed above gives us an insight into question asking as a cognitive activity, it doesn't address what questions are asked and how they are actually asked in a work situation. The following are field studies investigating questions asked in software development and engineering.

Berlin performed field study of consulting interaction between apprentices and experts [5]. The results showed that question-asking played an important role in collaborative conversation between apprentices and experts. Confirmative questions were used by experts to invite apprentices' interjections. Apprentices used questions that restated the explanation to signal their level of understanding and to ask for validation or help. This collaborative process was important to providing a successful explanation, because the pair continually sought and provided evidence that they understood each other, which resulted in rapid repairs of misunderstandings. Berlin also found that experts were quicker to seek help from other experts, which might be due to better self-monitoring skill or social factors, such as having more reciprocal relationship with other experts.

Herbsleb et al. performed a field study aimed to assess knowledge needs in software development by examining the questions asked in requirement specification and design meetings [6]. Data was collected from projects in requirements and early

design phases. The results showed that the most common questions were "what" and "how" questions targeting requirements, even for the project that already move into design phase. Very few "why" questions were asked although design rationale [7] is considered very important information.

Ahmed and Wallace studied queries made by novice and experienced designers in a large aerospace company. Similar to Herbsleb, the goal of the study was to identify knowledge needs of designers and their awareness of their knowledge needs. The results showed differences between novices and experts in both types of queries made and patterns of responses to the queries. In addition, the finding indicated that novice designers tended to be unaware of their knowledge needs and required support in identifying what they needed to know.

3. Empirical Method

The goal of our study was to gain an understanding of question-asking behavior as a means of knowledge transfer in software development projects. We observed how software developers asked questions to increase their understanding of the application in order to complete an assigned task.

3.1. Research Design

While quantitative studies use experimental methods and quantitative measures to make predictions and generalize findings, qualitative studies use a naturalistic approach to build understanding and extrapolate to similar situations [8]. In order to gain a better understanding of question-asking behavior, we chose to perform an exploratory qualitative study instead of a quantitative study which ignores effects that may be important, but are not statistically significant. Qualitative methods provide a wider understanding of the entire situation as it accepts the complex and dynamic quality of the social world. It provides results that are more in-depth and comprehensive than those produced by quantitative methods.

We performed a laboratory study simulating a situation in software maintenance. An existing developer has to transfer his knowledge to assist another programmer who is also assigned to make a modification to a software system. This allows us to evaluate the quality and relevance of questions asked by subjects. More detailed information of study design is presented elsewhere [9].

3.2. Procedure

Two subjects were required to participate in each session. Each session took a total of 210 minutes and comprised three tasks: Task A, Handover and Task B. The time line of each session is depicted in Figure 1.

	0:30	1:00	1:30	2:00	2:30	3:00	3:30
Subject A	Task A			Handover			
Subject B				Handover	Task B		

Figure 1: Time line of study procedure

Task A: The first subject was given a scenario where a customer requested a feature in the company’s survey management application. He was asked to complete a Change Request Proposal (CRP) form describing how to make the change. The CRP gives the guideline of what information should be provided to Subject B. The subject was also asked to not make any modification and had up to 90 minutes to finish this task individually.

Handover: In this task, Subject A verbally handed over to Subject B the information gathered in the first task. The Handover task began with Subject A giving an explanation without any interruptions. During the explanation, the application and its code might be shown to Subject B to improve his understanding. Once the explanation was completed, Subject B was allowed to ask questions. The subjects were given 30 minutes to perform this task.

Task B: Following the Handover, Subject B was left with the CRP and Subject A’s notes. Subject B had to work individually and make the modification within 90 minutes.

This division of tasks not only allows us to examine the collaboration behavior of asking question, it also mimics common work situations where research is separated from detailed work. The CRP is a commonly used process in which software evolution is managed by a Change Control Board (CCB).

Before performing the tasks, both Subjects were given a short description of typical architecture of web applications, task description and instructions on running and compiling the application. Subjects were allowed to use any information available on the Internet. There was no application developer documentation, such as a design document. However, this omission is not uncommon in real-world maintenance settings.

During each session, each subject’s activities were recorded by a web camera, a microphone and screen capturing software. Scratch paper was provided to the participants and was collected at the end of the study.

Eclipse IDE containing a project set up for the task and TextPad software were also provided.

3.3. Software characteristics

The application used in this study was an open-source web-based survey management tool called VTSurvey, developed at Virginia Tech. It is a typical n-tier web application created with JSP, Java, and XML technologies, and runs on the Tomcat application server. It enables users to create, maintain and run online surveys. It also provides a user management system for managing user accounts. Originally, VTSurvey didn’t maintain each user’s email address. We requested that the system be modified so that it can save and display user’s email addresses.

VTSurvey consists of 38 Java™ files, 74 JSP (Java Server Page) files. In addition, there were 4 DTD (Document Type Definition) files. It has a total of 10,342 lines of code. Subjects were presented with all source files, including those that were not relevant to the assigned task.

3.4. Subjects

The subjects were mainly recruited by word of mouth. A total of twelve subjects participated in the study. Half the subjects were novices and the other half experts. Novices were senior undergraduates, or recent graduates who had been working for less than one year. Experts were developers with five or more years of work experience. We had three female subjects and all of them were experts. All subjects considered themselves fluent in English and had experience working with Java. We also surveyed their experience (including non-work experience) in the areas of web development and database management. Figure 2 presents overall level of experience of subjects in each group.

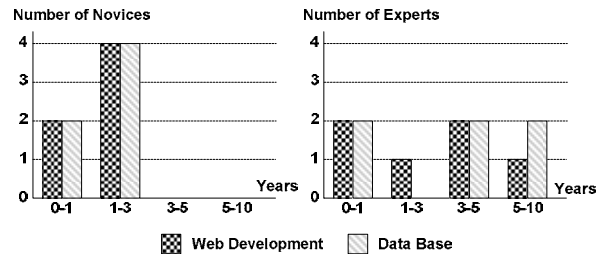


Figure 2: Level of experience in Web Dev. and DB

4. Analysis and Results

We analyzed audio and video data recorded during the six sessions. The implementation was graded based on its correctness and completeness regarding our requirements and was given a numeric score out of 55. The conversations during Handover sessions were transcribed in order to identify questions from both subjects. We included both implicit and explicit questions in our analysis. Explicit questions were formed in question sentences. The implicit questions were formed as normal sentences or fragments, but acted as questions due to the inquirer's expression and tone, and generated answers from the partner. The full list of questions is found in Table 3 and Table 5.

4.1. Number of Questions

We identified only total of 64 questions from the six sessions. Due to time constraints on the implementation task, we expected that Subject B would ask a lot of questions in order to take advantage of Subject A's knowledge. However, there were surprisingly few questions asked, in comparison with other studies [10, 11]. More than half of these questions were asked in a single session, the fourth one. In addition, the purpose of the questions, as well as the quality and type were very different from other sessions. As a result, we decided to analyze this session separately and this is presented in Section 4.4. The number of questions asked and total time spent in the Handover sessions are shown in Table 1.

Table 1: Number of questions and time spent

Run	Subject A	Subject B	Total	Handover Time (min)
1 (E-N)		5	5	9:55
2 (E-N)	1		1	3:00
3 (N-E)		7	7	7:44
5 (N-N)		5	5	10:00
6 (E-E)	4	8	12	12:45

4 (N-E)	10	24	34	27:56
---------	----	----	----	-------

* (Subject A - Subject B), E = Expert, N = Novice

4.2. Types of Questions

In order to perform the modification, the subjects needed to understand the VT survey application. Previous studies [12, 13] showed that in order to successfully comprehend software, information from different levels of abstraction is needed in order to build a mental model. To investigate what information subjects asked for, we categorized the knowledge requirement of the questions into three levels of

abstraction. The summary of the categories is shown in Table 2.

Domain Level (**D**) questions are concerned about a program's external behavior, such as those perceived by users of the application. We further categorized questions in this level into:

D1: Questions that ask about task's requirements, as provided in the task description document in order to clarify or to confirm them.

D2: These questions ask about what the application can do and how a user performs a specific task.

D3: These questions aim to confirm the scope of the task. The inquirers express concerns whether additional features are required to complete the assigned task.

Intermediate Level (**I**) questions ask about mechanisms that map between domain and program level behaviors. Examples of this kind of information includes: software architecture and high-level design information.

Program Level (**P**) includes questions about low-level design or information that is often grounded in the source code. These questions are divided into:

P1: Questions about location of data files, source code files and how to run the application.

P2: Question about meaning and behavior of methods, objects, or JSP files. These questions target the information in a lower level of abstraction than question in **I** category.

All other questions such as those asking about tool preferences are grouped in the Other (**O**) category.

Table 2: Summary of categories of questions

Level	Category
Domain	D1: Restatement of Change Request. D2: Application Usage D3: Task Scoping
Intermediate	I: Mechanics
Program	P1: Set Up P2: Code Level Mechanics
Other	O: Questions that were not asked about the application.

All questions asked in each category except those from the Run 4 are presented in Table 3 grouped by the run in which they were asked.

Table 3: All questions asked excluding run 4's

Questions	
D1: Restatement	
R1	<ul style="list-style-type: none"> ▪ So all they wanted was? ▪ What (field) you want to see is? ▪ (you need to display) Email address. Not the password?

R3	<ul style="list-style-type: none"> ▪ Is the user id basically any characters or numbers? ▪ Save (email) the same way (as user id and password)?
R6	▪ Just add. Right? Add this field.
D2: Usage	
R1	▪ They (users) can't edit?
R6	▪ So this one (UI) is add new user and this one (UI) is change?
D3: Scoping	
R3	<ul style="list-style-type: none"> ▪ Do I have to verify that it (email field) is blank? ▪ No other fields (to be added)? ▪ Do I have to require the person to answer in email twice?
R5	▪ I have to check for email and make sure it was entered into form?
R6	<ul style="list-style-type: none"> ▪ I'm not sure if we want to include this piece of information?* ▪ Should we allow the admin to change the user's email or not?* ▪ I'm not sure whether they require us to do this or not?* ▪ Whether the email information is require or not ... [if the blank data is allowed]?* ▪ [Do I have to validate incorrectly formed email address]?
I: Mechanics	
R3	▪ And surveyMetaData is like the database structure?
R6	▪ So they don't need... they don't use data base right?
P2: Code-Level Mechanics	
R1	▪ On the backend there is no password field?
R3	▪ What are these files?
R5	<ul style="list-style-type: none"> ▪ Which one talks to the user.dtd? ▪ So it is just those three things I'm going to worry about? ▪ Which files are going to need to be modified?
R6	<ul style="list-style-type: none"> ▪ Is this file going to control the elements in the xml file? ▪ I'm not sure whether we can use this new data structure to process the old data?* ▪ Leave what? There's going to be a pair in the xml files right? ▪ So what files are involved in these changes?
O: Other	
R2	▪ Whether you are familiar with the code?*
R5	▪ Should I just save it (currently opened files)?

* shows questions asked by Subject A

The majority of the questions were at the Domain level, although our requirements were quite simple. In addition, there were no "why" questions asked although design rationale is considered important information in comprehending software. This result is similar to the finding from Herbsleb et al. [6].

To completely understand a program, a programmer has to establish links between information in different levels of abstraction. Incomplete explanations and few questions suggested that a lot of information may be missing. To our surprise, there were few questions

asked at the Intermediate and Program level regardless of the completeness of the explanation received. Further examination of the implementations showed that subjects had difficulty finding out the information by themselves, especially for the novices. This suggests that subjects did not already possess the information that was not asked for. In addition, the coverage of question was very narrow (See Section 4.4 for comparison).

We also expected that Subject A would ask about his partner's experience in order to provide explanation in a suitable presentation level. However, there was only one question of this kind.

Table 4: Number of questions in each category and implementation score of each session

Run/ Level	1 EN	2 EN	3 NE	5 NN	6 EE	Total	4 NE
D1	3		2		1	6	
D2	1				1	2	1
D3			3	1	5	9	
I			1		1	2	13
P1						0	5
P2	1		1	3	4	9	5
O		1		1		2	10
Total	5	1	7	5	12	30	34
Impl.	35	12	35	50	44		55

The number of questions asked in each category and the implementation score from each session are presented in Table 4. In this study, subjects who asked about information in the I and P levels were better able to complete the task than those who did not. This is reasonable as our task had a time limit. Asking for this information allowed the subjects to exploit knowledge from his partner and to spend less time doing research. The fact that they were able to formulate the questions suggested that they would be able to understand the explanation and hence able to make use of the information.

Over the next two subsections, we discuss two interesting patterns of questions-asking that were manifested in our study.

4.3. Run 4: Asking in the Absence of Answers

More than half of questions in the entire study were asked in Run 4 by our most experienced subject, E4, who had 5-10 years of experiences in both web applications and databases. In addition, E4 was the only subject who completed the modification task. E4 received a very short and incomplete explanation from his partner, N4, who had less than one year of

experience in both areas. The complete list of questions asked in Run 4 is presented in Table 5.

A large number of questions might seem reasonable as E4 had sufficient knowledge to detect missing information and could ask questions to obtain the missing information. However, what made this session interesting is that E4 decided to continue “asking” N4, who was obviously a novice and could not answer even his simplest questions. The following excerpt was taken from the beginning of Run 4’s Handover session.

E4: Where is the data store?

N4: No idea. I don’t know where the data is stored. I can’t even find servlets.

E4: Hmm. Why couldn’t you find the servlets?

N4: I don’t know where it’s at.

E4: Why don’t you know?

It would not have been surprising if E4 disregarded N4’s explanation and ended the Handover session at this point. But E4 persisted. This question-asking pattern continued for nearly half an hour. When N4 could not answer the questions, E4 found the answers to his questions and explained them to N4. Since E4 clearly could not obtain answers from N4, what was the purpose of questions in this session?

As some information provided by N4 was incorrect, E4 used questions to judge the correctness of explanation provided by N4. In addition, questions were used to engage N4 in the collaborative conversation. For example, after examining a portion of Java code E4 posed a question:

E4: Doesn’t it gives you the impression that each files has a user associated wit it?

N4: Name of the user as a file?

E4: The user yeah, the name of the user is the name of the file.

N4: As you pointed out, it’s an array of

In addition, the type and quality of questions asked in this session were different from the other sessions. There was only one Domain level question and most of questions asked were in Intermediate level. In addition, the questions touched on more parts of the system in more detail.

Table 5: Questions asked in Run 4

Run	Questions
	D2: Usage
R4	▪ Where to actually go to that... the main page?
	I: Mechanics
R4	▪ Did you say I do or do not have to modify the Servlet? ▪ It is a Servlet. It’s not a javabean, not ... ?

	<ul style="list-style-type: none"> ▪ Where is the data stored? ▪ Why couldn’t you find the servlets? ▪ Why don’t you know (where the Servlet is)? ▪ Do you know if the data is stored in an xml file or a real relational database? ▪ Doesn’t it give you the impression that each file has a user associated with it? ▪ The name of the user as a file?* ▪ What give you the impression that there are Servlets involved and not just jsp files? ▪ Did you see something like that (how Servlet is used by Subject A in his past experience) here? ▪ So how would that (using only JSP to implement web application) work anyway?* ▪ How would what work? ▪ Do you know if it’s using the standalone or LDAP authentication method?
	P1: Set Up
R4	<ul style="list-style-type: none"> ▪ Do you know where that directory is? ▪ Do you know the name of one of the user on the system? ▪ Do you know where the user class is? ▪ Do you know where the source file is? ▪ Do you know how to build this stuff? Have you done that?
	P2: Code-Level Mechanics
R4	<ul style="list-style-type: none"> ▪ I don’t know what that (HttpUtils.getRequestURL(request)) is ?* ▪ How does the listAllUser.jsp file retrieve the email address or retrieve the user name for display? ▪ Where would you go? I mean the action... * ▪ It returns back to the page, which make sense. Right? ▪ If you get down to “setPassword”, what does it do?
	O: Other
R4	<ul style="list-style-type: none"> ▪ Have you seen the program?* ▪ You understand this part. I mean why you have to change it?* ▪ Can you do find for “test” (a user name)? ▪ Can you add another user in the system with a more unique name? ▪ Can you open the xml file? ▪ This (file that he was asked to open) one?* ▪ What do you want to see it (the file) with?* ▪ Could we look at the constructor?* ▪ How long will it take you to do that?* ▪ Localhost... Can we go back to the main ... main page?

* shows questions asked by Subject A

4.4. Run 1 and 2: Novices Don’t Ask Experts

In contrast to Run 4, very few questions were asked in Runs 1 and 2. These sessions had expert explainers and novice implementers, and very few questions were asked especially in Intermediate and Program level. However, the quality of explanations given in both sessions was different.

In Run 1, N1 received a very good explanation from E1. N1 asked very few questions, mostly in the D1 category. At first glance, this suggests that N1 received all the important information needed to complete the task down to the level of which files and methods were needed to be modified and how to modify them. There was no need to ask questions to obtain further information. However, this was not the case. The video from implementation task showed that N1 did not understand the explanation given and had difficulty utilizing the information given by E1. N1 disregarded the CRP from E1 and spent a lot of time trying to obtain the same information available in that document.

In Run 2, E2 gave a very short and incomplete explanation to N2. However, N2 did not ask *any* questions. From his implementation task result, we know N2 had trouble finding information on his own. In addition, E2's explanation contained some incorrect information; N2 ended up being misled and received the poorest implementation score.

It is obvious that both novices did not have the information required to understand the application. Why didn't they ask more questions to their partners who are experts and should be able to provide them with useful information? In the next section, we explore some possible explanations.

5. Discussion: Why developers don't ask

In this study, the Handover sessions were rather short and there were surprisingly few questions asked, especially by novices. We expected Subject B to ask for more information in order to benefit from Subject A's knowledge. In this section, we will speculate on the reasons why there were so few questions.

5.1. Experts

The result showed that experts were responsible for asking the bulk of questions in this study. Excluding Run 4, 20 out of the 30 questions were asked by experts. This makes an average of five questions asked per person which is still not many. A possible explanation is that experts are confident that they can find out required information on their own. Among the 20 questions asked by experts, 8 were asked to novices and 12 to experts. This shows a tendency to ask more questions to experts than to novices. This might be because experts have more common ground with other experts. Also, experts can easily identify whether their partners are novices or experts [14]. When paired with novices, it's possible that experts could detect their

lack of experience and discounted their ability to provide useful answers.

The types of questions asked by experts were also different from novices. Experts asked more questions at the Program level than novices. This might be because experts have better domain models which allow them to know what information they need at P level. P level information will be used to work out what information is needed at the I level. This was also substantiated by Run 4's collaborative problem solving and information seeking.

5.2 Novices

In this study, excluding Run 4, novices asked average of only two questions per person. A possible explanation is that novices lack domain knowledge and a suitable framework to understand the explanation, an inability to know what they need to know [4]. This explanation is supported by Ahmed and Wallace's finding that novices usually don't understand their knowledge needs [11]. It is also possible that novices may have questions but did not know how to ask them, because they have difficulty framing question due to the lack of common ground with the explainer. As presented by Ram [3], in order to ask a question, one almost needs to know what the answer is going to be. This is similar to forming a hypothesis about the answer. Also, in order to understand an answer, one needs to be able to anticipate the answer in order to incorporate it into existing knowledge.

5.3 The Social Act of Asking Questions

People don't always ask their questions. The simple explanation is that they are afraid of asking stupid questions or they are self-conscious. However, the reasons for this are more varied and more profound.

Flammer proposed that the process of asking is a decision process that negotiates between costs and benefit of asking the question [2]. Examples of cost include time and effort spent in asking and understanding the answer, and the shame of appearing ignorant. Other factors include importance of questions, likelihood of existing answers elsewhere, and likelihood of understanding an answer. Our laboratory study de-contextualizes the interaction between subjects. They had no prior experience with the application and were not familiar with each other, which makes background and credibility of the information source unclear. Our subjects may have felt hesitant to ask questions because benefit of asking was unknown.

Cultural factors also affect questions-asking behavior. Berlin [5] observed that among developers, semantic questions were preferred over questions about "simple" technical problems with the environment, tools, or programming syntax. In addition, the culture also encouraged novices to try to find out answer on their own before asking experts. Asking "trivial" questions might be considered bothersome or distracting to experts, whose time is valuable. Finally, face-to-face question-asking may not be valued as a means to transfer and manage knowledge because it doesn't leave a record. The use of Instant Messaging tools or email might be a preferable way to ask questions.

6. Future Work

We plan to address open issues and to further observe the question-asking behavior in future studies. Possible modifications to the study design include: i) controlling the quality of presentation and explanation given to Subject B using confederates; ii) allowing Subject B to ask questions during the implementation; iii) providing additional means for asking follow-up questions through Instant messaging or email; and iv) use the debriefing session to ask about the reasons that a subject asked or didn't ask questions, and what they think they should or should not have asked.

7. Conclusion

We have presented an initial study aimed at understanding question-asking behavior in knowledge exchange in software development. The study required novice and expert software engineers to collaborate on a change request for a web application. We found breakdowns in question-asking by both novices and experts. The results showed that this seemingly simple activity is often not performed well nor as frequently as necessary for successful knowledge collaboration. Novices may not realize their knowledge needs nor be able to frame questions due to lack of domain knowledge. Other factors that prevent developers from asking questions are lack of common ground, likelihood of finding an answer without asking, a disbelief in the credibility of information source, and the low cultural value of a some types of questions.

8. Acknowledgements

Our thanks to Oluwatosin Aiyelokun, Erin Morris, Justin Beltran, Matt McMahan, Teerawat Meevasin, John Situ, Derrick Tseng, Jonathan Zargarian for

providing invaluable assistance in preparing and conducting the experiment. Thanks also to Jeff Elliott and Yuzo Kanomata for helping us with initial pilot testing.

This material is based upon work supported by the National Science Foundation under Grant No. 0430026. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

9. References

- [1] B. Curtis, H. Krasner and N. Iscoe, "A field study of the software design process for large systems," *Communication of ACM.*, vol. 31, pp. 1268-1287, 1988.
- [2] A. Flammer, "Towards a theory of question asking," *Psychological Research*, vol. 43, pp. 407, 1981.
- [3] A. Ram, "A Theory of Questions and Question Asking," *The Journal of the Learning Sciences*, vol. 1, pp. 273, 1991.
- [4] N. Miyake, D.A. Norman, "To Ask a Question, One Must Know Enough to Know What Is Not Known." *Journal of Memory and Language*, vol. 18, pp. 357, 1979.
- [5] L.M. Berlin, "Beyond Program Understanding: A Look at Programming Expertise in Industry," in *Empirical Studies of Programmers: Fifth Workshop*, pp. 6-25, 1993.
- [6] J.D. Herbsleb, H. Klein, G.M. Olson, H. Brunner, J.S. Olson and J. Harding, "Object-oriented analysis and design in software project teams," *Human Computer Interaction*, vol. 10, pp. 249-292, 1995.
- [7] A.L. Jarczyk P. and I.F. Shipman, "Design Rationale for Software Engineering: A Survey," *Proceedings of the 25th Annual IEEE Computer Society Hawaii Conference on System Sciences*, pp. 577-586, 1992.
- [8] M.C. Hoepfl, "Choosing qualitative research: A primer for technology education researchers," *Journal of Technology Education*, vol. 9, pp. 47, 1997.
- [9] S.E. Sim, S. Ratanotayanon, O. Aiyelokun and E. Morris, "An Initial Study to Develop an Empirical Test for Software Engineering Expertise," *UCI-ISR-06-6*, 2006.
- [10] M.M. Sebrechts and M.L. Swartz, "Question asking as a tool for novice computer skill acquisition," in *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 293-299, 1991.
- [11] S. Ahmed and K.M. Wallace, "Understanding the Knowledge Needs of Novice Designers in the Aerospace Industry," *Des Stud*, vol. 25, pp. 155-173, 2004.
- [12] A. Von Mayrhauser, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, pp. 44, 1995.
- [13] N. Pennington, "Comprehension strategies in programming," pp. 100-113, 1987.
- [14] R.L. Campbell, N.R. Brown and L. DiBello, "The Programmer's Burden: Developing Expertise in Programming," in *The Psychology of Expertise: Cognitive Research and Empirical AI*, R.R. Hoffman, New York: Springer-Verlag, 1992, pp. 269-294.