

Peer Interaction Effectively, yet Infrequently, Enables Programmers to Discover New Tools

Emerson Murphy-Hill
Department of Computer Science
North Carolina State University
Raleigh, United States
emerson@csc.ncsu.edu

Gail C. Murphy
Department of Computer Science
University of British Columbia
Vancouver, Canada
murphy@cs.ubc.ca

ABSTRACT

Computer users rely on software tools to work effectively and efficiently, but it is difficult for users to be aware of all the tools that might be useful to them. While there are several potential technical solutions to this difficulty, we know little about social solutions, such as one user telling a peer about a tool. To explore these social solutions in one particular domain, we describe a series of interviews with 18 programmers in industry that explore how tool discovery takes place. These interviews provide a rich set of qualitative data that give us detailed insights into how programmers discover tools. One finding was that, while programmers believe that discovery from peers is effective, they actually discover tools from peers relatively infrequently. Another finding was that programmers can effectively discover tools from their peers both in a co-located and remote settings. We describe several implications of our findings, such as that discovery from peers can be enhanced by improving programmers' ability to communicate openly and concisely about tools.

ACM Classification Keywords

D.2.6 Software Engineering: Coding Tools and Techniques

General Terms

Human Factors

Author Keywords

discovery, learning, programmers, programming tools

INTRODUCTION

Software features or tools, such as the ability to correct grammar in Microsoft Word or to recover recently closed tabs in Mozilla Firefox, allow users to perform their tasks more efficiently and do things they were unable to do previously. However, users have difficulty becoming aware of tools that might be useful to them. For example, when Grossman and colleagues conducted a study of 10 users of a computer-aided drafting application, they found that a "typical prob-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CSCW 2011, March 19–23, 2011, Hangzhou, China.

Copyright 2011 ACM 978-1-4503-0556-3/11/03...\$10.00.

(02:02:21 PM)	FEZ:	Hold on.
(02:02:23 PM)	FEZ:	What did you just do?
(02:02:39 PM)	HAL:	Replace the first three arguments with a combined one.
(02:02:39 PM)	FEZ:	How'd you do that delete?
(02:02:45 PM)	HAL:	Oh. 'd%'
(02:02:59 PM)	FEZ:	But then you deleted the -> too.
(02:03:19 PM)	HAL:	Yeah, it scans forward for the next open thingy, then to the matching close thingy.

Figure 1: A snippet of an instant-messaging session where one user learns about a software feature from another user.

lem was that users were not aware of a specific tool or operation which was available for use" [6, p. 655]. As another example, Campbell and Miller have noted that awareness is a problem in integrated development environments that are used by programmers [2]. It is arguable that in any at least moderately sophisticated application, many users will remain unaware of the full range of tools available.

Several technical solutions to the problem of lack of awareness have been proposed. Some applications attempt to solve this problem with tip-of-the-day messages or role- or task-based customizations of the user interface. Researchers have proposed other technical solutions as well, such as recommender systems that suggest tools that you are not currently using [9, 10, 12].

While there has been much research into *technical solutions*, there has been relatively little research into *social solutions*, where a user learns about a tool from another user. To illustrate what we mean by social solutions to the awareness problem, let us give two examples drawn from the study we describe in this paper.

The first example illustrates how FEZ discovers a tool that is useful to him from another person. FEZ is a programmer who often works with a programmer named HAL. While using a remote screen-sharing session together, FEZ noticed that HAL did something to make some text move around in their shared vim editor. Figure 1 shows an exchange that fol-

lowed in an instant-messaging session. In this session, FEZ gained awareness of a tool that he later found very useful.

The second example illustrates how ELI has often failed to learn about useful tools from other people. ELI is an interactive media producer who uses Twitter (twitter.com), a social network where people share very brief messages via microblogging. ELI rarely discovers useful tools through Twitter because the people he interacts with tend to recommend the “latest and greatest” tools, but not necessarily the most useful ones.

What are the key differences in the social and technical contexts that allow FEZ to discover useful tools when programming with a peer, yet often disallow ELI to discover them through microblogs? In this paper, we explore these contexts by focusing on programmers, both because we are conversant with the tools that programmers may use and because the range of tools available to programmers is so wide. Although we cannot claim that our results will generalize to other types of software users, we view programmers as users of particularly advanced software, and those advances are likely to be shared with all software users in the future. Our long-term research goal is to encourage all kinds of software users to discover tools more successfully, more frequently.

Previous research by Cockburn and Williams [3] has suggested that tools are frequently discovered when programmers work together in an activity known as pair programming. We call this mode of discovery *peer interaction*, where programmers discover tools from their peers during normal development activities. Using Cockburn and Williams’ research as a starting point, we conducted interviews to determine how peer interaction works and how it relates to other modes of discovery, such as Twitter and exploring an application’s user interface. Based on our observations, we then discuss several implications for helping programmers discover new tools.

The three main contributions of this paper are:

- a characterization of peer interaction, a mode of discovery where programmers learn about the existence of new tools from peers;
- evidence that peer interaction may be the most effective way for programmers to learn new tools, yet it appears to occur infrequently; and
- implications for how the social process of peer interaction can be fostered so that it can occur more frequently in the future.

TERMINOLOGY

We are interested in how programmers discover tools, but what are tools, and what is discovery? By tools, we mean any software that helps a programmer accomplish a task, including standalone programs like development environments and features or commands in those environments, like source code formatters. Although there are many types of tools (such as those that enhance productivity or enable collaboration), we do not distinguish between types. Note that, while

we will mention several tools in this paper in examples of discovery, we will not explain the specifics of those tools.

When we say that a programmer discovers something, we mean that she becomes aware of that feature. Findlater and McGrenere distinguish findability from awareness; “findability measures the speed with which users can find known functions, and awareness measures the degree to which users are conscious of the full set of available functions” [5]. Similarly, we define a discovery to mean the event in which a user finds out about a tool that she did not know about.

Discovery is also closely related to, but distinct from, learning, in that discovery can be thought of as the first stage of some kinds of learning. We defer a thorough discussion of the relationship to the Related Work section.

A STUDY OF PEER INTERACTION

To study whether and how programmers discover new tools from peers, we conducted a series of interviews with programmers in industry. In the interviews, we discuss two forms of peer interaction: **peer recommendation** and **peer observation**. We hypothesized that peer recommendation and peer observation may occur during pair programming, when two programmers work on the same programming task at the same computer. In such situations, the “driver” is at the keyboard, and the “navigator” is sitting beside the driver, observing and making suggestions [3]. We hypothesized that a programmer may discover a new tool in either role:

- The driver may discover a new tool when the navigator says something like, “you know, you could really use tool X instead.” We call this peer recommendation.
- The navigator may discover a new tool when observing the driver using the tool, saying something like, “how did you do that?” We call this peer observation.

Methodology

To better understand peer interaction, we wanted to collect a substantial number of descriptions of peer interaction. Ideally, we might have observed programmers learning tools during their normal software development activities. Instead, we opted to conduct retrospective interviews for several reasons. First, we suspected that peer interaction occurs so infrequently that direct observation is impractical; this suspicion was confirmed by subjects, as we will explain. Second, interviews allow us to speak with a variety of programmers at different companies and with varying experience. Third, interviews allow participants to reflect on motivations and long-term effects of peer interaction, not just the events that are visible from an observing researcher’s perspective.

We conducted one-on-one, semi-structured telephone or instant-messaging interviews lasting about an hour each.¹ The interview began with questions to ascertain the subject’s programming experience. Next, we referred the subject to a document that listed several pictures of different kinds of

¹The interview script can be found at http://people.engr.ncsu.edu/ermurph3/experiments/adoption_interview_script.pdf.

Table 1: Seven discovery modes, as read to subjects.

Peer Observation where you observe someone else use a tool while programming that you didn't know about
Peer Recommendation where someone observes you programming and suggests the new tool
Tool Encounter where you just happen to find the tool by exploring the user interface of your development environment
Tutorial where you are reading or watching a tutorial that mentions a new tool
Written Description where you notice that a tool is mentioned on a website or publication
Twitter or RSS Feed where you learn about tool from someone or some site that you are following
Discussion Thread where you learn about a new tool after reading it on list of comments, forum, or email discussion

programming tools, which we chose from the Eclipse and Visual Studio development environments, as well as the extensible editors vim and emacs. Although retrospective interviews are common, the results can be influenced by people's memory of discovery and adoption. Therefore, we used the tool list to help stimulate the subject's memories of tools that she might have discovered, using them as recall cues for known-item memory retrieval [1]. We asked the subject to pick three tools from the list (or tools similar to tools on the list) and to describe how she discovered and learned about them. The purpose was to attempt to ascertain the most frequently occurring modes of discovery, on the assumption that the most frequently mentioned modes for a certain set of tools are the most frequent modes.

We then asked the subject to choose, in her experience, the two most effective modes for discovering new tools. We clarified effectiveness as how effective each mode is on "your likeliness to use a tool again." We gave the subject a list of seven different discovery modes, as shown in Table 1. We also encouraged the subject to think of other modes. We then asked the subject which, in her experience, are the two least effective modes.

At this point, we revealed to subjects that we were specifically interested in peer observation and peer recommendation, and asked for the subject to describe her experiences learning new tools in those modes. We asked the subject to relate experiences when she was the learner or teacher during peer observation and peer recommendation. For each experience, we asked a semi-structured set of questions to elicit detailed responses, including the context in which the learning happened, the nature of the relationship with the peer, and what was said or done to facilitate learning.

We then asked the subject directed questions about her experience with peer interaction, including how often she learns or teaches, and how it has changed over her programming ca-

reer. Finally, we asked the subject some opinion questions, then thanked the subject and concluded the interview.

To analyze the data that we collected, the first author recorded the interviews, then transcribed and summarized them. From the summaries, he coded the discovery instances by mode, and also by any other categories that appeared, such as by the location in which the discovery took place. He then re-read the summaries and codings several times, iteratively refining the codes during reviewing. He also categorized the contents of the summaries by question and identified patterns in responses and relationships between responses.

Subjects

We recruited subjects from two main sources. First, we emailed invitations to 62 participants who volunteered to be contacted at Open Source Bridge 2009, a conference for "developers working with open source technologies and for people interested in learning the open source way" (<http://opensourcebridge.org>). Second, we sent emails to personal contacts at seven large companies, asking them to pass on our invitation to potentially interested colleagues. Two people volunteered through these personal contacts and the rest through Open Source Bridge.

Overall, 18 people responded and completed the interview, comparable to the size of similar studies such as those by Twidale (5 subjects) [18] and Rieman (14 subjects) [16]. Subjects had between 3 and 32 years of professional programming experience (median=9); not all were employed as programmers or software developers, although programming played a role in their job, or most recently held job. Subjects were between the ages of 21 and 51 (median=30.5). Subjects reported using a total of 18 different editors or development environments within the last year; the common ones (ordered from most to least frequently mentioned) being vi/vim, emacs, Visual Studio, TextMate, Eclipse, and Netbeans. Subjects reported using a total of 24 different languages within the last year; the common ones being python, javascript, PHP, Ruby, Java, C, and perl.

Subjects reported a variety of working experience, which we suspected had some effect on peer interaction. We will refer to subjects in our study by pseudonyms, listed in the left-most column of Table 2. In the next column to the right, we list how many years of experience each subject reported. In the next column, we list whether or not each subject works on a team with other programmers in their current or most recent job. The next two columns show which subjects regularly read technical blogs — websites where people post regular writings on technical topics — and which subjects are users of Twitter. We were interested in blogs and Twitter because we suspected that they played a role in tool discovery. We explain the right two columns of Table 2 in the next section.

Results

Overall, subjects reported 41 different instances of peer interaction, of which 27 were peer observation and 14 were peer recommendation. In this section, we describe based on

Table 2: Subjects’ pseudonyms are displayed in the left-most column; pseudonyms assigned alphabetically based on subjects’ experience level (in years). Next, potentially programming-relevant social activities are listed. Finally, likeliness to learn or teach tools via peer interaction is listed.

● means that a subject learns via peer interaction between once every week and twice per month; a ◐ means that a subject learns every one or two months; and a ○ means that the programmer learns between once every three months and once per year. Programmers estimated that they taught less often (−), about equally often (≈), or more often (+) than they learned via peer interaction.

experience	team	blogs	Twitter	learn	teach
ART	3	✓	✓	◐	+
BEN	4		✓	●	−
CAL	5		✓	○	+
DEL	6	✓	✓	●	−
DON	6		✓	●	+
ELI	7		✓	◐	+
ENU	7	✓	✓	○	≈
FEZ	8	✓		●	+
GIL	9	✓	✓	◐	+
GUS	9	✓	✓	●	+
HAL	10	✓	✓	◐	+
HAO	10	✓		●	−
KAI	13	✓	✓	○	+
KEN	13	✓	✓	◐	≈
ROB	19	✓	✓	○	−
VAL	25	✓		○	≈
YIT	31		✓	○	+
ZAC	32		✓	○	−

the data how peer interaction relates to other types of discovery and how it works in the field. At the end of each subsection, we briefly summarize our findings.

The Steps of Peer Observation

The process of peer observation occurs in several steps: two programmers interact in some situation, the learner observes the teacher using a tool that she does not know, the learner interrupts the teacher, the learner asks a question about the tool, and then the teacher responds to the learner. In what follows, we describe what programmers told us happens during each of these stages.

Observation Situation. Peer observation occurred with subjects in four kinds of situations (Table 3): traditional pair programming, remote pair programming, happenstance interaction, and change notification.

Tools Observed. Subjects described teaching or learning a variety of different tools, including tools for debugging (such as Firebug and Web Developer), tools to help change code (such as sed/awk and refactoring), operating system tools (such as quicksilver), tools for collaboration (such as screen sharing), and shortcuts (such as vim macros).

Interruption Timing. Subjects reported that the learner almost always interrupted work to question the teacher, typ-

ically immediately after the tool is used. FEZ also pointed out an instance where the learner asked the teacher even before she was finished using the tool and ZAC described an instance after repeated uses of the tool in the same programming session. In contrast, BEN noted that, over the course of learning vim from peers, for the most part he did not ask questions while learning new tools within vim, presumably because the commands that his teacher was executing were largely visible and self-explanatory.

Interruption Wording. Typically the interruption is a comment along the lines of “what is that?” (ELI, HAO, ZAC), “how did you do that?” (FEZ, GUS, HAO, ROB), or an exclamation of amazement or surprise (BEN). Such reactions to initial tool use were not always polite, such as in the case of KEN, who recalled a peer remark in response to his tool use, “what the hell is all this crap?”

Response to Interruption. The teachers’ response to the interruption from the learner varied, though subjects reported that typically the teacher gave an explanation of what the tool did and a short demonstration (less than a couple of minutes). Several subjects also reported that they followed up with this discovery episode by trying the tool out when the teacher and learner separated. Other subjects reported being given URLs by the teacher for later reference.

In sum, subjects reported that peer observation occurs in pair programming situations, consistent with others researchers’ observations [3], but also in other situations where two programmers are not working on the same task. Rather than passive discovery, subjects reported that the observer interrupted the other programmer verbally (or by instant-messaging, if the interaction was remote), which was followed by an immediate discussion and demonstration, or post-discovery exploration and reading.

The Steps of Peer Recommendation

The process of peer recommendation has steps similar to peer observation: programmers interact in some situation, the teacher observes the learner do something for which the teacher knows an tool alternative exists, the teacher interrupts the learner, and then the teacher delivers the recommendation.

Recommendation Situation. Subjects reported that peer recommendation happened in five kinds of situations (Table 3): traditional pair programming, happenstance interaction, help giving, remote help giving, and email.

Interruption Timing and Wording. As with peer observation, most subjects reported that the person making the recommendation made it immediately. The recommendation was sometimes direct, as in “you should use X” (BEN, CAL, ENU, KEN), and sometimes more subtle, as in “you might try X” (HAL, GUS). However, not all subjects reported this immediate interruption. For instance, HAL described watching a colleague repeatedly open classes inefficiently, and recommended the Open Type dialog after some time:

Table 3: Situations in which tool discovery occurred via peer observation and peer recommendation, with examples.

Situation	Description	Example
Traditional Pair Programming	Two programmers work at the same computer and collaborate to complete the same task.	<i>Peer Observation.</i> While programming with a coworker, DEL noticed Firebug when the coworker used it to solve a problem.
		<i>Peer Recommendation.</i> KEN was recommended the Open Type dialog in Eclipse while pair programming.
Happenstance Interaction	One programmer observes another during a chance encounter.	<i>Peer Observation.</i> KAI had the Labview development environment on his screen, which caught a coworker’s attention when the coworker walked by.
		<i>Peer Recommendation.</i> A peer walked by to say that he updated code; the peer noticed GUS using repeated update and commit commands, and the peer recommended the synchronize command instead.
Help Giving	A programmer helps a collocated programmer with a task.	<i>Peer Recommendation.</i> HAL recommended the Open Type dialog while helping a peer with a coding problem.
Remote Help Giving	A programmer helps a remote programmer with a task.	<i>Peer Recommendation.</i> While helping a colleague over instant-messaging with a problem, DON recommended a specific debugger, which the colleague then used to fix the problem.
Remote Pair Programming	Two programmers work at different computers and collaborate to complete a task.	<i>Peer Observation.</i> While using a remote vim editor with a peer, FEZ saw text move; FEZ asked what happened via instant-messaging, and the peer indicated he was using a feature that FEZ did not know (Figure 1).
Change Notification	A programmer commits code to version control, and an email is sent to the team about the commit.	<i>Peer Observation.</i> After receiving a change notification, a coworker asked ENU why he made so many changes. ENU responded that he had made the changes based on recommendations from Findbugs, showed the peer a Findbugs report, and the peer ended up downloading and using Findbugs.
Email	One programmer observes another’s actions via email.	<i>Peer Recommendation.</i> FEZ sent a progress report to his supervisor; the supervisor recommended reporting progress on a company wiki. FEZ discovered that using a wiki helps him efficiently keep track of his own tasks and inform teammates of those tasks.

I’ll generally leave them to their way of working for a while before observing a pattern that I think I can help with. . . they may feel comfortable with what they’re doing, and comfort is important. . . I try to introduce things slowly, especially when I’m not sure that the person I’m working with sees it as a problem or thinks that they need help. If it doesn’t look like they’re suffering too much, it may be better to leave them alone.

Tools Recommended. Subjects mentioned a variety of tools that they had learned about via peer recommendation, the common ones being the Open Type dialog in Eclipse (ENU, HAL, KEN) and Firebug (ART, HAO). Subjects also described discovering Postgres, FTP, emacs tags, a debugger, and a tool for collaboration.

Recommendation Delivery. As with peer observation, most subjects reported that the recommender responded by demonstrating the tool in a task-relevant manner. For example, when ART recommended Firebug, he demonstrated how it was used on the same webpage with which the learner was having trouble. The learner also sometimes followed up the recommendation by visiting websites or tutorials, and trying out the tool on their own.

In sum, subjects reported that peer recommendation happened in similar circumstances to peer observation, with sim-

ilar follow-up. However, in contrast to peer observation, where the interruption was often made with little trepidation, during peer recommendation subjects reported sometimes exercising more sensitivity to the learner. These results may suggest that programmers are more comfortable professing ignorance than expertise.

Frequency of Peer Interaction

We estimated how often peer interaction happens in two different ways. The first way was to ask programmers to tell us about situations in which they learned about a new tool. We then categorized each situation according to Table 1 and compared how often peer interaction was mentioned versus other discovery modes. This provided an estimate of relative frequency. The second way was to ask programmers how many times per year, month, or day they learned about a new tool. This provided an estimate of absolute frequency. We also asked programmers to estimate how their frequency of learning has changed over time.

Peer interaction did not appear to occur particularly frequently, compared to how often other discovery modes were mentioned. In Table 2, a name in a box represents one subject’s description of an instance of discovery. The number of boxes in each mode is the total number of instances of discovery that subjects mentioned. For example, subjects mentioned a

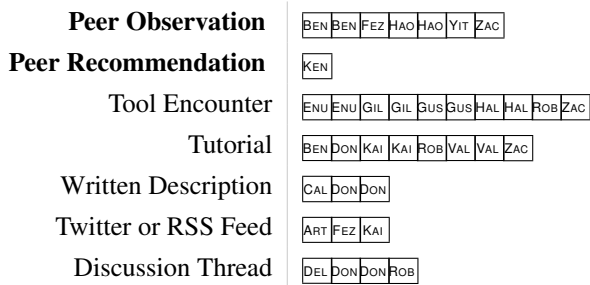


Figure 2: Histogram of the most frequent discovery modes.

total of three instances of written description: one from CAL and two from DON. Peer observation was mentioned seven times by five people; peer recommendation was mentioned only once.

Likewise, subjects reported learning and teaching via peer interaction fairly infrequently. The right two columns of Table 2 indicate how often subjects reported learning or teaching a tool via peer interaction.

In sum, compared to other discovery modes, peer observation and, especially, peer recommendation, were less frequently reported modes of discovery, compared to the most frequently mentioned mode. This finding is consistent with Rieman’s field study of learning and discovery, a study which provided evidence that tool encounter is the most frequent way of discovering tools in a variety of applications [16].

Effectiveness of Discovery Modes

We asked participants to rate how effective each mode is in terms of their likeliness to use a tool again in the future. Specifically, we asked subjects to name their two most effective modes, though we did not force subjects to choose exactly two. Figure 3 displays the results.

Peer observation and peer recommendation were rated as the most effective modes. These ratings are notable because the question was asked *before* we revealed to subjects that we were particularly interested in these two modes.

Effectiveness of Peer Interaction. Subjects reported that peer observation and peer recommendation were effective for several reasons:

- the learner has respect and trust in the teacher, so if the teacher had a good experience with the tool then the learner should take it seriously (BEN, CAL, DEL, GUS);
- the learner can reflect on the teacher’s use and apply it to her own programming (DON, HAL, KEN, YIT);
- programmers enjoy demonstrating their skills (ELI, HAO, ZAC); and
- the learner and the teacher share a common background so the tool is more likely to be relevant (ELI, HAL).

Also, subjects found peer observation effective because:

- the learner can see the value of a tool while it is in use

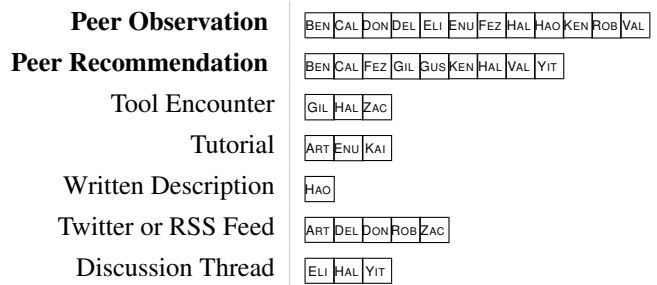


Figure 3: Histogram of the most effective discovery modes.

on a real problem (ENU, KEN, ROB, YIT);

- the teacher imparts a minimal amount of tool information, allowing the learner to feel like she discovered it herself and look up more material later (DEL, ELI); and
- the learner can associate the tool with its context of use, which makes it memorable (FEZ).

Due to space constraints, we do not report subjects’ responses as to why other discovery modes were effective or ineffective. However, we briefly summarize responses regarding Twitter/RSS, because it is similar to peer recommendation.

Subjects reported that Twitter/RSS are effective mechanisms because they trust or value the opinion of the people that they follow (ART, ROB) and they can gather the opinions of many people all at once (DEL, DON). However, some subjects reported finding Twitter/RSS ineffective, because the density of recommendations is too low (ART, ELI, HAL, HAO), people tend to recommend the most popular tools but not the most useful ones (ELI), the sources have low credibility (CAL), the sources do not have a similar background (HAL), some messages feel like advertising (CAL), and most messages are not relevant to programming (KEN).

In sum, subjects rated peer observation and peer recommendation as the most effective modes for discovering new tools. However, these modes also occur less frequently than other discovery modes (see Figure 2). This confirms McGrenere’s conjecture, who pointed out that exploratory learning is the most frequent kind of learning, but “it may not necessarily be the most efficient or effective method of learning how to use a system” [13, p. 15]. Indeed, these results confirm that it is not the most effective. Another important finding is that the determinant that subjects most cited as important to the effectiveness of discovery, whichever the mode, was trust.

Barriers to Peer Interaction

Subjects rated peer interaction as effective, but they also listed situations when it had not been effective. First, physical isolation makes peer interaction difficult because it is difficult to observe other programmers remotely, although other programmers reported effective remote observation. Second, when coworkers are working in entirely different programming environments, such as one using vim and another using Eclipse, then there are fewer tools that they can share. Third, once programmers have worked together for a certain

amount of time, they get acclimated to each others' toolsets, so the possibility of discovery is reduced. Fourth, company policies can inhibit social learning; subjects reported companies dictating which tools to use, even when they were not the best tool for the job. Fifth, when a project is under time pressures, such as a release deadline, programmers may not be willing to set aside time to discuss a tool during a development task. Sixth, programmers' themselves are sometimes unwilling to share tool knowledge.

We were especially interested in this last barrier to peer interaction; when are people unwilling to teach or learn? It is worth mentioning that, for the most part, the programmers that we interviewed appeared to be enthusiastic about learning and teaching tools to peers. However, subjects mentioned several cases where a programmer was unwilling to share or receive tool knowledge. First, ELI and HAL mentioned that people are sometimes unwilling to learn about a new tool because they are not sufficiently mature to appreciate the tool's usefulness. Second, ROB said that some programmers simply do not have an interest in learning new tools. Third, KAI and YIT mentioned that programmers sometimes feel that they do not need to discover a new tool because existing tools will do the job. Playing the role of such a programmer, YIT said:

“Why should I bother? I've got ido-mode, I've got ack, I've got this, that, and the other. . . the feeling is that, so far, I've made it without that [new] tool.” Developer inertia, I guess you could call it.

Fourth, FEZ mentioned that, in any given programming session, the programmers involved need to feel that they have made progress to feel positive, and when they end up spending all of their time learning about new tools, they have the feeling that it was not time well spent. Finally, DON described not learning new tools while programming because he was uncomfortable billing clients for learning about tools.

In sum, subjects reported the barriers to effective peer interaction are isolation, toolset differences, toolset acclimation, company policy, time pressures, peer maturity, lack of interest, “developer inertia,” the necessity of sensing progress, and client pressures. It is notable that these barriers occur because of a wide variety of internal and external sources: the client, the company, the management, the programmer, the development environment, and the tool.

Flow of Peer Interaction

Although we did not plan to ask subjects explicitly, we became interested in whether peer interaction occurs between peers or between a supervisor and a subordinate. If incidental tool learning is largely a kind of apprenticeship learning [8], then we should expect tool knowledge to flow largely from senior to junior programmers. While ART, FEZ, and GIL each described an instance of recommendations coming from supervisors, our results suggest that this is not always the case.

First, the instances of peer interaction were more often be-

tween peers than between programmers at different experience levels. As HAL explained, “differences [in skill sets] make the collaboration interesting, but the similarities make the collaboration easier.”

Second, two subjects took the opposite view, that during peer interaction, it is more often the junior programmers who are the teachers. FEZ and KEN explained this position; junior members have more free time to explore new tools, making them more likely to bring new tools into the organization. KEN said:

The junior members tend to be more voracious in their desire to learn new APIs and tools, and stay plugged in to what's going on with languages and stuff. My time is spent digging in to more bugs and more things that I'm responsible for delivering, I have less time to do independent research. . . No one is upset when a junior member says they have a better way to do things.

FEZ confirmed this:

There's a fair amount of bias towards me teaching [other engineers] something. . . I'm a student, an intern; I'm in the process of learning as much as I can from as many tools as I can. . . several developers I know, especially those that have 10, 20, 30, 40 years of experience, tend to say that they know the tools that they use, and they do not necessarily have the time, or more commonly, the patience to sit down and fiddle with a new tool.

These quotes provide anecdotal evidence that learning about tools can flow from the bottom upwards.

In sum, although tool knowledge appeared to flow primarily between peers, it also flows from supervisors to their subordinates *and* from subordinates to supervisors. This finding may be a result of the relatively flat organization of many software teams, where programmers feel comfortable asking about and recommending tools to other programmers, regardless of seniority.

A Remote Pair Programming Vignette

During the study, we learned that FEZ and HAL sometimes pair programmed together using a remote vim session. This offered a unique look into how peer interaction happens, from the perspective of both peers. Moreover, the two participants had saved full instant-messaging histories of their remote pair sessions, and were willing to share a few snippets of those histories with us.

Figure 1 displays one such occurrence. FEZ reported such occurrences were fairly common, where he would see something happen on the screen, ask about it, and HAL would reply. Interestingly, both peers gave examples of learning from one another, confirming the bidirectionality of peer interaction. One curious aspect of Figure 1 is that in order to learn the tool, FEZ needed to understand both the cause (pressing d%) and the effect (replacing the first three arguments with a combined one). We discuss the significance of

understanding causes and effects in the next section.

Threats to Validity

While this study provided a unique look into how programmers discover new tools, there are several threats to the validity of our study design.

Some subjects noted that it was difficult to remember instances of peer interaction. This difficulty of recall may have affected the accuracy of the results, especially when we asked subjects to estimate how often they learn via peer interaction. We tried to address this threat by focusing on specific instances of learning rather than generalizations, and when we did ask subjects more general questions, such as to estimate discovery modes' effectiveness, we preceded those questions by asking subjects to focus on specific instances.

To make conducting the study easier for the interviewer, we introduced the different discovery modes in a fixed order for every subject, as shown in Figure 1. This order may have biased subjects' effectiveness responses.

The study may suffer from ascertainment bias, because the programmers are not representative of all programmers, for two main reasons. First, although we did not specifically ask subjects about their cultural or geographic background, we suspect that subjects are largely Americans living in the western United States. Second, because each study participant volunteered to spend an hour talking to a researcher about discovery and learning, it may be that these programmers are more positive about social learning than the average programmer. Future studies should be conducted over a wider variety of programmers, both socially and culturally.

IMPLICATIONS

We have explored how peer interaction works, as well as shown that programmers have found it effective yet relatively infrequent. Our results have several implications for how to make it easier for programmers to discover new tools from their peers. Moreover, we suspect that these implications are applicable beyond software development environments to a variety of applications. We discuss two implications in this section: how tools and development environments can make it easier for programmers to discover tools from peers and how programming teams can encourage peer interaction.

Improving Tool Discoverability

The results of this study suggest at least two ways toolsmiths can make programming tools and environments more discoverable, so that when programmers interact, peer interaction is successful.

Noticeable Causes. If the manner in which a tool is used can be easily observed, then an observing programmer is more likely to both recognize that a tool was used and, implicitly, know how the tool is used. Hotkeys, used in many development environments so that programmers can quickly invoke commands, are a negative example because the keys that are pressed are typically not visible on the screen. One solution

is to show the keys that are pressed on the screen for a few seconds.

Noticeable Effects. In addition to making the causes of a tool invocation obvious, peer interaction may be facilitated if the effects of a tool are clear. Eclipse's Organize Imports command is an example of a tool that may not have noticeable effects; the tool automatically adds and removes import statements from Java files, but if those statements are not visible on screen, then an observer may not notice the effect of running the command.

Peer Interaction without Collocation

As software is developed on a global scale, teams become more and more distributed, reducing the discovery of tools during traditional pair programming. The results of our study also suggest that peer interaction can be facilitated when traditional pair programming is not possible.

Remote Pair Programming. Several subjects reported learning new tools via peer interaction with a peer by working at separate, geographically distributed workstations using a screen-sharing program. However, additional constraints have to be satisfied in order for such peer interaction to take place. First, the pair needs some channel by which to ask "what just happened?", such as using instant messaging or telephony. Second, visible causes and effects are especially important during remote pair programming, because implicit cues about how a tool is used, such as where a programmer's fingers are on the keyboard or where a programmer is looking, are absent. Third, programmers need a convenient, concise way to communicate about their tools. This third constraint is sometimes difficult to achieve. For example, if the programmers choose to collaborate using Eclipse and a tool requires several complicated steps to use, the teacher may be forced to say "first you click here, then here, then here, then type in this," and so on. More elegantly, if the programmers choose to collaborate using an environment with purely textual commands, like vim, the steps can be easily represented as a series of brief commands.

Learning from the Strengths of Peer Interaction. In the future, we might expect that collocated peer interaction will decrease as teams become more distributed, while at the same time, Twitter and internet tutorials (screencasts) may be increasingly common. Unfortunately, the results of our study suggest that Twitter and screencasts are not as consistently effective as peer interaction. We view this as an opportunity: what can we learn from peer interaction to make other discovery modes more effective?

Twitter bears some similarity to peer recommendation in that both are types of social discovery. One reason that subjects cited for Twitter being ineffective is lack of trust in the sources and lack of relevance. From the interviews, it appears what programmers meant by trust was that the recommender (human or otherwise) must have had some prior interaction with the developer, so that the developer can estimate the recommender's knowledge and skills. Our study suggests that trust and relevance might improve if the Twit-

ter messages originated from a trusted peer, someone that a programmer works with or has worked with in the past. Rather than burdening the trusted peer with having to report whenever she discovers a new tool, we imagine a system that automatically notices when she uses a novel tool and generates Twitter messages on her behalf.

Several subjects reported watching screencasts that were professionally produced (e.g., peepcode.com). Although watching screencasts is similar to peer observation, subjects reported that the tools used in screencasts may not be very relevant, presumably because the people who made the professional screencasts did not have working styles that aligned with individual subjects' working styles. Because peers are more likely to have similar working styles, a screencast produced by a peer is potentially more relevant. However, no subject reported watching a screencast produced by a peer. We suspect that the reason that programmers do not make screencasts for their peers is that the costs (recording, editing, and distributing) are too high compared to the benefits (the possibility of a peer discovering a tool). KAI hinted at this; "I wish people did make more screencasts; they're a pain in the ass to make." Better tool support for creation, editing, and distribution of screencasts may make it more likely that programmers will produce screencasts for their peers, improving screencasts' effectiveness.

RELATED WORK

The study presented in this paper is the first study of which we are aware of how programmers discover new tools, but several different research areas are closely related to peer interaction.

Diffusion of Innovations

Diffusion of Innovations is a theory that attempts to explain "the process by which an innovation is communicated through certain channels over time among the members of a social system" [17]. Typical studies of diffusion of innovations include research about internet use, hybrid corn in the US, and water sanitation in developing countries. The study presented in this paper can be considered a diffusion of innovation study that investigates tool discovery by programmers.

While we are not aware of any studies about diffusion of innovations involving programmers, several studies have investigated diffusion of innovations in the more general case of software engineering. For example, Fichman and Kemerer describe how relational databases, programming languages, and Computer-Aided Software/Systems Engineering (CASE) tools are acquired and deployed in organizations [4]. Similarly, Iivari described a study that suggests that the reason that companies do not use CASE tools is because of a lack of management support, a lack of perceived advantage, and a lack of freedom of choice [7].

Such research addresses critical issues, but it also tends to focus on tools that require a significant investment of time or money, and thus warrant careful organizational consideration of whether or not to adopt. In contrast, our research seeks to investigate a broad spectrum of tools, all the way

down to simple features such as code formatters, which likely require significantly less consideration from individual programmers than higher-level tools. Thus, while existing research has helped to determine how and why software environments have been adopted by organizations, our research also helps explain how and why programmers discover features within those environments.

Learning in Context

Several types of discovery that are similar to peer interaction have been documented in the literature.

One is Lave and Wegner's situated learning [8], where the learning occurs in the same place that the learning is used, a more general form of peer interaction. For instance, apprenticeships are a kind of situated learning. Whereas Lave and Wegner have focused largely on a fixed teacher-learner relationship, our research on peer interaction is on learning in peer-peer relations. Despite a focus on teacher-learner relationships, Lave and Wegner imply that there is significant potential in peer-peer learning: "There is anecdotal evidence... that where circulation of knowledge among peers and near-peers is possible, it spreads exceedingly rapidly and effectively" [8, p. 93]. Our study confirms this implication.

Another type of learning is Marsick and Watkin's informal and incidental learning, where learning happens as a by-product of other activities [11]. Marsick and Watkins note that with this type, "control of learning rests primarily in the hands of the learner" [11, p. 25]. In contrast, in peer interaction, learning is controlled by two people. By studying peer interaction with programmers, we extend research on informal and incidental learning into the domain of programming.

Yet another type is over-the-shoulder learning, where colleagues help each other informally to use a computer application [18]; over-the-shoulder learning is closely related to peer interaction in that they both occur among peers and both in a technology setting. The difference is that work on over-the-shoulder learning has primarily focused on situations where the learner explicitly asks for help, not in situations when the learner discovers a new tool by observation. In peer interaction, the learner does not initially know that she might find a tool useful.

As part of future work, we plan to compare peer interaction to other types of learning, such as proximal development, where [19] and constructionism [15].

Learning During Programming

As mentioned in the Introduction, Cockburn and Williams have described the learning of tools from peers during pair programming [3, p. 6–7]:

Knowledge is constantly being passed between partners, from tool usage tips (even the mouse), to programming language rules, design and programming idioms, and overall design skill. Learning happens in a very tight apprenticeship mode.

The partners take turns being the teacher and the taught,

from moment to moment.

Similar statements, describing peer interaction (as well as other kinds of learning), which suggests that knowledge about tools is passed between programmers, is oft repeated in the literature, but little evidence previously existed to support it. Both Cockburn and Williams [3] and Müller and Tichy [14] provide evidence that student programmers learn a variety of technologies during pair programming, but as Müller and Tichy question, “are these conclusions generalizable to professional software developers?”

Indeed, these studies prompt many questions about peer interaction. Does this kind of learning happen in the workplace, as well as in the university? Does it only happen during formal pair programming sessions, or in other situations as well? What kinds of tools do programmers learn in this way? How effective is learning in this way versus other kinds of learning? What makes this kind of learning effective or ineffective? How often does it happen? In this paper, we have extended Cockburn and Williams’ and Müller and Tichy’s findings by providing a more detailed analysis of the conditions, process, and results of this kind of learning for professionals with a variety of programming experience.

CONCLUSION

A wide variety of tools have been built to help programmers, but individuals must necessarily discover those tools before they can be used. In this paper, we have described a discovery mode called peer interaction, which encompasses both peer observation and peer recommendation, in which programmers learn about the existence of new tools. Based on information that we have collected in a series of interviews, peer interaction is the most effective mode for programmers to discover new tools. Unfortunately, it currently does not occur as frequently as in other discovery modes. The results suggest new ways to make existing tools and environments more discoverable and distributed collaboration more effective. Likewise, our results may be used to help other types of software users effectively discover new tools.

Acknowledgments

Thanks to the interview participants. Thanks also to Christian Bird, Andrew Black, Kellogg Booth, Giuseppe Carenini, Christina Conati, Nando de Freitas, William Griswold, Ciarán Llachlan Leavitt, Karon MacLean, Joanna McGrenere, Nancy Perry, Kenneth Reeder, Martin Robillard, Claudia Rocha, Martin Schulz, Jonathan Sillito, Laurie Williams, Phil Winne, Petcharat Viriyakattiyaporn, and the members of the Software Practices Lab and Interaction Design Reading Group at UBC for their help, during various stages of this research. This work was supported by the IBM Ottawa Center for Advanced Studies and NSERC.

REFERENCES

1. B. Allen. Recall cues in known-item retrieval. *J. of the American Soc. for Inf. Science*, 40(4):246–252, 1989.
2. D. Campbell and M. Miller. Designing refactoring tools for developers. In *WRT '08*, pages 1–2, 2008.
3. A. Cockburn and L. Williams. The costs and benefits of pair programming. In *XP '00*, pages 223–247, 2000.
4. R. G. Fichman and C. F. Kemerer. The illusory diffusion of innovation: An examination of assimilation gaps. *Inf. Systems Research*, 10(3):255–275, 1999.
5. L. Findlater and J. McGrenere. Evaluating reduced-functionality interfaces according to feature findability and awareness. In *INTERACT '07*, pages 592–605, 2007.
6. T. Grossman, G. Fitzmaurice, and R. Attar. A survey of software learnability: metrics, methodologies and guidelines. In *SIGCHI '09*, pages 649–658, 2009.
7. J. Iivari. Why are CASE tools not used? *Communications of the ACM*, 39(10):94–103, 1996.
8. J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, 1st edition, September 1991.
9. F. Linton, D. Joy, H. Schaefer, and A. Charron. OWL: A recommender system for organization-wide learning. *Educational Technology & Soc.*, 3(1):62–76, 2000.
10. C. Maltzahn. Community help: discovering tools and locating experts in a dynamic environment. In *SIGCHI '95*, pages 260–261, New York, NY, USA, 1995. ACM.
11. V. J. Marsick and K. E. Watkins. Informal and incidental learning. *New Directions for Adult and Continuing Education*, 2001(89):25–34, 2001.
12. J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. CommunityCommands: command recommendations for software applications. In *UIST '09*, pages 193–202, 2009.
13. J. McGrenere. *The Design and Evaluation of Multiple Interfaces: A Solution for Complex Software*. PhD thesis, The University of Toronto, 2002.
14. M. M. Müller and W. F. Tichy. Case study: Extreme programming in a university environment. In *ICSE '01*, pages 537–544, 2001.
15. S. Papert. *Constructionism*. Lawrence Erlbaum, 1991.
16. J. Rieman. A field study of exploratory learning strategies. *ACM TOCHI*, 3(3):189–218, 1996.
17. E. M. Rogers. *Diffusion of Innovations*. Free Press, 5th edition, 2003.
18. M. B. Twidale. Over the shoulder learning: Supporting brief informal learning. *CSCW*, 14(6):505–547, 2005.
19. L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 14th edition, March 1978.