Computer Supported Cooperative Work (CSCW) (2015) 24:389–422
DOI 10.1007/s10606-015-9230-9

How Do Users Discover New Tools in Software Development and Beyond?

Emerson Murphy-Hill[1], Da Young Lee[1], Gail C. Murphy[2] & Joanna McGrenere[2]
[1]*Department of Computer Science, North Carolina State University, Raleigh, NC, USA (E-mail: emerson@csc.ncsu.edu; E-mail: dlee10@csc.ncsu.edu);* [2]*Department of Computer Science, University of British Columbia, Vancouver, Canada (E-mail: murphy@cs.ubc.ca; E-mail: joanna@cs.ubc.ca)*

Abstract. Software users rely on software tools such as browser tab controls and spell checkers 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, we conducted two studies, an interview study and a diary study. The interview study describes a series of interviews with 18 programmers in industry to explore how tool discovery takes place. To broaden our findings to a wider group of software users, we then conducted a diary study of 76 software users in their workplaces. One finding was that social learning of software tools, while sometimes effective, is infrequent; software users appear to discover tools from peers only once every few months. We describe several implications of our findings, such as that discovery from peers can be enhanced by improving software users' ability to communicate openly and concisely about tools.

Keywords: Discovery, Learning, Programmers, Programming tools

1. Introduction

Software tools such as the functionality to correct grammar in Microsoft Word (2012) or to recover recently closed tabs in Mozilla Firefox (2012), allow users to perform their tasks more efficiently and do things they were unable to do previously. We define a 'tool' broadly as any software that helps a user accomplish a task. This includes standalone programs like development environments and desktop publishing software, but also includes features or commands in those environments, like source code formatters and spell checkers.

Users have difficulty discovering tools that might be useful to them. When we say that a user discovers a tool, we mean that she becomes aware of that tool. For example, when Grossman and colleagues (2009) conducted a study of 10 users of a computer-aided drafting application, they found that a "typical problem was that users were not aware of a specific tool or operation which was available for use". As another example, Campbell and Miller (2008) have noted that awareness is a problem in integrated development environments that are used by programmers.

Arguably, in any sophisticated software, many users will remain unaware of the full range of tools available.

The focus of this paper is on social solutions to the problem of lack of awareness, namely where a user learns about a tool from another user. There has been relatively little research into such solutions. In contrast, there has been considerable research into technical solutions. Some applications attempt to solve this problem with tip-of-the-day messages or role-based customizations of the user interface (Findlater et al. 2008). Researchers have proposed other technical solutions as well, such as recommender systems that suggest tools that you are not currently using (Linton et al. 2000; Maltzahn 1995; Matejka et al. 2009). These systems attempt to assist or replicate users helping other users, such as ToolBox (Maltzahn 1995), which helps Unix users find new commands based on the commands that their coworkers are using. As Fischer and colleagues (1984, p. 115) point out, these technical solutions "should guide and advise an user [sic] similar to a knowledgeable colleague or assistant", so understanding how such knowledge is transferred socially should help inform the design of such technical solutions.

To illustrate what we mean by social solutions to the awareness problem, let us give an example drawn from one of the studies that we describe in this paper. FEZ is a programmer who often works with another 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 followed in an instant-messaging session. In this session, FEZ gained awareness of a tool that he later found very useful. We call this mode of discovery **peer interaction**, where users discover tools from their peers during normal work activities.

In this paper, we investigate the intertwined social and technical contexts that allowed FEZ to discover a useful tool from a peer, yet sometimes make it difficult for other software users to discover other useful tools. In our first study, we explore these contexts by focusing on programmers, both because we are conversant with the tools that programmers use and because the range of tools available to programmers is so

(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. During a remote screen sharing session, a snippet of an instant-messaging log shows where one user learns about a software tool from another user.

wide. In our second study, we extend our participant group to software users in general to compare and contrast our results from the first study. The technical contexts we explore in this paper are complex software environments, such as programming editors and desktop web browsers. The social contexts we explore are broad, representing many different workplace environments where software users collaborate to discover new tools. Our long-term research goal is to encourage all kinds of software users to discover useful tools more successfully, more frequently.

This paper is an extension of a conference paper presented at CSCW 2011 (Murphy-Hill and Murphy 2011), where we presented the first study. In that paper, we made three primary contributions:

- an enumeration of the modes in which programmers discover new tools;
- a characterization of peer interaction, a mode of discovery where programmers learn about the existence of new tools from peers; and
- evidence that peer interaction may be the most effective way for programmers to learn new tools, yet it appears to occur infrequently.

In this paper, we additionally describe a diary study that provides a broader account of tool discovery, adding two additional contributions:

- a comparison between tool discovery in software development and tool discovery for other software users engaged in information work other than software development; and
- a more accurate quantification over a wider domain of tool use of how often various modes of tool discovery, including peer interaction, occur in practice.

2. Related Work

We base our work on several existing theories of adoption and learning, described in several areas of related work.

2.1. Tool Discoverability

Two existing studies have looked at tool discoverability directly. First, in prior work we interviewed software developers to understand why they adopt security tools (Xiao et al. 2014). Although that study is similar to the first study we report in this paper, the present paper seeks to understand tool adoption in a wider context: beyond security tools and beyond software developers. Second, the diary study that we present in this paper is similar to a study performed by Rieman (1996). Rieman asked 14 computer users to keep a diary of daily activities and specific learning events. The main difference between Rieman's diary study and our own is that we seek to characterize how users discover tools that they were not intending to learn (we call this *serendipitous* discovery), whereas Rieman largely reported on how users purposefully sought out help on how to complete a task with tools (we call this

purposeful discovery). Of the 60 tool discovery events recorded by Rieman, only 7 were serendipitous; in contrast, in our analysis of our diary study we describe 45 reports of exclusively serendipitous tool discoveries.

2.2. 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" (Rogers 2003). Typical studies of Diffusion of Innovations include research about internet use, hybrid corn in the US, and water sanitation in developing countries. Similar models have been developed for more specific contexts, including the Technology Acceptance Model (Davis 1989; Venkatesh and Davis 2000), Perceived Characteristics of Innovating (Moore and Benbasat 1991), Theory of Planned Behavior (Ajzen 1991), and Model of Personal Computers Utilization (Thompson et al. 1991). Both studies presented in this paper can be considered Diffusion of Innovation studies that investigate tool discovery.

Several studies have investigated Diffusion of Innovations in other software engineering contexts. For example, Fichman and Kemerer (1999) describe how programming languages, relational databases, and Computer-Aided Software/Systems Engineering (CASE) tools are acquired and deployed in organizations. Similarly, Iivari (1996) 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. 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 tools such as source 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 development environments have been adopted by organizations, our research additionally helps to explain how and why software users discover tools within those environments.

2.3. Social Learning

Tool discovery is closely related to learning, in that discovery can be thought of as part of certain learning theories.

One related theory is Lave and Wenger's (1991) situated learning, 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 Wenger have largely studied 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 Wenger 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." Our studies confirm this implication.

Another type of learning is Marsick and Watkin's (2001) informal and incidental learning, where learning happens as a by-product of other activities. Marsick and Watkins note that with this type, "control of learning rests primarily in the hands of the learner." In contrast, in peer interaction, learning is controlled by two people. We extend research on informal and incidental learning into the domain of software.

The zone of proximal development (Vygotsky 1978), the distance between what a learner can do on her own and what she can do with help from more capable peers, is also related in that learning about tools during peer interaction is within the zone of proximal development. While the zone of proximal development has applied to help people learn while using software (Borthick et al. 2003; Crook 1991; Luckin 2001), we do not believe it has been applied to understanding how people learn software itself.

Yet another related concept is over-the-shoulder learning, where colleagues help each other informally to use a computer application (Twidale 2005); 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, as defined by Twidale (2005), is focused on situations where the learner purposefully asks for help, not in situations when the learner discovers a new tool serendipitously. In peer interaction, the learner does not initially know that she might find a tool useful.

2.4. Learning During Programming

Some existing work has explored software tool learning specifically in the domain of programming, such as Cockburn and Williams' (2000) description of learning of tools from peers during pair programming. Specifically, they describe peer interaction happening 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. We hypothesize 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 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**.

Note that, in both cases, the learner did not expect beforehand to learn something new. Thus, in this paper, we focus on unexpected learning events, where a learner does not realize she needs a tool before she learns about it.

In pair programming, Cockburn and Williams suggest that:

> 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 suggest 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 (2000) and Müller and Tichy (2001) provide evidence that student programmers learn a variety of technologies during pair programming, but as M¨uller 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 people 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? Does it happen to non-programmers as well? In this paper, we extend Cockburn and Williams' and M¨uller and Tichy's findings by providing a more detailed analysis of the conditions, process, and results of this kind of learning for software users. We begin with an interview study of programmers, then attempt to generalize our findings through a diary study of a variety of software users.

3. Two Studies of Peer Interaction

Programmers work with environments that contain thousands of tools (Murphy-Hill et al. 2012), and the number of available tools is frequently expanding because new "plugins" are often added to such environments; as a result, we began our research of peer interaction in the domain of programming. Using Cockburn and Williams' research as a starting point, we conducted an interview study (Section 3.1) to determine how peer interaction works and how it relates to other modes of discovery, such as Twitter (2012) and exploring an application's user interface. After the interview study, we conducted a diary study (Section 3.2) to determine how these results generalize to a wider variety of information workers.

3.1. Study 1: Interviews

To better understand peer interaction, we wanted to collect a substantial number of descriptions of peer interaction, so we conducted retrospective interviews for several reasons. First, we suspected that peer interaction occurs so infrequently that direct

observation is impractical; this suspicion was confirmed. Second, interviews better enable 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. Our interviews are an instance of the Critical Incident Technique, where researchers gather observations about individuals' contributions to an activity (Flanagan 1954). Readers of our prior CSCW paper (Murphy-Hill and Murphy 2011) may safely skip this section because it does not differ significantly from the study we presented there.

3.1.1. *Methodology*

We conducted one-on-one, semi-structured telephone or instant-messaging interviews lasting about an hour each. The interview script can be found in the Appendix (Murphy-Hill et al. 2015). The interview began with questions to ascertain the participant's programming experience. Next, we defined tools broadly as "something that helps you perform a task," and referred the participant to a document that listed several pictures of different kinds of programming tools, which we chose from the Eclipse (2012) and Visual Studio (2012) development environments, as well as the extensible editors Vim (2012) and Emacs (2012). Although retrospective interviews are commonly used in this type of research, the results can be influenced by people's memory of discovery and adoption. Therefore, we used the tool list to help stimulate the participant's memories of tools that she might have discovered, using them as recall cues for known-item memory retrieval (Allen 1989). We asked the participant 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 are the most frequently occurring modes.

We then asked each participant to choose, in her experience, the two most effective modes for discovering new tools, from a list of seven different purposeful discovery modes, as shown in Table 1. We also encouraged the participant to think of other modes. We then asked the participant which, in her experience, are the two least

Table 1. Seven discovery modes, as read to participants.

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

effective modes. We defined effectiveness as how impactful each mode is on "your likeliness to use a tool again."We defined effectiveness in this way because we assumed that, if a user is likely to use a tool in the future, the user believes that the tool will be useful to her.

At this point, we revealed to participants that we were specifically interested in peer observation and peer recommendation, and asked for the participant to describe her experiences learning new tools in those modes. We asked the participant 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 participant directed questions about her experience with peer interaction, including how often she learns or teaches, and how it has changed over her career. Finally, we asked the participant some opinion questions, then thanked the participant 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 emerged, such as by the location in which the discovery took place. Similar to open coding (Glaser and Strauss 2009), 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.

3.1.2. *Participants*

We recruited participants 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 the conference.

Overall, 18 people responded and completed the interview, comparable to the size of similar studies such as those by Twidale (2005) (5 participants) and Rieman (1996) (14 participants). Participants 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. Participants were between the ages of 21 and 51 (median=30.5). Participants 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 (2012), emacs (2012), Visual Studio (2012), TextMate (2012), Eclipse (2012), and Netbeans (2012). Participants reported using a total of 24 different languages within the last year; the common ones being python (2012), Flanagan (2006), PHP (2012), Ruby (2012), Java (2012), C (2012), and perl (2012).

Participants reported a variety of working experience. We will refer to participants 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 participant reported.

Overall, participants had between 3 and 32 years of experience, with a mean of 12 years of experience. In the next column, we list whether or not each participant works on a team with other programmers in their current or most recent job. Overall, 12 of 18 of participants worked on teams. The next two columns show which participants regularly read technical blogs — websites where people post regular writings on technical topics — and which participants are users of Twitter. Overall, 13 of 18 participants used blogs and 14 of 18 used Twitter. We were interested in blogs and Twitter because we suspected that they played a role in tool discovery. We will explain the right two columns of Table 2 in the next section.

3.1.3. *Results*

Overall, participants reported 41 different instances of peer interaction, of which 27 were peer observation and 14 were peer recommendation. In this section, we describe the steps involved in peer observation and peer recommendation, how frequently

Table 2. Participants' pseudonyms are displayed in the leftmost column; pseudonyms assigned alphabetically based on participants' experience level (in years).

Pseudonym	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		✓	✓	○	−

Next, potentially programming-relevant social activities are listed. Finally, likeliness to learn or teach tools via peer interaction is listed. ● means that a participant learns via peer interaction between once every week and twice per month; a ◖ means that a participant learns every 1 or 2 months; and a ○ means that the programmer learns between once every 3 months and once per year. Programmers estimated that they taught less often (−), about equally often (≈), or more often (+) than they learned via peer interaction.

peer interaction occurs, how effective peer interaction is compared to other modes of discovery, the barriers to successful peer interaction, and how tool knowledge flows between peers. At the end of each subsection, we briefly summarize our findings.

The Steps of Peer Observation. Based on our interviews, 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 participants in four kinds of situations (Table 3): traditional pair programming, remote pair programming, happenstance interaction, and change notification.
- *Tools Observed*. Participants 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*. Participants reported that the learner almost always interrupted work to question the teacher, typically 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 tools 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 participants reported that typically the teacher gave an explanation of what the tool did and a short demonstration (less than a couple of minutes). Several participants also reported that they followed up with this discovery episode by trying the tool out when the teacher and learner separated. Other participants reported being given URLs by the teacher for later reference.

In sum, participants reported that peer observation occurs in pair programming situations, consistent with other researchers' observations (Cockburn and Williams 2000), but also in other situations where two programmers are not working on the same task. Rather than passive discovery, participants reported that the observer interrupted the other programmer verbally (or by instant-messaging, if the interaction

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.	*Peer Observation*. While programming with a coworker, DEL noticed Firebug when the coworker used it to solve a problem. *Peer Recommendation*. KEN was recommended the Open Type dialog in Eclipse while pair programming.
Happenstance Interaction	One programmer observes another during a chance encounter.	*Peer Observation*. KAI had the Labview development environment on his screen, which caught a coworker's attention when the coworker walked by. *Peer Recommendation*. 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.	*Peer Recommendation*. 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.	*Peer Recommendation*. 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.	Peer Observation. 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 tool 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.	*Peer Observation*. 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.	Peer Recommendation. 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.

was remote), which was followed by an immediate discussion and demonstration, or post-discovery exploration and reading.

- *The Steps of Peer Recommendation*. Based on our interviews, 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 alternative tool exists, the teacher interrupts the learner, and then the teacher delivers the recommendation.
- *Recommendation Situation*. Participants 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 participants 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 participants reported this immediate interruption. For instance, HAL described watching a colleague repeatedly open classes inefficiently, and recommended the Open Type dialog after some time:

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. (HAL)

- *Tools Recommended*. Participants mentioned a variety of tools that they had learned about via peer recommendation, including program navigation tools (such as Open Type in Eclipse and emacs tags), debugging tools (such as Firebug), and data transfer tools (such as Postgres and FTP).
- *Recommendation Delivery.* As with peer observation, most participants 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, participants reported that peer recommendation happened in similar circumstances to peer observation, with similar follow-up. However, in contrast to peer observation, where the interruption was often made with little hesitation, during peer recommendation participants 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, before we told them we were specifically interested in peer recommendation and peer observation. 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 that were mentioned. In Figure 2, a name in a box represents one participant's description of an instance of discovery. The number of boxes in each mode is the total number of instances of discovery that participants mentioned. For example, participants mentioned a 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, participants estimated that they learned and taught via peer interaction fairly infrequently. The right two columns of Table 2 indicate how often participants reported learning or teaching a tool via peer interaction.

While we expected that participants in a team would report more instances of discovery via peer interaction than participants not in a team, that hypothesis appears not to be supported by Table 2. Something that the table suggests that we did not expect was that the most experienced participants (ROB, VAL, YIT, ZAC) learned via peer interaction relatively infrequently. This infrequency might suggest that experienced participants have more to teach than to discover. However, three out of four of these more experienced participants taught as often or even less often than they learned this way. This suggests that more experienced programmers may be simply less involved in the practice of peer interaction.

When we asked participants to describe how their learning via peer interaction has changed over their careers, participants generally believed that peer interaction tended to be higher in their initial years as programmers, and has since decreased.

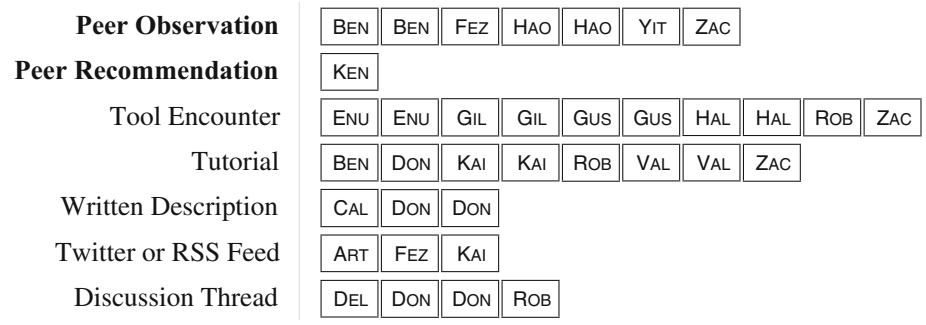

Figure 2. Histogram of the most frequent discovery modes.

They attributed this decrease to two sources. One source was an environment that was initially suitable to peer interaction (for example, school projects and internships), but then later in their career being in environments that were less suitable (for example, distributed development teams). The other source that participants mentioned was becoming more accustomed to their tools and having fewer new features to discover within those tools. DON, however, reported the opposite trend; he has become more likely to learn via peer interaction as his peer network has grown.

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 encounters are the most frequent way of discovering tools in a variety of software applications (Rieman 1996).

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 participants to name their two most effective modes, though we did not force participants 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 participants that we were particularly interested in these two modes.

– *Effectiveness of Peer Interaction*. Participants 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).

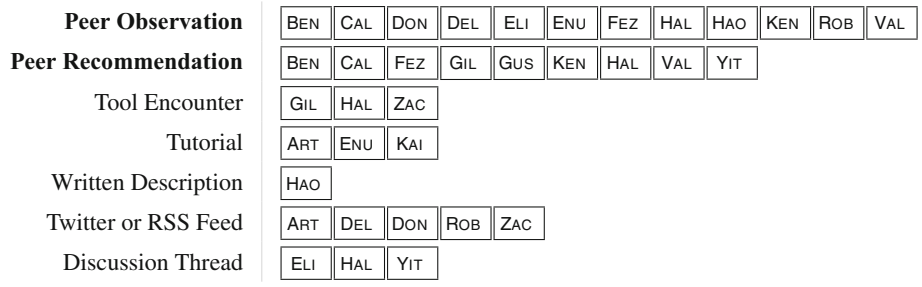

Figure 3. Histogram of the most effective discovery modes.

Also, participants found peer observation effective because:

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

– *Effectiveness of Twitter/RSS*. Participants 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 participants 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), some messages feel like advertising (CAL), and most messages are not relevant to programming (KEN). Moreover, HAL felt that Twitter/RSS and discussion threads were ineffective for the same reasons: the author does not have a similar background to the recipient.
– *Effectiveness of Discussion Threads*. Participants reported that discussion threads are effective because participants reported having trust in the sources (YIT) and because discussion threads tend to have a high level of detail (KAI). Others reported them being ineffective because the people who post are outside of their trust network (ROB, GUS, CAL).
– *Effectiveness of Tutorials*. Participants said that tutorials are effective because they fit with their personal learning style (ART) and because tutorials, specifically in the form of online screencasts, typically have a real-world example and are "highly rewindable" (KAI). Other participants reported that tutorials are ineffective because the tools used in tutorials are esoteric, not useful, or not widely available (FEZ), and because tutorials require a significant investment of time (YIT).
– *Effectiveness of Tool Encounters*. Although some participants reported that tool encounters (finding a tool by exploring the user interface) were effective, none gave a rationale. Some participants reported that tool encounters were ineffective because the environment that they use does not lend itself to exploration (BEN, DEL, HAO), a tool encounter takes too long (HAO), tools found in this manner are easily forgotten (DON), and because exploration tends to lessen as the programmer becomes more familiar with their environment (ENU).
– *Effectiveness of Written Descriptions*. Finally, HAO found written descriptions to be effective because that is how he currently learns. Others found written descriptions ineffective because of lack of trust or because there is a suspicion of marketing (GIL, GUS, KAI, ROB).

In sum, participants 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, pointing out that while exploratory learning may be the most frequent kind of tool discovery, "it may not necessarily be the most efficient or effective method of learning how to use a system" (McGrenere 2002). Indeed, these results confirm that exploratory learning is not the most effective. Another important finding is that trust was the most commonly cited determinant to the effectiveness of discovery, whichever the mode.

Barriers to Peer Interaction. Participants 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 hard to observe other programmers remotely, although other programmers reported instances of 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; participants 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 with peers, a self-selection bias. However, participants 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, participants 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 initially plan to ask participants 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 (Lave and Wenger 1991), 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 between 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 participants 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 programmers] 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 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 Section 4.

3.1.4. *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 participants noted that it was difficult to remember instances of peer interaction. This difficulty of recall may have affected the fidelity of the results, especially when we asked participants 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. When we did ask participants a general question, such as to estimate discovery modes' effectiveness, we preceded the general question with other questions that focused on specific instances.

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

The study may suffer from sampling bias, because the programmers are not representative of all programmers, for two main reasons. First, although we did not ask participants about their cultural or geographic background, we suspect that participants 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.

Finally, another threat to validity is whether participants' understood what we meant by "tools." We attempted to mitigate this threat by providing both a definition and examples. The tools mentioned by participants fell within our broad definition,

though participants may still have been overly conservative in their interpretation of the word.

3.2. Study 2: Diaries

In our second study, we had two primary goals: first, to obtain more recent accounts of tool discovery than could be provided by interviewees during the first study, and second, to examine how our results from programmers generalize to a broader population of information workers. To meet these goals, we asked a variety of software users to keep an electronic diary of when they discovered new tools at work.

This study augments the results of our first study in four ways. First, while our interview study focuses on only programmers, this study additionally included other types of information workers. Second, because our participants use types of software beyond programming environments, the types of tools that they discover are broader. Third, rather than depending on participants' memories of discovery events, we attempted to capture discoveries soon after they occurred. This complements the methodology used in the interview study by reflecting users' more concrete and recent tool discovery experiences. Fourth, in this study we investigated the effect of tool discovery modes on longer-term adoption as a way to assess the effectiveness of different discovery modes.

3.2.1. *Methodology*

In this study, we recruited participants from academia and the larger community by offering participants the chance to win a tablet computer. When participants consented to participate, they submitted a short screening survey.

We conducted a *diary study* of what software users are learning about new software tools at their work. Over a 13 week period, each week we randomly selected people from our volunteer pool to participate. We chose 13 weeks as the length of the study because it is the length of a typical co-op work term at the University of British Columbia, so any co-op students that participated would likely be working during the study. During the study, for 1 week each participant submitted a webbased report whenever they discovered a new tool at their jobs. This report included questions about how the participant discovered the tool.[1] To answer this question, the participant chose one of the discovery modes listed in Table 1, or optionally filled in an "Other" mode. They also chose one of four *intents*, "I was in the process of purposefully learning how to use the software", "I had a problem, so I looked for a [tool] to solve that problem," "I discovered the feature by chance," and "Other." The first two options indicate *purposeful* intent, while the third indicates *serendipitous* intent.

[1] In the reports, we used a term "feature" rather than "tool" because we felt it was more understandable to general software users. We will continue to use the word "tool" in this paper for consistency. Nonetheless, the definition we gave participants in the diary study for "features" was the same one we gave interviewees for "tools," that is, "something that helps you perform a task."

The report included several other questions, including which software the tool was discovered in and whether the participant took any steps to remember the tool. After participants submitted reports, the first author of this paper wrote follow-up emails to participants when more elaboration on a discovery was warranted.

At the end of the each participant's week of participation, we asked the participant to fill out a post-study questionnaire, which included information about perceptions of effectiveness of discovery modes, demographics, and software experience. A blank discovery report and post-study questionnaire can be found in the Appendix (Murphy-Hill et al. 2015).

About 6 weeks after a participant submitted a tool discovery report, we sent a follow-up questionnaire to determine whether the participant was still using the tool.

3.2.2. *Data Cleaning*

In manually reviewing discovery reports, we found that several reports were not internally consistent. For example, one participant reported learning a tool while browsing the internet forum, but classified it as "tool encounter." However, the correct classification for this mode of discovery was "discussion thread." As another example, one participant reported that the intent for discovering a tool in Excel was "purposefully learning how to use the software," yet should have instead classified the intent as "I had a problem, so I looked for a [tool] to solve that problem" because the participant searched Google to find a solution to a problem she was having.

To correct such inconsistencies, we recruited three raters to re-code each report's discovery mode and intent. To do so, each rater performed the reclassification by manually reading each report and using a pre-defined classification rubric that contained definitions for each discovery mode and intent. If two of the three raters agreed on each re-classification, the new classification was considered correct. If none of the raters agreed, the classification was treated as a missing value for the remainder of the analysis.

To determine whether how internally consistent our new ratings were, we measured agreement between different raters using the Fleiss' kappa (Fleiss et al. 1981) value. The higher the kappa value, the more inter-rater reliability: Fleiss' kappa values between 0.61 and 0.80 are considered as substantial, and values between 0.81 and 0.99 are considered as almost perfect (Landis and Koch 1977). For discovery mode, the kappa value was 0.86, while for intent, the kappa value was 0.79. These kappa values indicate strong agreement between the three raters.

3.2.3. *Participants*

We postulated that peer interaction occurs not only between programmers, but for a variety of technology workers as well. To examine this postulate, we recruited two different groups of participants. The first group was students from a variety of majors participating in summer-term work co-ops; we contacted these students through the University of British Columbia's co-op offices. We call this group "co-op students," as shown in the first major row in Table 4. The second group was recruited from the

Table 4. Participant demographics.

Participants	Specializations	Variables	Mean	Min	Max	n*
Co-op students	Computer Science (n=16)	Age (years)	22.4	20.0	26.0	9
		Job Length (months)	2.8	1.0	7.0	12
		Experience (months)	11.8	0.0	32.0	12
	Electrical and Computer Engineering (n=12)	Age (years)	20.3	19	22	6
		Job Length (months)	2.2	0.8	4	6
		Experience (months)	7.8	0	19	6
	Other Majors (n=23)	Age (years)	22.6	19	43	15
		Job Length (months)	16.7	0.5	240	18
		Experience (months)	25.9	0	264	18
Workers	Software Developer (n=5)	Age (years)	34.3	33	35	3
		Job Length (months)	31.5	2	72	4
		Experience (months)	126.0	96	180	4
	Non-Software Developer (n=20)	Age (years)	32.9	20	55	14
		Job Length (months)	25.7	1	72	15
		Experience (months)	122.7	0	396	15

*n (2*nd column) denotes the number of participants. The last column n* represents the number of participants (out of *n* participants) who filled out the corresponding demographic variable.

general public in British Columbia; we contacted these participants using an advertisement posted in public libraries and on local websites. We call this group "workers," as shown in the second major row in Tables 4 and 5.

We required participants to meet the following three qualifications: the participant anticipated working during the 13 week period; the participant anticipated spending at least of 50% of their work time using computer software; and the participant anticipated working a minimum of 35 hours per week. 51 co-op students consented to participate and 25 workers consented, a total of 76 participants.

We classified participants into four different categories. Co-op students were divided into computer science (n=16), electrical and computer engineering (n=12), and other majors (n=23) such as biochemistry, cognitive systems, and statistics.

Table 5. How often tools that were discovered using each discovery mode were used again 6 weeks later.

Discovery Modes	Frequently	Occasionally	Not at all
Peer Recommendation	0% (n=0)	80% (n=4)	20% (n=1)
Peer Observation	0% (n=0)	100% (n=1)	0% (n=0)
Written Description	0% (n=0)	50% (n=1)	50% (n=1)
Tool Encounter	54% (n=7)	31% (n=4)	15% (n=2)

n denotes the number of discoveries.

Workers were divided into software developers (n=4) and non-software developers (n=20), such as designers, bookkeepers, librarians and medical office assistants. When randomly choosing participants to participate each week, we took a stratified sample from these categories so that, for example, at least one computer science student participated each week. We used stratified sampling to prevent temporal bias, so that, for example, we collected students' discovery experiences from the beginning, middle, and end of their co-ops.

Table 4 shows mean, minimum, and maximum values for each sub-group with regard to age, job length, and experience. Job Length refers to the number of months participants have been at their current jobs. Experience indicates the number of months of work experience participants have. Note that some of participants did not fill out all of demographic information; the last column n* represents the number of participants who filled out data for each variable.

3.2.4. *Data Characterization*

In this study, 76 participants submitted a pre-study questionnaire. Of those, 50 submitted a post-study questionnaire and at least one tool discovery report. 5 submitted a post-study questionnaire, but did not submit any discovery reports. 4 submitted at least one discovery report, but did not submit a post-study questionnaire. 17 submitted neither tool discovery reports nor a post-study questionnaire. 156 tool discovery reports were submitted by 54 participants. Of the 156 reports submitted, participants also filled out 84 follow-up questionnaires, indicating how often they were using each tool, 6 weeks later. Of the 76 participants who filled out the pre-study questionnaire, 51 were co-op students and 25 were workers.

To be consistent with the previous interview study, for the following analysis (with one clearly marked exception; Table 6), we only included tool discovery reports where the user reported that they discovered the tool serendipitously. This excluded reports where the user was purposefully in the process of learning how to use the software and when the user purposefully went looking for a tool to solve a problem that they were having. Participants were not aware that we were exclusively interested in serendipitous discovery. Of the 156 tool discovery reports submitted, 45 of them met this criterion.

Table 6. The amount of off-task time (OTT) for different modes and intents for discovery.

Discovery Modes	Intent	*n*	OTT (sec)
Peer Observation	Discovered the tool by chance	2	240
	Had problem, looked for a tool to solve	0	.
	In process of learning to use the software	0	.
Peer Rec.	Discovered the tool by chance	8	97
	Had problem, looked for a tool to solve	22	410
	In process of learning to use the software	7	930
Tool Encounter	Discovered the tool by chance	31	26
	Had problem, looked for a tool to solve	21	245
	In process of learning to use the software	5	533
Written Desc.	Discovered the tool by chance	3	120
	Had problem, looked for a tool to solve	40	753
	In process of learning to use the software	4	680
Discussion Thread	Discovered the tool by chance	0	.
	Had problem, looked for a tool to solve	4	390
	In process of learning to use the software	1	1200
Twitter/RSS Feed	Discovered the tool by chance	1	60
	Had problem, looked for a tool to solve	0	.
	In process of learning to use the software	0	.

n represents the number of participants for the corresponding discovery mode and intent.

3.2.5. *Results*

Frequency. To estimate how often peer interaction occurs, we measured how many times participants' reported discovering a tool via peer interaction. Figure 4 shows the frequency of each discovery mode. For example, there were two reports of peer observation: one (in blue ▮) from a computer science co-op student and the other one (in pink ▮) from a non-software developer. Note that no "Tutorial" or "Discussion

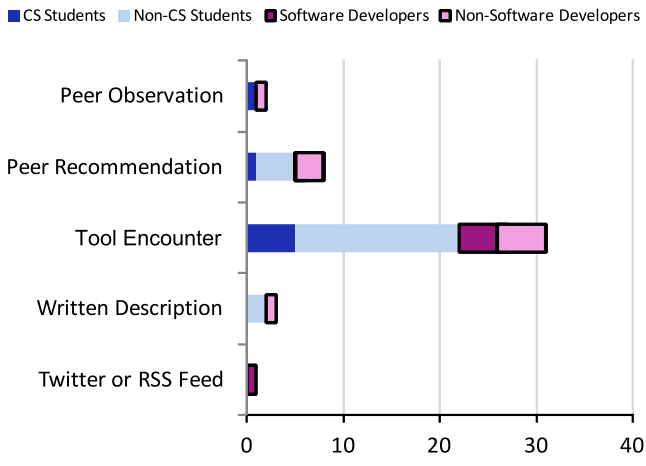

Figure 4. Histogram of the most frequent discovery modes in the diary study (n=28 participants).

Thread" bars are shown because no participant submitted this type of discovery report.

Figure 4 shows that participants reported tool encounter more frequently than any other mode of discovery, consistent with the findings in our interviews (Figure 2). The figure also shows that participants experience peer recommendation more frequently than peer observation, in contrast to our interview findings (Figure 2). If we assume that all participants reported all instances of peer recommendation, this data suggests that the average software user will discover a new tool via peer recommendation about once every 2 months. If we make the same assumptions about peer observation, the data suggests that the average software users will discover a new tool via peer observation only about once every 8 months. These projections are even less frequent than our interviewees' estimates of learning via peer interaction (Section 3.1.3).

Effectiveness. As with our first study, we wanted to estimate how effective each mode of discovery is for adopting new tools. In this study, we did this in two ways.

First, we estimated effectiveness in the same way that we did in the interviews, by asking participants' to rank the modes from most effective to least effective in their post-study questionnaire. Figure 5 shows a histogram that displays the top two highest ranked discovery modes for each participant. For example, the figure shows that in total 11 people reported that written descriptions were one of their top two most effective ways that they learned about new tools. When we asked this question in our interviews, participants rated peer observation and peer recommendation as the most effective modes for discovering a new tool; however, in this study, this was not the case. In particular, computer science students and software developers tended to rank tool encounter as their most effective mode. Interestingly, however, students

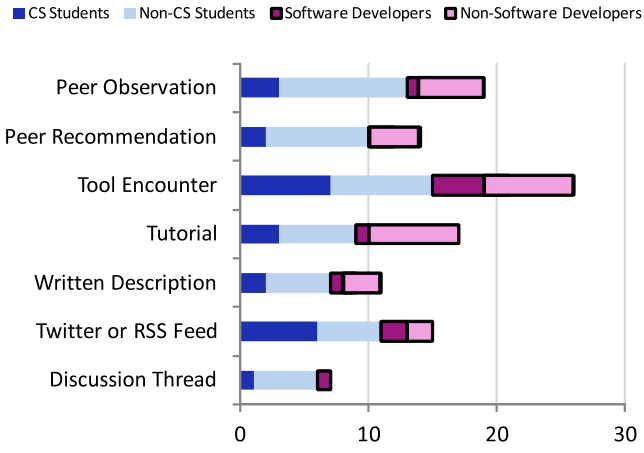

Figure 5. Histogram of the most effective discovery modes estimated by participants in the diary study (n=50 participants, 2 modes per participant).

who were not in computer science tended to rank peer observation and peer recommendation as their most effective modes.

Discovery Frequency by Age and Experience. In our interview study, we noticed that more experienced programmers tend to teach and learn via peer interaction less than junior programmers (Table 2). To investigate this further, we evaluated whether age, job length and experience is correlated with discovery frequency. Using a Pearson correlation, we found that age is very weakly correlated with number of tool discoveries ($r=-0.03$ and $n=47$). Similarly, job length and experience show weak correlations with number of discoveries ($r=-0.1$ and $n=55$, and $r=0.06$ and $n=55$, respectively). This suggests that users discover new tools at all levels of experience.

Off-Task Time for Tool Discovery. In addition to understanding the effectiveness of discovery modes, we are also interested in understanding the efficiency. To do so, we analyzed how long participants spent learning about new tools, with respect to discovery modes and intent. Table 6 shows our results, where we notice two trends. First, tool encounter tends to be the fastest discovery mode, in terms of off-task time. Second, finding a tool by chance was consistently the fastest intent under which tools were discovered, across all modes.

3.2.6. *Threats to Validity*

Although in this diary study we attempted to strengthen and generalize the results of the interview study, there are still several threats to the validity of the diary study that the reader should consider when interpreting our results. One threat was that, in follow-up emails, some participants noted that they made mistakes when categorizing a discovery by mode. We attempted to mitigate this threat by having three raters recode the data based on the remainder of the discovery report (Section 3.2.2). Another threat is that the study may suffer from sampling bias because participants were not representative of all software users, in part because participants were all working in British Columbia, Canada. A third threat is that, because participants sent tool discovery reports at their leisure, they may not have been motivated to write especially detailed reports, reducing the fidelity of our results. We tried to mitigate this thread by asking for additional information via email, when necessary. Another consequence of having participants report tool discoveries is that it probably made participants especially reflective about the tools they used, thereby increasing the probability that they would use the tool again at some point in the future. Participants' self-estimates of the elapsed time it took to discover a tool may not have reflected the wall-clock time to discover the tool, a phenomenon that has been observed in experimental psychology (Huttenlocher et al. 1990). While the time estimate may not be accurate to the actual elapsed time, if the inaccuracy is consistent across all modes, we can still reasonably compare the time estimates against one another. Finally, as the interview study, another threat to validity is whether participants' understood what we meant by "tools."

3.3. Study Comparison

In both our interview and our diary study, tool encounters dominated as the most frequent way that software users discovered new tools, both for programmers and other types of software users. Peer interaction was rarely reported in either study, both relative to tool encounters and in absolute terms; based on the diary study, peer interaction would only occur every few months. Peer interaction may be even rarer than we thought after conducting the interview study; even though we had about four times as many participants in the diary study than in the interview study, we only collected about a quarter as many episodes of peer interaction in the diary study.

The assessed effectiveness of peer interaction was not consistent between the studies. In the interview study, it appeared clear that programmers believed that peer interaction was effective, but in the diary study, programmers (and other workers) did not rate it as especially effective. We believe this disagreement probably arises partly from differences between the groups of participants, but also partly from differences between how participants interpreted "effectiveness" in the two studies. Further research is necessary in a more controlled setting to evaluate the effectiveness of the various discovery modes.

In our interview study, participants' remarks suggested that tool learning may decline with age and experience, yet the diary study did not show any substantial correlations between such maturity measures and the number of tools discovered. This agrees with previous work on technology learning in the workplace (Brooke 2009), which suggests that there is a perception that older technology workers are less able to cope with new technology, even when research suggests the opposite (Morrison and Murphy-Hill 2013). Indeed, our second study suggests that learning, at least in terms of software tools, continues unabated throughout information workers' careers.

4. Implications

While preliminary, our results have several implications. We discuss four of them in this section: how software environments can make it easier for users to discover tools from peers, how teams can encourage peer interaction, implications for the design of recommender systems, and methodological implications.

4.1. Improving Tool Discoverability

Our diary study suggests that users discover new tools at all levels of experience (Section 3.2.5, *Discovery Frequency by Age and Experience*), underscoring the importance of discoverability for all types of users. We suggest that there are at least two ways toolsmiths can make tools more discoverable, so that when users interact, peer interaction is more likely to be successful.

– *Noticeable Causes*. If the manner in which a tool is used can be easily observed, then an observing user is more likely to both recognize that a tool

was used and, implicitly, know how the tool is used. A positive example of such a tool — and one mentioned by several programmers in our interview study— is Eclipse's Open Type dialog box, which shows a noticeable dialog box used for searching. Hotkeys, used in many software environments so that users can quickly invoke commands, such as Ctrl+] in Microsoft Word to increase text size, 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. This is a collaborative extension of Nielsen's (1993) "visibility of system status" usability heuristic.

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. Another counter example is Microsoft Powerpoint's "Set Layout" command, which may not produce any obvious visual changes, yet links a slide to a slide template, so that future changes to the template are reflected in the slide.

4.2. Peer Interaction Without Collocation

Our results from both studies suggest that peer interaction is already very rare (Figures 2 and 4). As teams increasingly do more and more work in a remote fashion, our section on barriers to peer interaction imply that it will become even more challenging for remote teammates to learn tools from one another.

– *Remote Pair Programming*. Several interviewees reported learning new tools via peer interaction with a peer by working at separate, remote 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 an audio link. 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. Although we

know of no equivalent of pair programming for general information workers, we believe that these constraints apply equally to situations where remote information workers wish to learn from their peers.

– *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, micro-blogging (Twitter) and internet tutorials (screencasts) may be increasingly common. However, both of these typically lack the qualities that make peer interaction effective, such as that peer interaction takes place in the context in which the tools are used. 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, yet few interviewees and zero participants in the diary study learned about tools via Twitter. One reason that interviewees cited for Twitter being ineffective is lack of trust in the sources and lack of relevance (Section 3.1.3, *Effectiveness of Twitter/RSS*). From the interviews, it appears what programmers meant by trust was that the recommender (human or otherwise) requires some prior interaction with the recommendation recipient, so that the programmer can estimate the recommender's knowledge and skills (Section 3.1.3, *Discovery Modes*). Our studies suggest that trust and relevance might improve if the Twitter messages originated from a trusted peer, someone that a software user 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 microblog messages to coworkers on her behalf.

Several interviewees reported watching screencasts that were professionally produced (e.g., peepcode.com). Although watching screencasts is similar to peer observation, participants reported that the tools used in screencasts may not be very relevant (Section 3.1.3, *Effectiveness of Tutorials*), presumably because the people who made the professional screencasts did not have working styles that aligned with individual interviewees' working styles. Because peers are more likely to have similar working styles, a screencast produced by a peer is potentially more relevant. However, no participant in either study reported watching a screencast produced by a peer. We suspect that the reason that software users do not make screencasts for their peers is that the costs (the time required for recording, editing, and distributing) are too high compared to the benefits (the possibility of a peer discovering a tool). KAI, from the first study, 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 software users will produce screencasts for their peers, improving screencasts' effectiveness.

4.3. Design of Recommender Systems

Systems that can recommend relevant tools are one approach to helping users learn new tools. Past systems have included the controversial Clippy (http://www.microsoft.com/presspass/features/2001/apr01/04-11clippy.mspx), which takes the form of a virtual talking paperclip to recommend features of Microsoft Word. In what follows, we discuss two design considerations for such systems.

– *Trust*. Interviewees' high valuation of trust (Section 3.1.3) underscores trust's importance in tool recommender systems. CAL, when asked about his opinion of a potential recommender system, summed up the problem as:

> Honestly; I bet the [recommender system] would have better success rate [than a peer] at recommending things that I would like, but that doesn't mean that I would trust the [recommender system] more.

Since trust appears to be a by-product of prior social interactions during peer interaction, how can a recommender system without any prior interactions be trusted at all?

One way is to borrow trust from a trusted peer. Specifically, rather than saying, "You should use this tool," a system could instead say, "your friend, John, also uses the tool, so you should check it out,"[2] making a recommendation based on what your peers are using. In this way, the user no longer needs to have trust in the recommendation system beyond that it is accurately recording her teams' tool usage, and instead the user can rely on trust between teammates (Murphy-Hill 2012). This was the approach taken by Toolbox, where the system looked for expert tool users, requested that these experts write down descriptions of the tools, and published the descriptions as newsletters in the expert's organization (Maltzahn 1995). We have sketched such a system using screencasts for software developers in a recent paper (Murphy-Hill 2012).

Another way to avoid the trust dilemma is to sidestep it altogether. ELI mentioned that a recommender system might be acceptable if it makes "you feel like you discovered it." Implementing such a recommender system would require a more subtle interface than the traditional "you should use this tool" interface.

– *Beyond Peer Interaction*. Although recommender systems may have been originally inspired by recommendations offered by trusted peers, the design of recommender systems could benefit by going beyond traditional peer recommendation by closely examining the limitations of peer interaction.

Interviewees noted that peer interaction does not work when the peers are under time pressure because the teacher feels like he cannot respond to the learner's question, "what did you just do?" (Section 3.1.3, *Barriers to Peer Interaction*). If a

[2] A direct quote from YIT.

recommender system can capture experts' usage of tools, that usage can be played back repeatedly and at the convenience of a user who wants to learn a tool.

Peer interaction has other limits: just because a single software user uses a tool, that does not mean that other users would find it useful. However, if many users find a tool useful, it is more likely that a user who does not yet know it will also find it useful. Recommender systems could gather, summarize, and present the usage habits of the community, rather than relying on only one individual. This is the approach taken by some recommender systems, such as CommunityCommands (Matejka et al. 2009).

Interviewees noted that sometimes they did not make recommendations because they did not think that their peer was ready to appreciate the usefulness of a tool (Section 3.1.3, *Barriers to Peer Interaction*). One way that recommender systems could deal with this would be to explicitly model how specific users learn by monitoring what tools they know, and inferring discovery patterns and contexts. A recommender system could then recommend appropriate tools at the right time and in the right context, and therefore not only make relevant recommendations, but also improve the likelihood that the user will adopt the tool.

4.4. Methodological Implications

- *Individual Interpretation Differences*. As we speculated about in Section 3.3, different participants may have had different interpretations of the word "effectiveness." They may have also had different interpretations of the meaning of different modes of discovery (Table 1). Because we re-categorized discovery reports based on our own intended meaning, the impact of this potential issue was limited. Nonetheless, in future studies we intend to provide richer definitions to participants, congruent with the complexity we observed of each mode in this study.
- *Generalizability*. As we mentioned in the threats sections for both studies, the studies have limited generalizability due in part to the number of instances of peer interaction captured. In retrospect, the low number of instances that we were able to capture were inevitable, given that peer interaction apparently occurs so infrequently. A much longer-term diary study might have been able to capture more instances, but at the same time participants may have been unwilling to keep a diary of their tool discoveries for much longer than a week. One alternative would have been to have them focus on peer interaction only, rather than reporting all modes of tool discoveries. A focus on peer interaction may, however, encourage more social tool discoveries than would normally take place. Ultimately, our goal of capturing realistic peer interaction in the field remains a challenge that is difficult to overcome.
- *Within-Group Comparisons*. As a consequence of the somewhat low number of instances of peer interaction, our ability to make reliable within-group comparisons was limited. For example, in Table 5, we compared how different discovery modes correlated with participants' likelihood to continue to use a tool 6-weeks after the initial discovery. In that table, most cells have four or

fewer instances of tool discovery, making it difficult to draw strong conclusions about the effectiveness of different modes.

One method that would enable more reliable comparisons between groups would be a laboratory experiment. For instance, we could teach several undergraduate students about a given new tool, exposing groups to different discovery modes, then ask them several weeks later if they were still using the tool. Fortunately, based on the results from the present studies, we can craft several realistic learning scenarios for such an experiment.

5. Conclusion

A wide variety of tools have been built to help software users, but individuals must necessarily discover those tools before they can be used. While there are many ways that users discover useful tools, the interview study described in this paper suggested that peer interaction is effective but infrequent. Our diary study suggested that tool encounters occur more frequently than peer interaction. Successful peer interaction occurs in social contexts where users have peers that they can observe, such as in a collocated team, but also in technical contexts where such observations are facilitated, such as when one user can see the outcome of another user's tool use. Towards our goal of encouraging software users to discover useful tools more successfully and more frequently, our results open new possibilities to make existing tools and environments more discoverable and distributed collaboration more effective.

Acknowledgments

Thanks to the interview participants. Thanks also to Christian Bird, Andrew Black, Kellogg Booth, Giuseppe Carenini, Cristina Conati, Nando de Freitas, William Griswold, Ciarán Llachlan Leavitt, Karon MacLean, Rahul Pandita, 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 IBM, NSERC, as well as by the National Science Foundation under grant number 1252995.

References

Ajzen, Icek (1991). The Theory of Planned Behavior. *Organizational Behavior and Human Decision Processes*, vol. 50, no. 2, December 1991, pp. 179–211.

Allen, Bryce (1989). Recall cues in known-item retrieval. *Journal of the American Society for Information Science*, vol. 40, no. 4, 1989, pp. 246–252.

Borthick, A. Faye, Donald R. Jones, and Sara Wakai (2003). Designing learning experiences within learners' zones of proximal development (ZPDs): Enabling collaborative learning on-site and online. *Journal of Information Systems*, vol. 17, no. 1, Spring 2003, pp. 107–134.

Brooke, Libby (2009). Prolonging the careers of older information technology workers: continuity, exit or retirement transitions? *Ageing & Society*, vol. 29, no. 2, February 2009, pp. 237–256.

C (2012). C. http://www.open-std.org/jtc1/sc22/wg14/.

Campbell, Dustin and Mark Miller (2008). Designing refactoring tools for developers. In D. Dig, R. Fuhrer and R. Johnson (eds): *WRT'09. Proceedings of the 2nd Workshop on Refactoring Tools, Nashville, US, 19 October, 2008,* New York: ACM Press, pp. 1–2

Cockburn, Alistair, and Laurie Williams. (2000). The costs and benefits of pair programming. In *Proceedings of the Conference on Extreme Programming, Cagliari, Italy, 21 June–23 June, 2000*, pages Boston: Addison-Wesley, pp. 223–247.

Crook, Charles (1991). Computers in the zone of proximal development: Implications for evaluation. *Computers & Education*, vol. 17, no. 1, pp. 81–91.

Davis, Fred D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, vol. 13, no. 3, September 1989, pp. 319–340.

Eclipse (2012). Eclipse IDE. http://www.eclipse.org.

Emacs (2012). emacs. http://www.gnu.org/software/emacs/.

Fichman, Robert G., and Chris F. Kemerer (1999). The illusory diffusion of innovation: An examination of assimilation gaps. *Information Systems Research*, vol. 10, no. 3, September 1999, pp. 255–275.

Findlater, Leah, Joanna McGrenere, and David Modjeska (2008). Evaluation of a role-based approach for customizing a complex development environment. In D. Tan (ed): *CHI'08: Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems, Florence, Italy, 5 April–10 April, 2008*. New York: ACM Press, pp. 1267–1270.

Firefox (2012). Mozilla Firefox. www.mozilla.org/en-US/firefox.

Fischer, Gerhard, Andreas Lemke, and Thomas Schwab (1984). Active help systems. In G. C. van der Veer, M. J. Tauber, T. R. G. Green, P. and Gorny (eds): *Readings on Cognitive Ergonomics – Mind and Computers*, Berlin: Springer-Verlag, pp. 115–131.

Flanagan, John C. (1954). The critical incident technique. *Psychological Bulletin*, vol. 51, no. 4, pp. 327–358.

Flanagan, David (2006). *JavaScript: the definitive guide*. Newton: O'Reilly Media, Incorporated.

Fleiss, Joseph L. (1981). The measurement of interrater agreement. *Statistical Methods for Rates and Proportions*, vol. 2, pp. 212–236.

Glaser, Barney G., and Anselm L. Strauss (2009). *The discovery of grounded theory: Strategies for qualitative research.* Alexandria: Transaction Books.

Grossman, Tovi, George Fitzmaurice, and Ramtin Attar (2009). A survey of software learnability: metrics, methodologies and guidelines. In K. Hinckley, M. R. Morris, S. Hudson, S. Greenberg (eds): *CHI'99. Proceedings of the Twenty-Seventh Annual SIGCHI Conference on Human Factors in Computing Systems, Boston, US, 4 April–9 April*, New York: ACM Press, pp. 649–658.

Huttenlocher, Janellen, Larry V. Hedges, and Norman M. Bradburn. (1990). Reports of elapsed time: bounding and rounding processes in estimation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 16, no. 2, March 1990, pp. 196–213.

Iivari, Juhani (1996). Why are CASE tools not used? *Communications of the ACM*, vol. 39, no. 10, October 1996, pp. 94–103.

Java (2012). Java. http://java.net.

Landis, J. Richard, and Gary G. Koch. (1977). The measurement of observer agreement for categorical data. *Biometrics*, vol. 33, no. 1, March 1977, pp. 159–174.

Lave, Jean, and Etienne Wenger (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press, 1st edition.

Linton, Frank, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron (2000). OWL: A recommender system for organization-wide learning. *Educational Technology & Society*, vol. 3, no. 1, January 2000, pp. 62–76.

Luckin, Rosemary (2001). Designing children's software to ensure productive interactivity through collaboration in the zone of proximal development (ZPD). *Information Technology in Childhood Education*, vol. 2001, no. 1, pp. 57–85.

Maltzahn, Carlos (1995). Community help: discovering tools and locating experts in a dynamic environment. In I. Katz, I. Mack, L. Marks (eds): *CHI'95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Denver, US, 14 April–18 April, 1996*, New York: ACM Press, pp. 260–261.

Marsick, Victoria J., and Karen E. Watkins (2001). Informal and incidental learning. *New Directions for Adult and Continuing Education*, vol. 2001, no. 89, Spring 2001, pp. 25–34.

Matejka, Justin, Wei Li, Tovi Grossman, and George Fitzmaurice (2009). CommunityCommands: command recommendations for software applications. In F. Guimbretire (ed): *UIST'09: Proceedings of the Symposium on User Interface Software and Technology, Victoria, Canada, 4 October–7 October, 2009*. New York: ACM Press, pp. 193–202.

McGrenere, Joanna (2002). *The Design and Evaluation of Multiple Interfaces: A Solution for Complex Software*. Ph.D. dissertatino. University of Toronto, Ontario: Department of Computer Science.

Moore, Gary C. and Izak Benbasat (1991). Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation. *Information Systems Research*, vol. 2, no. 3, September 1991, pp. 192–222.

Morrison, Patrick, and Emerson Murphy-Hill (2013). Is programming knowledge related to age? An exploration of StackOverflow. In A. Bacchelli (ed): *MSR'13. Proceedings of the 10th Working Conference on Mining Software Repositories, Challenge Track, San Francisco, US, 18 May–19 May, 2013*. Los Alamitos: IEEE Press, pp. 69–72.

Müller, Matthias M. and Tichy, Walter F. (2001). Case study: Extreme programming in a university environment. In M. J. Harrold andW. Schafer: *ICSE'01. Proceedings of the 23rd International Conference on Software Engineering, Toronoto, Canada, 12 May–19 May, 2001*. Los Alamitos: IEEE Press, pp. 537–544.

Murphy-Hill, Emerson (2012). Continuous social screencasting to facilitate software tool discovery. In A. Bertolino and A. Wolf: *ICSE'12. Proceedings of the 34th International Conference on Software Engineering, 2 June–9 June, 2012*. Los Alamitos: IEEE Press, pp. 1317–1320.

Murphy-Hill, Emerson and Gail C. Murphy. (2011). Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In J. Bardram and N. Ducheneaut (eds): *CSCW'11. Proceedings of the Conference on Computer Supported Cooperative Work, Hangzhou, China, 19 March–23 March 2011*, New York: ACM Press, pp. 405–414.

Murphy-Hill, Emerson, Rahul Jiresal, and Gail C. Murphy (2012). Improving software developers' fluency by recommending development environment commands. In M. Robillard and T. Bultan: *FSE'12. Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Cary, North Carolina, 11 November–16 November 2012*, New York: ACM Press, pp. 1–11.

Murphy-Hill, Emerson, Da Young Lee,, Gail C. Murphy, and Joanna McGrenere (2015). Appendix to how do users discover new tools in software development and beyond? Technical Report TR-2015-7, North Carolina State University. ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc anon/tech/2015/TR-2015-7.pdf.

NetBeans (2012). NetBeans IDE. http://netbeans.org/.

Nielsen, Jacob (1993). *Usability Engineering*. San Francisco: Morgan Kaufmann Publishers Inc.

Perl (2012). Perl. http://www.perl.org/.

PHP (2012). PHP. http://www.php.net/.

Python (2012). Python. http://python.org/.

Rieman, John (1996). A field study of exploratory learning strategies. *Transactions on Computer-Human Interaction*, vol. 3, no. 3, September 1996, pp. 189–218.

Rogers, Everett M. (2003). *Diffusion of Innovations*. New York: Free Press, 5th edition.

Ruby (2012). Ruby. http://www.ruby-lang.org.

TextMate (2012). TextMate. http://macromates.com.

Thompson, Ronald L., Christopher A. Higgins, and Jane M. Howell (1991). Personal Computing: Toward a Conceptual Model of Utilization. *MIS Quarterly*, vol. 15, no. 1, March 1991, pp. 125–143.

Twidale, Michael B. (2005). Over the shoulder learning: Supporting brief informal learning. *Computer Supported Cooperative Work*, vol. 14, no. 6, December 2005, pp. 505–547.

Twitter (2012). Twitter. https://twitter.com/.

Venkatesh, Viswanath, and Fred D. Davis (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, vol. 46, no. 2, February 2000, pp. 186–204.

Vim (2012). vim. http://www.vim.org.

Visual Studio (2012). Visual Studio. http://msdn.microsoft.com/vstudio.

Vygotsky, Lev S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Cambridge: Harvard University Press.

Word (2012). Mirosoft Word. http://office.microsoft.com/en-us/word/.

Xiao, Shundan, Jim Witschey, and Emerson Murphy-Hill (2014). Social influences on secure development tool adoption: Why security tools spread. In M. R. Morris and M. Reddy (eds): *CSCW'14. Proceedings of Computer Supported Cooperative Work and Social Computing, Baltimore, Maryland, 15 February–19 February 2015*. New York: ACM Press, pp. 1095–1106.