Interruptions on Software Teams: A Comparison of Paired and Solo Programmers

Jan Chong, Rosanne Siino

Center for Work, Technology and Organization
Department of Management Science and Engineering
Stanford University
380 Panama Way, Stanford, CA 94305, USA
{jchong, rsiino}@stanford.edu

ABSTRACT

This study explores interruption patterns among software developers who program in pairs versus those who program solo. Ethnographic observations indicate that interruption length, content, type, occurrence time, and interrupter and interruptee strategies differed markedly for radically collocated pair programmers versus the programmers who primarily worked alone. After presenting an analysis of 242 interruptions drawn from more than 40 hours of observation data, we discuss how team configuration and work setting influenced how and when developers handled interruptions. We then suggest ways that CSCW systems might better support pair programming and, more broadly, provide interruption-handling support for workers in knowledge-intensive occupations.

Categories and Subject Descriptors

H5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Collaborative Computing*.

General Terms

Management; Human Factors

Keywords

Pair programming, collaborative work, ethnography, interruptions, eXtreme programming

1. INTRODUCTION

Interruptions are increasingly recognized as a natural and inevitable feature of collaborative work. Workers in virtually all industries and work settings must contend with all manner of disruptions to their work routines in the course of a day. Interruptions come from various sources and occur for many reasons [20], yet as we try to understand interruption dynamics and build systems to ease their handling, our general conception of interruptions has remained fairly narrow. Research has primarily considered externally-driven intrusions on individual workers, but the nature of work, the work environment and team configuration all may influence how workers handle both externally and self-initiated interruptions.

Knowledge-intensive work is a particularly interesting case for study because of its complexity. Interruptions are at times essential for swapping or gaining information required for high quality work. Software developers, for example, are subject to interruptions that may help them as they program, giving them insight into teammates' work or access to critical data. Given the mentally intensive nature of this work and our nascent understanding of interruption dynamics in such an environment, direct observations of a natural work setting have the potential to better inform further study and system design.

The work presented in this paper is drawn from a larger ethnographic study of software development practices. We studied two teams, one which practiced eXtreme programming (XP), a relatively new methodology that includes such unconventional practices as pair programming and radical collocation [6], and a team with more conventional practices, such as independent programming in individual cubicles. In the course of our observations, we noticed that interruptions and interruption-related behavior on the XP team differed markedly from that of the non-XP team. In this paper, we first present a description of the teams' configurations and an analysis of the differences in interruption dynamics. We then suggest how these insights might inform technology design to support this new form of work and, more broadly, improve availability awareness and interruption handling in computer supported collaborative work.

2. RELATED WORK

Observations of interruption activity in the workplace have been quite effective in helping us build an understanding of interruption dynamics. We will here review some of the existing studies of interruption behavior among knowledge workers as a basis for our current work.

Taken together, previous research reveals that collaborative knowledge work, as it occurs in the "wild", is a complex and fragmented activity in which workers must negotiate multiple tasks simultaneously and where interruptions are not only inevitable but necessary for work completion. An early study by Bannon *et al.* [4] of the command history of a research group observed that these computer users did not complete tasks in an orderly, linear fashion, but rather took on multiple tasks and switched between them. Czerwinski, Horvitz and Wilhite [10] found similar behavior in their diary study of task switching behavior, as did Gonzalez and

CSCW'06, November 4–8, 2006, Banff, Alberta, Canada.

Mark [13]'s study of analysts, developers and managers at work. Czerwinski, Horvitz and Wilhite's [10] participants reported an estimated 0.7 interruptions per task and indicated that they not only switched tasks of their own accord but were also frequently externally interrupted.

Studies in the workplace have consistently found interaction to be a significant component of knowledge work. Hudson *et al* [15] found that managers spent 46 percent of their time communicating with others and 19 percent of their time in unplanned communication. Whittaker, Frohlich and Daly-Jones [27], tracking two mobile knowledge workers, found that informal interactions were a significant component of the work, comprising 31 percent of total work time. They also noted that these interactions tended to be brief, context-dependent and opportunistic in nature. O'Conaill and Frohlich [21], drawing on the same data to look at disruptive, unscheduled interactions, concluded that such interactions were beneficial for the interrupted party but disturbed work flow, as workers often failed to resume the interrupted task. In Perlow's [22] ethnography of software engineers, interruptions were also found to be critical to work, but the high volume of interruptions coupled with a lack of control over interruption incidence led workers to report high levels of stress and frustration.

Taken together, these studies suggest that interruptions are a natural and necessary component of collaborative knowledge work. Although research seeking to understand task organization has noted that workers interrupt their own tasks to switch to another, many existing workplace studies, which focus on interactions rather than tasks, consider external interruptions only. Our study offers a look at both self and externally-initiated interruptions, defining an *interruption instance* to be any instance in which workers turn attention away from their primary work task, either on their own or in response to others' actions.

Although workers observed in previous studies took part in collaborative interactions, they primarily worked alone and were interrupted while engaged in independent work. By contrast, the programmers we observed fall into two categories: those that primarily worked alone and those that consistently worked in pairs, a practice called *pair programming*. Developers in the latter category formed ad hoc pairs each morning, each pair sitting side by side at a shared computer, developing code together throughout the day.

Working in pairs presents an interesting case for a study of interruptions. Unlike previous studies of joint work [14, 17], the paired programmers share a single task, a single machine and, for the most part, the same role on a continuous basis. While pair programmers claimed to have two separate roles – the *driver*, who controlled keyboard input and primarily considered issues of immediate implementation, and the *navigator*, who provided more strategic feedback and guidance – in practice, we found that the term "driver" was used largely to denote keyboard and mouse control. Outside of keyboard and mouse duties, the paired programmers occupied much the same role, with no distinct gap in the level of abstraction at which they considered their programming problem (as noted through conversational content and observed interaction). We know of no prior studies of pair programming as it occurs *in situ* or of how interruptions might affect knowledge workers configured in pairs rather than individually. Our current study therefore aims to increase understanding of paired work and to inform tool design to support this new work form.

3. RESEARCH SITE

Our data is drawn from ethnographic observations of two software development teams at a mid-sized startup company in Silicon Valley, California. The programmers on one team, which we will call "Team Pair," worked in pairs as a part of the team's use of the XP software development methodology. Team Pair was radically collocated, working in a large, open, bullpen-like space, giving team members both visual and aural access to the actions and dialogue of other members of the team. Programmers consulted with teammates throughout the day, almost exclusively through face-to-face communication. The developers had e-mail, but e-mail use was extremely limited, primarily for purposes of communicating with members of the company outside of the team. Programmers did not generally use the telephone for work purposes, but frequently received personal calls throughout the day. These programmers almost never worked from home, partially due to the requirement that code be written in pairs.

The programmers on a second team, which we will call "Team Solo," worked primarily alone, but frequently consulted with others on the team through face-to-face communication, e-mail, telephone and lily, a computer mediated communication tool designed and developed at Rensselaer Polytechnic Institute. Programmers on Team Solo worked in individual cubicles in the same area of the building. This configuration allowed team members to overhear dialogue and conversation taking place in adjacent cubicles, even though other team members were not visually accessible. Unlike Team Pair, e-mail was an important means of inter-team communication. It was also common practice for the Team Solo programmers to work from home several days a week, using the lily to maintain contact with team members at work. Lily is a text-based chat service that allows users to log in and communicate with other users, either via private message or shared discussion lines. The tool was widely adopted within the company, and was used not only by Team Solo but by other functional groups and several members of management (although notably, it was not used by Team Pair). Solo developers used lily daily, often logging into the tool whenever they were engaged in company-related work regardless of where and when that might be. The more enthusiastic users logged on upon waking up and logged off right before heading to bed. Team Solo developers also commonly called into team meetings via telephone when working remotely. Outside of team meetings, they used the phone primarily for personal communication.

Both the work structure and environment made work more visible and more social for programmers on Team Pair than for those on Team Solo. Writing code in pairs meant that programmers on Pair were engaged in constant interaction with their pair partners. The need to communicate as a part of this interaction meant that programmers generated a steady stream of conversation during work. Pair programmers also shared a physical space, a rich sensory environment that allowed for physically and temporally immediate interactions in the course of work. By contrast, Team Solo programmers were more physically isolated from each other but consistently shared a virtual space. Thus, their work interactions were asynchronous and text-based – widely accessible across geographic and temporal distance, but more limited in sensory context.

Both Solo and Pair generally held a morning status meeting where all team members gave a brief update on the previous day's progress as well as on any issues that had developed in the course of work. We watched both teams release a version of their products to customers and then engage in planning activities for the next release. Consequently, our observations of each team spanned a "crunch" period of intense pressure, as the team tried to make release deadlines, and a more leisurely planning period, reducing the chance that the observed behaviors were exclusive to activities associated with a particular project phase.

4. METHODOLOGY

We conducted ethnographic observations of the developers at work approximately once a week for a total of seven months. Each observation session tracked activities of either a particular programmer (for Team Solo) or a particular pair of programmers (for Team Pair). We sat physically behind the programmers as they worked, taking extensive notes throughout the session. Whenever possible, we audio recorded dialogue that transpired, and then transcribed and integrated it into the field notes to produce a detailed description of both action and dialogue.

Across both teams, we gathered more than forty hours of observation data, with a total of 242 observed interruptions. We had 21 hours and 41 minutes of observation data for Team Pair and 18 hours and 29 minutes of observation data for Team Solo. During this time, we observed 138 interruptions on Team Pair and 104 interruptions on Team Solo. All subjects explicitly identified their primary work task for us as each observation session began, which allowed us to easily identify interruptions instances. Primary work tasks included such activities as adding new functionality to the software code base, optimizing project code to performance expectations and fixing software bugs.

Drawing on the taxonomy presented by Jett and George [18], we categorized each interruption as either an intrusion, break, distraction or discrepancy. Due to the nature of our data, discrepancies, defined by Jett and George as "perceived inconsistencies between one's knowledge and expectations and one's immediate observations that are perceived to be relevant to the task at hand and personal well-being", were difficult to conclusively detect; thus, this category was subsequently dropped from our analysis. For each interruption instance, we noted the source of the interruption, the duration of the interruption, the activities being conducted before the interruption, the content of the interruption and the pair or the individual's activities immediately after the interruption.

In the following section, we present a descriptive portrait of interruption characteristics, content and behavior across the two teams. We will then argue that these differences are caused by differences in work and team configuration and discuss the implications for the design of systems that might help decrease negative effects of interruptions.

5. FINDINGS

Our analysis of the ethnographic data revealed numerous differences in interruption instances between the two teams. Pair and Solo's interruptions differed by length, content, type, time of occurrence, interrupter strategy and interruptee strategy.

5.1 Interruption Length

Interruptions on Team Pair were consistently shorter than on Team Solo, regardless of interruption type and interruption source (see Table 1 for details). The average length of an interruption on Pair was 1 minute and 55 seconds. The average interruption length on Solo was 2 minutes and 45 seconds.

5.2 Interruption Content

To analyze each interruption's content – the activities comprising each interruption – we categorized interruptions by type using the taxonomy proposed by Jett and George [18].

5.2.1 Intrusions

Jett and George define an intrusion to be "an unexpected encounter initiated by another person that interrupts the flow and continuity of an individual's work and brings that work to a temporary halt." Intrusions tend to be immediate in nature and much of their disruptive impact derives from the perceived need to respond promptly to the needs of interrupter.

On Team Pair, intrusions initiated by other team members were primarily functional in nature, consisting mainly of requests for information, requests for help, delivery of task-relevant information, efforts to coordinate actions or information across the team and, occasionally, requests for status updates. These intrusions varied greatly in length and scope, and discussions between group members could come to involve the entire group if the issue had implications beyond original discussants. In the following excerpt from our data, Dana interrupts Ben and Carlos, who are working as a pair, to ask a question about locking, and the ensuing discussion grows to encompass the entire team. Andy and, later, Eric, both working in separate pairs, join the conversation. Notice that neither Andy nor Eric is explicitly asked to give an opinion; they simply join the conversation because they overhear it and have relevant questions:

Dana: [*to Ben and Carlos*] Did you stop with the locks?
Carlos: Kevin said something about just adding some methods somewhere to lock an entity.
Ben: For each operation factory, isn't there a common object factory it extends?
Carlos: Yeah.
Ben: So we can have common generic test methods and stuff.
Andy: [*joining the discussion from his seat*] As far as the methods – what semantics are you using?
Dana: What's not, as far as semantics, is if you lock and another user – does it prevent you from updating?
Carlos: No, you should be able to update.
Ben: One would think it would and we don't check for that right now.

Table 1. Average Interruption Length by Team

	Team Pair		**Team Solo**	
	#	Average Length (M:S)	#	Average Length (M:S)
Self-initiated	39	2:39	41	3:46
Externally-initiated	99	1:38	62	2:23
Distraction	20	0:58	21	1:28
Intrusion	94	1:40	51	2:17
Break	24	3:42	30	5:08

Dana: [*agreeing with Ben*] We don't have that right now.
Eric: [*joining in the discussion*] We don't have what? The edit entity update test?
Ben: Well, what's cool about the generic API integration is that we can automatically have tests everywhere.
Dana: If you go to the UI and you lock and object, and I try to update an object-
Ben: It's the same lock.
Eric: Yeah, but I don't know if that's the best response.

Some intrusions on Team Pair were purely social in nature, but these were rare and, by comparison to the functional intrusions, relatively short – usually a two- or three-line conversational exchange. In the following example, Andy, Ben and Carlos joke about Andy's initials. After the exchange, the three developers, who are working in separate pairs, return promptly to their respective tasks:

Andy: Ah, no, it's like my first name is Walter, but nobody knows that so when I do my initials, people are like, huh?
Carlos: [*looking up*] So, you're Wally?
Andy: Yeah, I'm Wally. I'm Wak. [*Someone chuckles*] Yeah, I got a lot of abuse for that.
Ben: W-A-K?
Andy: Wak, yeah.
Ben: It's like Yak, except with a W.
Andy: I make milk. [*Ben laughs*]

All Team Pair members carried cellular phones, which were another major source of intrusions. Team members usually answered phone calls immediately, leaving the work area if the conversation showed signs of continuing beyond a few seconds.

On Team Solo, by contrast, intrusions were both functional and social in nature. Intrusions were longer and generally involved movement – team members physically visited another team member's cubicle. Here, Feroz and Henry arrive at Grant's cubicle to ask him about the gag gifts that Grant brought back from a conference. In the course of the intrusion, their discussion moves from talk about the gifts (datagram pads) to a discussion of the current product release status. The intrusion draws Ingrid over to Grant's cubicle roughly halfway through the interruption, and she relays the fate of some cake in the office kitchenette, which she had accidentally dropped on the floor earlier.

Feroz: [*to Grant*] So I was wondering. You were handing out these pads of datagrams. And it occurred to me, "Huh. Grant hasn't give me one. But Grant left two in Henry's cube. And I was sitting in Henry's cube at the time…"
Grant: [*teasing him*] You know, Feroz, if you want, you can ask Henry if he wants both of those pads.
Henry: You can have one if you want.
Feroz: I was wondering and I was thinking, "Maybe that's why he left two? Maybe Henry's just special."
Grant: Henry is special. I figured he would get the joke first and laugh the hardest so. And he did.
Henry: If they ever have a IPV6 version of it, it won't fit.
Grant: [*picks up the note pad and reads*] Refer to RFC 6121c! Maybe next year we'll see if they have a bigger one.
Henry: [*changing the subject*] Should I check in code to work with the scanning stuff?
Grant: You seem to have gotten it working haven't you?
Henry: I've seen RTS use it and it works on my machine…
Grant: So, on the platforms that don't support it…
Henry: It does support it everywhere now.
Grant: Linux… and FreeBSD?
Henry: Not that I've tried. But it should work.
Grant: We're going to release this Wednesdays.
Ingrid: [*Appearing at the cubicle entrance*] I just found out that the XP guys [*Team Pair*] aren't on lily.
Feroz: Took you long enough.
Ingrid: I was realizing that as I was walking back to the kitchen and saw one of them eating the caramel tart.
Feroz: Ooops! [*laughing*]
Ingrid: On the other hand, it's almost finished, so I'm guessing there wasn't too much dirt on it.

Team Solo programmers, like those on Team Pair, also received phone calls frequently during work – calls that were almost always social in nature. Unlike the pair programmers, however, solo programmers remained in their workspace while on the phone, even though these conversations were frequently audible to all team members due to the proximity of their cubicles.

5.2.2 Distractions

Distractions are "psychological reactions triggered by external stimuli or secondary activities that interrupt focused concentration on a primary task." [18] We identified instances in the data where subjects' attention was diverted from the primary task by noting when they physically paused their work activity, either turning their heads towards the external stimuli or making a comment regarding the distraction.

Team Pair programmers worked in the midst of multiple overlapping ambient conversations because of the team's open shared workspace. These conversations served as an easy source of distraction. In the following example, Eric and Nathan are working as a pair, next to Levi and Mark. When Kevin begins to converse with Levi and Mark, Eric and Nathan will become distracted by the conversation:

Kevin walks over to talk to Levi and Mark. Levi says that a dialog box is too small and is cropping text. Eric and Nathan stop working and listen in to the conversation.

Eric: [*vaguely directed to Levi*] You've got to make shorter names. [*Nathan chuckles*]

Levi does not seem to have heard. Eric turns back and opens the application on his computer. Nathan continues to listen, watching Kevin and Levi until Kevin walks away.

On the whole, however, pair developers were accustomed to their work environment, and conversations of others only became problematic when they were held at particularly high volume. Developers seemed to find irregular noises far more distracting:

Eric gets up and walks over to his machine. He picks up a sandwich wrapper from the table and begins to fold it; the sound is very loud. Eric walks behind Owen and Dana, still folding. Owen and Dana look up at him, annoyed at the noise. Eric walks over to the trash can and throws away the wrapper.

Pair programmers also appeared to get distracted by physical objects in their work environment. The developers quite deliberately displayed the current project build status on a large

monitor, visually available to the entire room. The monitor displayed a large rectangle that was green if the build had passed all unit tests, yellow if the build was currently running and flashing red if any tests failed, in order to attract the attention of team members and alert them to news of the failure:

Eric, sitting at his computer looks up at the build monitor. It's flashing red for three out of the five builds.

Eric: Is anyone fixing the build?

People, particularly those unfamiliar to the team, served as another source of distraction. With seven team members working in the same room, Team Pair programmers were accustomed to each other's movements in and out of the work room, but visits from members of other teams were unusual and attracted attention.
Team Solo's programmers, however, worked in cubicles, both increasing the physical space between team members and blocking easy views of teammates. Although (or perhaps because) conversation and noises carried easily over cubicle walls, the area was extremely quiet, particularly in contrast to Team Pair's work area. Team members found unusual computer or telephone noises (beeps, buzzes, unusual ring tones) quite distracting; when they occurred, at least one team member would generally comment. Extended conversations were tolerated if they were technical in nature. Extended social conversations, however, were either shushed or seemed to serve as a beacon, bringing other team members out of their cubicles to join in.

Team Solo's programmers extensive use of lily, the CMC tool, served as another source of distraction. Lily had a host of public (public within the company) chat lines, as well as dedicated team discussion chat lines. Consequently, the lily window was almost constantly displaying a stream of new messages as various company employees posted to the chat forums. Changes in the window, when it was visible on the desktop, frequently drew developers' attention as they worked, particularly because group announcements and technical discussions often occurred on the team chat line during the day.

5.2.3 Breaks

Jett and George define breaks as "planned or spontaneous recesses from work on a task that interrupt the task's flow and continuity." The developers on both Team Solo and Team Pair took frequent breaks in the course of work.

On Team Pair, most breaks involved physically leaving the workspace. Developers would leave to make personal phone calls, run errands (usually requiring them to leave the office entirely), go to the restroom or retrieve drinks and snacks from the company kitchenette. When they were physically in the work area, developers took breaks by checking e-mail or looking at web pages when they had no immediate tasks assigned to them or if they were waiting for another developer.

On Team Solo, in contrast, breaks included a broader scope of activities. In addition to those listed above, developers also took breaks to write e-mail or chat on lily, sometimes as a respite from their current task. In the following, Feroz grows frustrated with the interface to a bug report database and stops to send a series of rapid online messages to the team discussion line:

Feroz switches to his lily window and begins to type:

From feroz to team-rds:
Okay.

From feroz to team-rds:
It's going to be… hard to set the priority of a ticket to 10 or 5

From feroz to team-rds:
The priority field appears to only offer 1, 2, 3, 4 as options.

Feroz re-examines the bug report interface and notices that there are multiple fields labeled "priority". He switches to his lily window again and types:

From feroz to team-rds:
Oh no wait, it's just STUPID

From feroz to team-rds:
List of fields: "Priority", "Final priority", "Priority"

Team Solo developers also took breaks for social purposes, by physically visiting other developers in their cubicles to chat or by sending messages over lily. Physical visits, in particular, seemed to draw the attention of other developers nearby, frequently causing multiple team members to join in. In the following example, Paul is perusing a lily conversation while waiting for a build to complete. He follows a web link posted on the chat forum, which leads to a music video. As the video begins to play, Feroz and Grant come to the cubicle to watch as well.

Paul, reading through his lily window, suddenly laughs.

Paul: Oh my god, I might have seen that.

Paul moves to his other machine and opens a web browser. He types in the URL. As music begins, Feroz enters the cubicle.

Paul: Oh, I have seen this! [*Grant arrives in the cubicle*]
Grant: What have you seen?
Paul: This horrible Bilbo Baggins movie.

The video plays. Then Grant suggests they watch another.

Overall, Team Solo's breaks were more likely than Team Pair's to have a recreational or social dimension and to involve multiple group members, while breaks on Team Pair seemed to primarily serve as periods of individual physical or mental refreshment or opportunities to attend to personal non-work related activities.

5.2.4 Summary of Content Differences

Patterns of interruption types were far more varied for Solo developers than for Pair developers. Some Solo developers took many breaks, for example, while others took very few. On Team Pair, distribution of interruption types was fairly consistent from day to day and from pair to pair. Overall, Team Pair's breaks and intrusions were functional, with individuals using them to gather task knowledge or to get personal work done during the paired task. For Team Solo, by contrast, breaks and intrusions were both functional and social. Finally, intrusions from team members occurred primarily through face-to-face communication on Team Pair, while intrusions occurred through e-mail, through lily and face-to-face for Team Solo members.

5.3 Interruption Incidence

On both teams, how interruptions occurred and the points at which they occurred differed depending on whether the interruption was externally initiated or self-initiated. Still, Team Pair generally responded quite differently than Team Solo to interruptions. We

will first present the two differing strategies for handling externally-initiated interruptions, and then, second, the strategies used for internally-initiated interruptions.

5.3.1 Externally Initiated Interruptions

Developers on both teams had little control over the times at which external interruptions occurred, but had strategies for both interrupting others and handling interruptions.

5.3.1.1 Interruptor Strategies

On Team Pair, programmers attempted to leverage the physical visibility of their team members to choose better times to interrupt. Interrupters could often be seen scanning the room (if they needed to make an announcement to the group) or monitoring a particular pair's activities and conversation in an attempt to minimize the disruptive effect of intruding. Their ability to wait for a good time to interrupt, however, was limited by the time pressure they felt to complete their work; at some point, the need for information would drive them to interrupt regardless of the status of the programmer to be interrupted. This situation was particularly true if both pair members were stymied in their task by lack of information; the pair would then often interrupt the programmer with the information, regardless of that programmer's current state of preoccupation.

On Team Solo, lack of visibility and the general silence in which work was done made it more difficult to determine "good interruption points." When a developer made the effort to physically visit another, he or she almost always immediately interrupted the person visited, unless the visitee was already occupied with another person (and sometimes the developer interrupted anyway). If the person sought was not physically in the office or if the question was of lower priority, the interrupter generally used lily to reach that person. In these cases, the interrupter often began by checking the status of the interruptee (connected and/or active) both to determine interruptee availability and to estimate how rapidly a response might come.

5.3.1.2 Interruptee Strategies

Developers on both teams used several strategies to handle interruptions when they were the ones being interrupted.

On Team Pair, developers had little control over intrusion timing and when intrusions could be addressed. The physical immediacy of the intrusion generally meant that handling could not be deferred. Pairs minimized an interruption's disruptive impact by having one member continue working while the other addressed the interruption. If the intrusion were directed at the entire group, pairs would assess if it was relevant to them and if they decided it was not, they would ignore it. In the following, Eric directs a question to the entire team, which Dana and Levi will assess and then proceed to ignore:

Eric: Is anyone fixing the build?
Dana: Not us. [*turning back to Levi*] Is that a zone file?
Eric: [*to the group*] Is anyone fixing the build?
Dana: [*to Eric*] Not us.
Ben: Yes, it's fixing itself.
Kevin: I've heard that a couple of times [*he laughs*]
Dana: [*to Levi, ignoring the larger conversation*] So that's...
Ben: I deleted a test.

The group debates Ben's decision to delete a test. Dana and Levi work quietly together, ignoring the entire conversation.

Unlike Team Pair, Team Solo programmers could minimize the impact of intrusions by limiting their own availability. Several solo programmers worked from home frequently, citing lack of disruptions as a reason for this practice. Queries to those programmers would then come through lily, where, due to the asynchronous nature of the chat medium, they could be ignored. Intrusions that came through lily could also be more easily filtered or addressed only in a cursory manner. Although programmers might intend to follow up later, these requests would frequently be forgotten without explicit reminders from the person asking the question. Team Solo programmers sometimes also opted to work through an interruption – turning back to their computers and resuming their tasks, even as social conversation was directed at them or went on around them. This multitasking was not possible when the intrusion came through lily, as the need to read messages in the lily window made it difficult to simultaneously work and monitor the interruption.

Team Pair programmers had less control than Solo members over interruption incidence, but they had more control over interruption duration. Because Team Pair programmers worked in pairs, the person being interrupted generally traveled to the site of the interruptor to handle the interruption. This left the interruptee free to depart the area when he or she felt that the interrupting issue had been adequately addressed, a determination that might vary with the import of his or her primary task. On Team Solo, however, physical intrusions involved the interruptor traveling to the cubicle of the interruptee; thus, when the interruptee felt that the interruption had been resolved, he or she could only signal to the interruptor that that he or she wished to return to work – but it was the interruptor who decided when to leave. Thus, if the interruptor ignored or was oblivious to the signal, the interruptee's only recourse was to resume work, despite the ongoing interruption.

5.3.2 Self-initiated Interruptions

On Team Pair, programmers took breaks as they saw fit. Breaks were slightly more frequent between tasks, when both members might take a break, or at stages when work could be handled by one of the pair. Breaks were distributed, however, rather randomly through the day. Developers often simply stood up and walked away, with little or no warning to their pair partners:

Dana: This doesn't work for relative links, so let's go to java.
Eric: Oh, they're not URLs. It's the URL class.
Dana: Let's see, do we want to start extracting or is this the class?

Eric suddenly gets up and bolts from his chair. Dana looks at the code and begins to create another class.

Periods of downtime – such as during test runs or the code check-in process where no interaction or input was required of programmers for several minutes – frequently led a member of the pair to take a break while the remaining member attended to the running process, doing little but waiting for it to complete.

Similarly on Team Solo, many self-initiated interruptions were also triggered by periods of downtime in their primary work task. While pair programmers could send off one developer to take a break while the other continued to monitor the task's progress, solo programmers did not have this luxury. Unlike Team Pair programmers, Team Solo developers rarely sat and waited for the primary task process to complete. Instead, developers guessed at when the process might complete and did something else in the

meantime. In addition to such activities as trips to the kitchenette or bathroom, the programmers would often take up a secondary task such as checking e-mail and lily or even working on a completely new task.

For Solo programmers, switching between a secondary and primary task was sometimes problematic, particularly when both tasks involved digital interfaces on the same display. The secondary task window often partially or fully obscured the primary, impairing the developer's ability to monitor primary task progress. This situation sometimes led to delays in primary task resumption as the solo programmer became engaged in break activities. In the following example, Henry decides to run the project's test suite to check on test failure status, a process that takes approximately five minutes. As he waits for the test suite to complete, he checks lily and then e-mail:

Henry: So I think that's… that may be the last thing I need to do to make the rtests. Now I think I'll run the rtests and see what else breaks.

He runs them and looks over his lily window while he waits. There is a private message from Feroz to Henry about a bug.

Henry: Huh? I don't want to know what's going on.

Nevertheless, he scrolls back to read over the conversation that Feroz's message references on the team-rds line. New messages from Paul to the team line appear at the bottom of the window. Henry turns to his e-mail.

Henry: Oh right, there was something I had to look at…

Henry types out a response to an e-mail; Ingrid, Paul and Feroz appear, ready to leave for lunch. After some discussion, Henry leaves for lunch as well.

As Henry grows more engaged in secondary tasks, he forgets to check his test run, which will remain unchecked until after lunch. Lacking an easy mechanism for tracking both tasks, solo programmers such as Henry resorted to polling, transitioning rapidly between two tasks in order to check on the primary task's progress. In contrast, this behavior was rarely seen on Team Pair, where one of the pair simply sat and waited out the process.

6. DISCUSSION

Our analysis indicates significant differences between the pair programmers (Team Pair) and solo programmers (Team Solo) in length, content, type, time, context of occurrence and strategies for handling work interruptions. We will now explore these differences before discussing implications for system design.

6.1 Why Interruptions Differed by Team

As noted above, interruption times were shorter for Pair vs. Solo programmers, regardless of interruption time or source (self-initiated or external). Research has shown that social exchanges and relational contracts can develop between individuals who work closely together, creating a sense of mutual obligation [8, 24]. Because their work was highly cooperative in nature, Pair programmers may therefore have felt a strong social obligation to their pair partners, leading them to handle interruptions quickly so that they could return to the primary task they shared with their pair partner. The programmers on Team Pair were much more likely to stay on task than the Solo programmers. Social interruptions for the pair programmers were also shorter in duration than those on Team Solo and they were less prone to picking up the secondary tasks that interfered with primary task resumption.

Several previous studies indicate that a work environment's physical artifacts can function as cues for interrupted tasks [23, 11, 13]. For Pair members, the uninterrupted programmer seemed to serve as that cue – a cue that was visible, audible and immediately present in the physical workspace, and which therefore maintained the interrupted task's salience for the interrupted programmer. The pair partner may reduce the time required to recover state by acting as a source of richer and more easily interpreted task information. Pairs can make greater use of multiple modalities such as speech and gesture to augment perception and understanding of the current task state, rather than relying primarily, as the solo programmers must, on interpretation of the incomplete task as presented by the application interface. Speech and gesture may also provide information to the returning programmer in a more accessible and potentially more parsimonious representational state [16]. Much the same way that conversational partners work "moment by moment, to identify and remedy inevitable troubles that arise" in conversation [26], pair programmer interactions provide a natural and seamless means for interruption recovery.

Because Pair members worked in an open, shared workspace, the interrupted programmer could also easily monitor his or her partner's primary task progress during the interruption. Thus, if the partner programmer appeared to be struggling or stuck, the interrupted programmer could hurry or attempt to end the current interruption in order to return to the task. Pair programmers may also have been more motivated to respond promptly to interruptions because their actions were so visible to the entire team. Thus, these programmers may have felt a pressure to balance being seen as a knowledgeable "team" player with the need to be an attentive and available pair partner.

Interruption content for Team Solo programmers also differed from that of Team Pair programmers. Team Solo interruptions, particularly intrusions, had a significant social as well as functional component, while Team Pair's interruptions were primarily functional in nature. For Solo programmers, the primary work mode was alone, either at home or within a cubicle. Thus, interruptions for Solo programmers likely served as a mechanism not only for gathering information but also for social interaction. In contrast, because pair programming requires constant communication between the pair, Team Pair members were engaged constantly in social interaction and seemingly had little need to interrupt others for social purposes.

The notion that Solo programmers used interruptions as a source of social interaction is also supported by the difference between the two teams in interruption-type variability among team members. Because individuals vary in social interaction needs, we would expect to see variation in the amount and types of interruptions across individuals on a team as those who need high levels of social interaction use interruptions more for that purpose than those with lower social needs. Indeed, we found that Team Solo members varied widely in types and quantity of interruptions across the group. Pair programmer interruptions, in contrast, were fairly similar in types and quantity across pairs, supporting the notion that they did not use interruptions for social interaction.

Intrusions occurred frequently for programmers on both teams, and individuals creating interruptions were largely insensitive to the current state of interruptees. For interruptors on Team Pair, time pressure and need for information generally outweighed significant

consideration of an interruptee's task status. Even when the relatively low priority of an interruption's content made consideration possible, many interruptions required the entire team's attention, limiting the impact of a particular pair's status on the decision to interrupt. On Team Solo, the programmers generally checked availability (via lily) before interrupting, but gave little to no consideration to the interruptee's status.

The physical configuration of Team Pair's work environment gave pair programmers a broad context for their interruption activities. Not only could interruptors see and hear activities of programmers they intended to interrupt, but also the interrupted programmers had rich information about the interruptor's situational context. This rich awareness likely increased programmers' willingness to respond quickly to external information requests (a common reason for intrusions); they possessed the situational awareness to make their own determination of the interruption's importance to the interruptor. Thus, if lack of response rendered an interrupting pair unable to continue work, a programmer being interrupted could see this situation and might be more willing to respond. Solo programmers, however, relied on the interruptor's estimation and, if they wished to make their own determination of an interruption's criticality, had to explicitly request context from the interruptor.

These differing contextual considerations are reflected in the two teams' differing interruption handling strategies. Team Solo programmers generally employed strategies that would benefit the interruptee, such as deciding to defer or ignore an interruption. Team Pair programmers, meanwhile, although often unable to avoid an immediate response to an intrusion, had greater recourse in influencing how the interruption unfolded. They had substantially more control over when and how to end physical intrusions: because Team Pair interruptees generally traveled to the interruptor, they had the ability to end an interruption simply by returning to their own space. And because work was done in pairs, these programmers could leverage obligations to their pair partners to manage interruption duration. This ability was reflected in differences between Pair members' responses to group-wide intrusions and pair-to-pair intrusions. Pair programmers had little control over group-wide intrusions and could only attempt to ignore them and carry on with their work. In pair-to-pair intrusions, however, Pair programmers frequently paused to monitor their partners' progress and, if they were needed, to act quickly to end the intrusion.

For self-initiated interruptions, both the sense of accountability to the pair partner and the visibility of their actions to the entire team likely kept Pair programmers from adopting secondary tasks or from taking long breaks. In contrast, primary task resumption lags occurred more frequently among Solo programmers, because they lacked immediate accountability to their teammates.

Finally, unlike Solo programmers, Pair programmers shared tasks between partners. This task sharing seemed to reduce the time required for interrupted partners to recover from the interruption, because the task itself was often not interrupted: one member of the pair generally continued to work while the partner handled the interruption. Thus, while Solo programmers were drawn out of their task and had to switch cognitive gears during an interruption, Pair programmers could rely on their partners to continue the primary task and to bring them back to it upon the interruption's conclusion. Previous studies document the adverse effects of interruptions on individuals [25, 3], but pair work seemed to allow the Pair programmers to reduce somewhat the derogatory effects of interruptions by dulling potential losses in productivity.

6.2 Implications for Design

As our analysis reveals, interruption patterns and dynamics for paired programmers differed in numerous ways from those of solo programmers. Our findings suggest a number of potential directions for the design of computerized work support systems.

Support Self-Interruptions. Many systems for interruption support primarily address deleterious effects of intrusions [11, 5, 7]. Our data indicate, however, that many interruptions during the workday are not from external sources but are in fact self-initiated. Our findings suggest that self-interruptions are a natural part of knowledge-intensive work. Systems to support knowledge work (including systems for interruption support) should therefore consider self-initiated as well as external interruptions, taking into account when and why people pause in the course of their tasks.

Provide contextual information for the interruptee. Prior work on interruption systems has focused either on when and how to present interruption information to the interruptee or providing contextual information about the interruptee to the interruptor [12]. The dynamics of interruption response among the radically collocated programmers on Team Pair suggest that providing the interruptee with easily processed, easily accessible and independent contextual information about interruptor may ease the adverse affects of intrusions [20]. Interruptees can weigh the need for information on the part of the interruptor with their own schedule and priorities and against the activities of the team as a whole. In teams and organizations where cooperation is necessary for work, this may facilitate cooperation and reduce annoyance at intrusions.

Refine contextual information for the interruptor. The pair programmers in this study had available both the visual and aural context of the person or pair they intended to interrupt, but they rarely postponed intrusions for significant amounts of time. Most pairs appeared to use a rough and potentially inadequate heuristic for interruptability, waiting for particularly animated conversations to die down before calling over to the pair in question, if they chose to wait at all. This finding suggests that if contextual information is to be useful in guiding the timing of interruptions, it must be refined so that interruptors can better weigh the urgency of their requests against the interruptees' sense of their own interruptability at a particular time.

Ease awareness of multiple tasks. For the solo programmers in our study, interruptions were often occasioned by the need to wait for long-running processes, such as compilations or test runs, to complete. While waiting, the solo programmers commonly took breaks or engaged in secondary tasks. The programmers often found it difficult to monitor the status of a primary task while engaged in a secondary one, frequently leading to unnecessary delays in primary task resumption or a rather frenetic polling of the primary task application. Our observations suggest that these programmers, along with knowledge workers in similar task environments, would benefit from improved system support for maintaining general awareness of changes in application state, along the lines of Matthews *et al.*'s work on clipping lists [19]. Providing a consistent, generalizable interface for monitoring state changes in multiple applications would reduce time and energy spent polling for primary task progress and facilitate task resumption after mental breaks.

Give interruptees greater control over interruption duration and conclusion. As with much knowledge work [27, 22], interruptions are a critical part of programming, facilitating communication of

key information across a team. Differences in interruption duration and conclusion across the two teams suggest that giving interruptees control of when and how interruptions end may help attenuate interruptions' disruptive effects. An interface, for example, that sounds an alert when a specified time has passed in which the computer is idle might provide a graceful "excuse" for interruptees to turn back to their own tasks and signal interruptors that an intrusion should end.

Increase task salience and interactive cues to ease task resumption. On Team Pair, partners played an important role in helping programmers return to their primary tasks after interruptions. Pair partners served to make relevant components of the task salient to the returning partner and acted as an interactive, easily accessible means for the programmer to recover current task state. By contrast, solo programmers had no such resource, bearing the full burden of remembering their task state prior to interruption and increasing the need for such strategies as rehearsal [1, 9] and physical cues [2]. Applications could potentially facilitate task resumption in similar ways, perhaps by emphasizing recently accessed or particularly relevant task elements upon a return from an interruption or by providing programmers and other knowledge workers with an easy means to visualize work history.

Consider the special needs of pair programmers. Our study provides a window into the interruption implications and practices of workers engaged in a relatively new form of cooperative work. As noted above, pair programming differs from forms of cooperative work previously studied [14, 17] in that workers are jointly oriented about a single physical artifact (the computer) in a shared physical space and do not divide task labor into distinct roles. Because pair programmers differ markedly from solo programmers in their interruption behavior, systems to support this brand of collaborative work must incorporate substantially different considerations from those for either more traditional forms of collaborative work or individual work. These systems might, for example, provide a simple means by which a member of a pair when left alone could record his or her actions so that the pair partner can review them quickly, when necessary, upon returning from an interruption, or so that subsequent pairs can review work done by others on previous days.

7. CONCLUSIONS

The consideration of how interruptions may be supported has centered primarily on a conception of knowledge work as individually conducted and of interruptions as externally triggered disruptive tasks. Our data shows that a substantial number of interruptions during the workday are self-initiated. Interruptions are a natural and sometimes essential part of work, particularly knowledge intensive work. Our comparison of interruption behaviors in a solo programming team and pair programming team demonstrates that joint work may have different implications for the impact of and potential support for interruption handling.

8. ACKNOWLEDGMENTS

We are grateful to Diane Bailey for her guidance during the formative stages of this work, and to Pamela Hinds and Scott Klemmer for their feedback and input on this paper. We thank our informants for participating and for their kind tolerance while being observed. This work was partially supported by a gift from the Microsoft Corporation.

9. REFERENCES

[1] Altmann, E.M. and W.D. Gray. Managing attention by preparing to forget. In *Proceedings of the International Ergonomics Association and Human Factors and Ergonomics Society (IEA '00/HFES '00)* (Santa Monica, CA, July 31, 2000). 2000, 152-155.

[2] Altmann, E.M. and J.G. Trafton. Task interruption: Resumption lag and the role of cues. In *Proceedings of the 26th annual conference of the Cognitive Science Society*, 2004.

[3] Bailey, B.P., J.A. Konstan, and J.V. Carlis. The effects of interruptions on task performance, annoyance and anxiety in the user interface. In *The Proceedings of INTERACT 2001* (Tokyo, Japan, July 9, 2001). IOS Press, 2001, 593-601.

[4] Bannon, L., A. Cypher, S. Greenspan, and M.L. Monty. Evaluation and analysis of users' activity organization. In *Proceedings of the ACM conference on human factors in computing systems (CHI'83)* (Boston, MA, December 12, 1983). ACM Press, 1983, 54-57.

[5] Bardram, J.E. and T.R. Hansen. The aware architecture: Supporting context-mediated social awareness in mobile cooperation. In *Proceedings of the ACM conference on computer supported cooperative work (CSCW'04)* (Chicago, IL, November 6, 2004). ACM Press, 2004, 192-201.

[6] Beck, K., *Extreme programming explained.* Addison Wesley, Reading, MA, 2000.

[7] Begole, J., N.E. Matsakis, and J.C. Tang. Lilsys: Sensing unavailability. In *Proceedings of the ACM conference on computer supported cooperative work (CSCW'04)* (Chicago, IL, November 6, 2004). ACM Press, 2004, 511-514.

[8] Blau, P., *Exchange and power in social life*. Wiley, New York, NY, 1964.

[9] Clifford, J.D. and E.M. Altmann. Managing multiple tasks: Reducing the resumption time of the primary task. In *Proceedings of the 26th annual conference of the Cognitive Science Society*, 2004.

[10] Czerwinski, M., E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. In *Human factors in computing systems: Proceedings of CHI'04* (Vienna, Austria, April 24, 2004). ACM Press, 2004, 175-182.

[11] Dey, A.K. and G.D. Abowd. Cyberminder: A context-aware system for supporting reminders. In *Proceedings of the 2nd international symposium on handheld and ubiquitous computing (HUC '00)* (Bristol, UK, September 25, 2000). Springer, 2000, 172-186.

[12] Fogarty, J., A.J. Ko, H.H. Aung, E. Golden, K.P. Tang, and S.E. Hudson. Examining task engagement in sensor-based statistical models of human interruptibility. In *Proceedings of the ACM conference on human factors in computing systems (CHI'05)* (Portland, Oregon, April 2, 2005). ACM Press, 2005, 331-340.

[13] Gonzalez, V.M. and G. Mark. "Constant, constant, multi-tasking craziness": Managing multiple working spheres. In *Human factors in computing systems: Proceedings of CHI'04* (Vienna, Austria, April 24, 2004). ACM Press, 2004, 113-120.

[14] Heath, C. and P. Luff, *Convergent activities: Line control and passenger information on the London Underground*, in

Cognition and communication at work, Y. Engestrom and D. Middleton, Editors. 1996, Cambridge University Press, New York, 97-129.

[15] Hudson, J.M., J. Christensen, W.A. Kellogg, and T. Erickson. "I'd be overwhelmed, but it's just one more thing to do": Availability and interruption in research management. In *Human factors in computing systems: Proceedings of CHI'02* (Minneapolis, MN, April 20, 2002). ACM Press, 2002, 97-104.

[16] Hutchins, E., *Cognition in the wild*. The MIT Press, Cambridge, MA, 1995.

[17] Hutchins, E. and T. Klausen, *Distributed cognition in an airline cockpit*, in *Cognition and communication at work*, Y. Engestrom and D. Middleton, Editors. 1996, Cambridge University Press, Cambridge, 15-35.

[18] Jett, Q.R. and J.M. George, *Work interrupted: A closer look at the role of interruptions in organizational life.* Academy of Management Review, *28*, 3 (2003), 494-507.

[19] Matthews, T., M. Czerwinski, G. Robertson, and D. Tan. Clipping lists and change borders: Improving multitasking efficiency with peripheral information design. In *Proceedings of the SIGCHI conference on human factors in computing systems (CHI'06)* (Montreal, Quebec, Canada, April 22, 2006). ACM Press, 2006, 989-998.

[20] McFarlane, D., *Comparison of four primary methods for coordinating the interruption of people in human-computer interaction.* Human-Computer Interaction, *17*, 1 (2002), 63-139.

[21] O'Conaill, B. and D. Frohlich. Timespace in the workplace: Dealing with interruptions. In *Proceedings of the ACM conference on human factors in computing systems (CHI'95)* (Denver, CO, May 7, 1995). ACM Press, 1995, 262-263.

[22] Perlow, L., *The time famine: Toward a sociology of work time.* Administrative Science Quarterly, *44*, 1 (1999), 57-81.

[23] Rouncefield, M., J.A. Hughes, T. Rodden, and S. Viller, *Working with "constant interruption": CSCW and the small office.* The Information Society, *11*, 3 (1995), 173-188.

[24] Rousseau, D.M. and J.M. Parks, *The contracts of individuals and organizations*, in *Research in organizational behavior*, B.M. Staw and L.L. Cummings, Editors. 1993, JAI Press, Greenwich, CT, 1-43

[25] Speier, C., J.S. Valacich, and I. Vessey. The effects of task interruption and information presentation on individual decision making. In *Proceedings of the 18th international conference on information systems* (Atlanta, GA, December 14, 1997). 1997, 21-36.

[26] Suchman, L.A., *Plans and situated actions: The problem of human machine communication*. Cambridge University Press, New York, 1987.

[27] Whittaker, S., D. Frohlich, and O. Daly-Jones. Informal workplace communication: What is it like and how might we support it? In *Proceedings of the ACM conference on human factors in computing systems (CHI'94)* (Boston, MA, April 24, 1994). ACM Press, 1994, 131-137.