Social Behaviors on XP and non-XP teams: A Comparative Study

Jan Chong
Stanford University, Department of Management Science and Engineering
jchong@cs.stanford.edu

Abstract

This is an ethnographic study of two software development teams within the same organization, one which utilizes the Extreme Programming (XP) methodology and one which does not. This study compares the work routines and work practices of the software developers on the XP team and the non-XP team. Observed behavior suggests that certain features of the XP methodology lead to greater uniformity in work routine and work practice across individual team members. The data also suggest that the XP methodology makes awareness development and maintenance less effortful on a software development team.

1. Introduction

Nascent research on any complex phenomena is bound to be piecemeal in nature, as different facets of the phenomena are produced for scrutiny subject to scrutiny by each researcher that seeks to understand the phenomena as a whole. The state of research on the phenomena of Extreme Programming (XP) is no different; researchers have explored such varied topics as the methodology's efficacy [11, 20], the methodology's use in classroom settings [10, 14] and still others yet evaluating particular practices, and the effectiveness of particular practices, most notably pair programming [13, 15, 5].

This work contributes to the piecemeal effort by offering yet another analytical perspective, albeit one that I believe has been absent so far. This study approaches the XP methodology from a social perspective, viewing XP as a system of organizing work that is expressed as a set of beliefs and practices. The subsequent analysis compares work-related behaviors across two teams, one which has adopted XP and one which has not. I explain the differences in two types of behaviors by tracing the roots of this behavior back to the adoption (or non-adoption) of certain XP principles and values. I end by building an explanation of how team adoption of XP methodologies transforms the work of software development for a developer on that team.

2. Theoretical framework

XP structures how individuals in a group understand both their role as a team member and the larger process of software development. It gives software developers a framework by which to understand and interpret the activities of others. This is not an explicit framework, but a set of beliefs and practices that, once adopted, shape the social structure of the group. The framework shapes how the members of the group interact with each other and how they fundamentally understand the task of "software development" in which they are engaged. This structured understanding of work, in turn, manifests itself in observable behaviors.

My perspective on this interaction process draws heavily on the sociological tradition of symbolic interactionism [4]. In particular, I draw both methodological and theoretically on the works of Garfinkel [7] and Goffman [8] as a framework for my exploration of how specific social behaviors between individuals can reflect the larger beliefs and understandings that these individuals hold.

3. Research site

I observed two teams of software developers at a mid-sized startup company in Silicon Valley, California. The XP team was formed six months prior to my arrival at the site. During my time with the XP team, the team size fluctuated between 12 and 7 members. 6 of project team members were short-term contractors, 4 of which left the team during the observation period. The non-XP team was composed two overlapping product development teams within the company that merged during the observation period. The non-XP team began as two teams, one well-established product team with 5 members (2 of which were contractors) and one relatively new

product team with 5 members (1 of which was a contractor). By the end of the observation period, the two teams had merged for a final team size of 10, 2 of which were newly hired short-term contractors. Both teams therefore varied in composition between full time and temporary employees and experienced fluctuations in team membership during the observation period.

The XP team adopted all XP practices with the exception of customer acceptance tests. The team was radically co-located at a single site. Team members paired whenever possible, working in a large, bullpen-like room around two-monitor stations. Communication between team members almost exclusively occurred through face-to-face communication.

A previous incarnation of the non-XP team had briefly adopted Scrum, but the practice, with the exception of the daily status meeting, was largely abandoned. Instead, the non-XP team adopted a relaxed version of the waterfall process, separating design, implementation and testing. The non-XP team was also geographically distributed. Although the bulk of the team spent most of their time on site, 2 members of the team worked remotely all of the time and the remaining 8 team members regularly worked from home. When team members were on-site, they worked in their own cubicles. The non-XP team utilized a company-wide computer-mediated communication (CMC) tool (notably the XP team did not use this tool, although its use was prevalent among the rest of the company) that can be roughly characterized as a large chat room. They also maintained contact through e-mail and face-to-face conversations.

I watched both teams release a version of their product to customers and engage in planning activities for the next release. Consequently, the observations of both teams spanned a "crunch" period of intense pressure to make release deadlines and a more leisurely planning period, reducing the chance that the observed temporal routines were exclusive to a particular project phase.

4. Methodology

I conducted ethnographic observations of the developers at work, either once or twice a week. Across both teams, I spent a total of 18 weeks on site starting in June of 2004 and ending in December. During observations of the non-XP team, I tracked the activities of a single programmer during the entire observation period. In the process, I also observed any other team members that came in to contact with the programmer. For the XP team, I followed roughly the same methodology, by tracking the activities of a pair through a session. Due to the close physical proximity and high frequency of interactions between team members, however, my observation data frequently spanned the activities of multiple pairs.

I took extensive notes during my observations and, whenever possible, made audio recordings of the entire session. The audio record was then transcribed and integrated into my notes to produce a faithful record of dialogue annotated by detailed descriptions of action.

To analyze my data, I employed inductive qualitative techniques [19]. I began with multiple readings of our field notes and review of the events that occurred. I wanted to understand the temporal structure of a typical developer's day. In my analysis, I tried to create a portrait of what developers attended to in the course of the workday, looking at both the events that drew their attention away from their primary work tasks and the events that they choose to turn their attention to. I created summary charts breaking down the sequence of events, noting attentional shifts. My analysis drew on techniques used previously by Pentland [16] in his exploration of organizational routines and Perlow [17] in her exploration of work patterns. To explore the use of awareness maintenance mechanisms, I identified a primary task for the informant or pair observed in the observation session (frequently, this was self-identified by the informant or pair in the course of the observation). I then coded the data for all instances where a developer's attention, either voluntarily or prompted by some external event, deviated from his or her primary task. In doing so, I also identified in the data every instance of a social interaction between team members.

5. Findings

In the following section, I present behavioral differences observed between developers on the XP team and developers on the non-XP team. I focus on two particular categories of behavior: temporal patterns of work and awareness maintenance mechanisms.

5.1 Work patterns on the XP team

Analysis of work routines across team members of the XP team revealed a startling uniformity in the temporal structure of the work sessions across team members. Below, I present a sample sequence of events[1] for two XP team members. Following the actual steps used in analysis, each event is annotated by line number. In this case, the activities listed follow a pair, Andy and Brian[2] in a morning work session.

0110-0122: Andy is pulled into a discussion of a technical issue with Lee, the customer. Brian leaves to get coffee.
0122: Andy begins working alone.
0133: Erin asks the team a question. Andy answers.
0160-0173: Carl calls over Andy to consult on a customer issue.
0173: Andy returns to his computer. Brian returns from coffee and joins him. They begin to pair.
0408: Carl, overhearing some of their conversation, makes a joke.
0413: Lee returns, calling Andy away from the area to review an issue.
0425: Brian notices that a test is broken and asks the team if this is meant to be
0443-0473: Andy returns to the area to speak with Erin about the customer issue.
0473-0504: Andy and Brian pair.
0504-0609: Andy and Brian ask Carl to review the bug they have encountered. The three of them work together on the issue.
0609: Andy returns to his computer, while Brian continues to work with Carl.
0628-0639: Brian and Carl ask the team a question that leads to a brief team-wide discussion.
0689-0692: Lee comes in and announces that a feature is broken (the issue that Brian and Carl are currently working on). Brian and Carl update the customer.
0703: Brian returns to working with Andy.
0710-0726: Dennis asks Brian for help.
0726-0758: Andy and Brian return to pairing.
0758-0794: Erin asks Andy for help
0794: Brian and Andy return to pairing.
0866-0880: Carl initiates a group wide discussion about lunch
0895: Brian goes to lunch, while Andy continues to work on the issue.

Andy and Brian's work session is fairly representative in terms of the underlying structure of the workday for an XP team member. In general, the work routines and work hours across the XP team were remarkably similar. Team members generally arrived at the office slightly before the daily standup meeting. Time before the stand up meeting was unstructured, spent checking e-mail, web-surfing or catching up on work tasks from the previous day. The standup meeting was short, running between ten and fifteen minutes. Pairs for the day were arranged at the end of the standup meeting. Immediately after the meeting, the team scattered to retrieve coffee from the company break room. Teams members promptly returned to begin pair programming, which they continued until lunchtime. After lunch, team members resumed pairing until the end of the workday.

As illustrated by Andy and Brian's session, pair programming sessions were frequently interrupted by events that directed one or both programmer's attention away from their primary task, such as inquiries for technical information or support from other team members, social exchanges of conversations and coffee/snack breaks. In my observations, pairs never completed a two or three hour stretch of pair programming activity without some sort of interruption. The general structure of pair programming sessions were strikingly consistent across the team. The pair would work together until interrupted, at which point one member would leave to attend to the interruption. The other member would continue to work on the pair's task. When the first member returned from the interruption, the second member would update him or her on the status of the task and they would return to pairing.

5.2 Work patterns on the non-XP team

While the XP workday had a consistent structure across team members, the non-XP work varied greatly from individual-to-individual and day-by-day. Work was much more diffuse, occurring at a greater variety of times and locations. The non-XP team members frequently worked from home, logging into the CMC tool to check on team status and sometimes code or answer technical inquiries before leaving the house to come to the office. Team members maintained communication on the CMC tool until late at night and sometimes kept irregular work hours. Like the XP team, the non-XP team also had a daily status meeting, but team members could and frequently did attend remotely. Consequently, individual team members had very different work routines. To illustrate this point, I present a sample work session from two different non-XP developers, both working at the office. The first sequence tracks a developer named Fred through a

[1] This and subsequent excerpts have been edited for space considerations.
[2] All of the names used in this paper are pseudonyms.

morning work session. The second sequence tracks a developer named Ian through an afternoon work session.

0371-0399: Fred responds to a technical inquiry sent to him by e-mail.
0399: Fred turns to his primary task, writing a new set of routines.
0750-0789: Fred participates in a discussion of where to go to lunch over the CMC tool.
0789-0798: Fred checks his e-mail.
0798: Fred turns back to coding.
0839-0849: Fred reads and then briefly responds to a discussion on the CMC tool.
0849: Fred continues to code
0855-0865: Fred makes a joke over the CMC tool, in response an ongoing discussion.
0865: Fred continues to work.
0885: Greg and Hilary come to Fred's cubicle to fetch him for lunch.

In this session, Fred maintains a fairly focused pattern of work. There are fewer external interruptions, presenting the opportunity for longer stretches of concentrated work. Fred, however, turns his attention away from his primary task at several points during the session to monitor communication from the rest of this team. In this particular session, he does so twice, once via e-mail and one three times via the CMC tool. After each check, he promptly turns his attention back to his work. Ian's work session, on the other hand, has a very different structure:

0097-0111: Upon returning from lunch, Ian sits down to carefully review the log of discussions over the CMC tool.
0142: Ian responds to an inquiry from John, who is working remotely, over the CMC tool.
0143: Ian begins working on his primary task for the day, which is determining and then documenting how to set up a particular tool.
0168: Ian pauses to flip through a stack of mail that has been delivered to his cubicle.
0169: Ian turns back to his primary task.
0256-0380: Ian stops to review the discussion log of the CMC tool. He spends some time following a set of humorous websites posted by others to the CMC discussion.
0380: Ian turns back to work.
0812: Ian asks Kai, who is walking by Ian's cubicle, for help.
0838: Kai leaves and Ian continues to work.
1065: Ian chimes in, to make a joke, to a loud conversation that Greg and Hilary are having in Greg's cubicle.
1072: Ian turns back to work.
1108: Ian notices that his emacs settings on his machine are not working. He spends some time fixing them.
1195: Ian turns back to his primary task.
1287-1297: Ian takes a snack break.
1297: Ian posts a question to the CMC discussion about a problem he's been having with a tool running on his home machine.
1324: Ian downloads a Star Trek video that has been posted to the CMC tool and watches it. The noise draws Greg, Fred and Hilary over to watch and chat about the video.
1390: Ian reviews the CMC tool discussion log.
1402: Ian downloads and watches another video posted to the CMC log.
1411: Ian turns back to work.

In this session, Ian has few external interruptions, which gives him the opportunity to devote long stretches of uninterrupted attention to his primary work task. Instead, however, we find that Ian's sessions is filled with points at which he turns his attention away from his primary work task without any external impetus. Ian's session alternates between engagement in periods of focused work and sustained periods of social conversation or other activities unrelated to work.

Ian's and Fred's sessions are, of course, selected for maximum contrast, but in comparison with the XP team, the non-XP team members had a marked lack of consistency when it came to their patterns of work.

5.3 Awareness on the XP team

Awareness has long been established as a critical component of collaborative work [6, 9, 18]. Dourish and Bly [6] define awareness to be "an understanding of the activities of others, which provides a context for your own activity." Maintaining awareness is the process by which individuals working with others transmit and acquire information, consciously or unconsciously, about their work efforts and how that effort fits in with the on-going work of others. Teams of people engaged in highly collaborative work often develop mechanisms to help other team members maintain awareness by making certain features of their work more salient than others, for instance, through gestures and speech [9].

In a rich work environment, awareness rarely has a single source; an XP team is no different. Developers become aware of the work of others on the team through a variety of physical cues, ranging from physical artifacts in the workspace

to observable actions or audible utterances made by teammates.

One obvious way that awareness is developed and maintained in an XP environment is through the stream of dialogue between developers during pair programming. In the following excerpt, John and Kai are attempting to debug some newly written code, after watching it fail the test:

John: There's the test. It apparently fails.

John and Kai examine the code for a bit, silently.

Kai: We didn't do anything on end.
John: Oh, we have an infinite loop.
Kai: On end you didn't increment.
John: Oh yeah, that.
Kai: Oh no, you did look for end. Okay. Might be sufficient.
John: Why do we have an array out of bounds exception? Nice. Oh. Nothing matches? Well let's go to the line…

They add a breakpoint and run it in the debugger to see what is being generated by the code at that point.

John: Hrm. Do you see the difference? We have some extra stuff at the end? I think the actual regexp still might be wrong because of the \n.

In this incident, John and Kai use speech to direct each other's attention to various aspects of the situation. John begins by observing that the test has failed, focusing both of their attention on that issue. Kai then responds by drawing attention to a particular segment of code where he thinks the problem might be – the code that deals with the "END" tag. Their speech throughout the except acts as a coordinating mechanism, communicating what aspects of the code and the bug each member of the pair programming duo finds relevant to their work.

The observation that pair programmers use dialogue to coordinate their efforts is by no means earth shattering, particularly in light of the fact that their speech is the main means by which they share information about their work task. But the dialogue produced in these pairs also serves to make information about the pair's work available to other team members who are physically situated around them. Exchanges, such as the following, were fairly commonplace:

Andy: Dennis, are you a CVS expert? This icon gets corrupted every ten seconds?

Matt: [to Andy] I know how to fix it now. You have to delete it from the store and enter it and the first time that you enter it, enter it as binary the first time.
Andy: We've done that, several times.
Matt: [shrugs] I've created several of them and every time I've followed that rule, it's worked.
Dennis: But you have to do it on create, you can't change the type, right?
Matt: That's why you have to delete it.
Dennis: Okay.

Here, Dennis and Andy are paired and, in the course of working together, Andy asks Dennis a question. Although Matt is in the process of working with Erin, he nevertheless turns to answer Andy's question, demonstrating that he is not only aware of their conversation, but actively listening to it. Of course, not all team members engaged in this behavior to the same level. Limitations on human attention dictate that the amount of attention that team members to devote to cultivating awareness varied by day and task. Nevertheless, even in the absence of a conscious effort to develop an awareness about the work going on around them, a team member immersed in these surroundings was always, to some extent, building this awareness.

Similarly, transmission of awareness information is a relatively effortless act in the XP environment. For instance, in the following incident, Matt and Erin grew increasingly frustrated as they wrestled with a bewildering bug. Although they only ask Brian for help with the bug, others on the team become aware of the issue as it unfolds. This leads Carl to eventually step in and offer unsolicited help, when Matt and Erin grow too frustrated to continue working. Here, when they first notice the error, Matt is actually speaking to Andy, on the other side of the room, while Erin gives a demo to Lee. When the error occurs, Matt trails off mid-sentence and walks back over to Erin and Lee:

Matt: But the beauty is that you have the exact copy...

Matt trails off as he notices the error on Erin's screen. He walks back over to Lee and Erin.

Matt: [joking] If it doesn't work, I didn't do it. I was working on ACLs.

On the screen, the application page displays an exception.

Andy: [from his computer] That looks like an error.
Erin: Huh.

Shortly afterwards, Matt and Erin begin to trace the issue in earnest. They soon call Brian over for help, because they believe it may be related to some of the code he recently altered. In this exchange, Carl, who is working separately, overhears this, looks up and begins to ask about the issue. Initially, however, Matt and Erin reveal little information in response to his questions and the conversation ends quickly:

Erin: Brian, have you done the lazy collection things?
Brian: [distracted with own work] Yes.
Erin: Because we have an exception
Carl: [looking up] Do you have a test fail? or just something?
Matt: Or just something.
Carl: Because all the tests pass. Maybe we need a test if this isn't getting caught.
Matt: We don't know what we're testing for.
Brian: I'll be there, let me just finish this.

As more time passes, Carl checks in once again. Although Matt, Erin and Brian respond to Carl's statements, Carl is not included in the subsequent conversation and Carl will return to his work on other code:

Brian: But now the store is closed too early for this context. And it might be good that we've exposed this.
Erin: Because it may always be closed too early.
Matt: But I thought the object rendered from the store.
Carl: No one should be closing the store?
Matt: [joking] We're open 24 hours.
Brian: What about uncommitted objects? Do we throw away the hibernated session when we commit? If we do, objects retrieved from that session are now stale, need to be refreshed.
Carl: Yes, but closing the store should be the last thing that we do before the action.
Erin: [to Brian] When we commit we throw away the session.

As time passes, Matt and Erin grow increasingly frustrated. At the time of the next exchange, Erin's responses to Brian grow less and less substantive. Carl comes over, unsolicited by the others, to step in and help:

Erin: There's an error.
Brian: That here? [Brian points to the screen.] There's no error. It's just not displaying the objects that exist.
Erin: If you try to load a container that doesn't exist?
Brian: How?
Erin: [defeated] I don't know
Brian: By ID?
Erin: [frustrated and defensive] I don't know, I'm just saying.

Carl comes over.

Carl: So what doesn't work?
Erin: [gloomily] Everything that worked this morning.
Carl: [Carl moves to stands closely behind Brian] Let's go back up, I'm trying to see where this stuff gets called from. [He instructs Brian to scroll up on the screen.] Execute. If you open up the web server…

In the above incident, the means by which Carl becomes aware of the situation as it develops was largely passive, a mere fact of his presence in a shared workspace. Once he became aware that there was a situation, he began to also actively solicit information. Nevertheless, these incidents illustrate a hallmark of awareness development on the XP-team, namely that team members can be largely passive, but still transmit significant information about their work as well as acquire significant information about the work going on around them.

5.4 Awareness on the non-XP team

Members of the non-XP team lacked both the consistent physical proximity with other team members and access to a dialogue similar to that produced by the pair programmers; consequently they used very different mechanisms to maintain awareness. Notably, members of the non-XP team seemed to require more concerted attention and effort to develop and maintain the awareness necessary for their work.

Strategies for cultivating awareness differed across the team members. Some team members actively broadcasted status information to team members. The following exchange, which occurs some time after Nate asked Olivia to examine the cause of a build failure, is fairly typical:

The browser loads and Nate reviews the options on the webpage, but a beep announces a private message on [the CMS tool], so he checks the [the CMC tool] window. Olivia has sent him a message, "RDS [the name of their product] should compile on AIX now."

Nate: Apparently RDS should compile now. Let's see.

The developers regularly broadcast updates while working through the team-wide channels on their CMC tool, private messages on the CMC tool, e-mail or via face-to-face conversation. Not all broadcasting behaviors were so explicit. Awareness information was also available through more subtle channels, such as casual face-to-face conversations or technical discussions conducted on the team-

wide channels of the CMC tool. All the non-XP team members were careful to track the technical discussions that occurred over CMC, although they often filtered out the social discussions.

Broadcasting behaviors reduced the effort required on the part of other team members to maintain awareness, although they often required concerted effort on the part of the broadcaster. For developers that did not broadcast, the remaining team members often simply lacked a basic awareness of their activities, something that they actively complained about. During the observation period, the team employed a contractor named Peter, who transmitted exceptionally little information about his work, both by being rarely physically present in the office and by remaining relatively silent on the communication channels used by the team. The team discussed Peter over lunch one day:

Nate: [to me] She [Olivia] suggested that you shadow Peter, so you can tell us what he does! [They all laugh heartily]
Hilary: Yeah, I don't know what he does.
Nate: Sometimes I'll see him suddenly in his cube when I'm getting ready to leave or headed in that direction to talk to someone and it's like, "Oh Peter's here."
Hilary: How many days a week does he work from home?
Nate: I don't know. I wish he would send out schedules.
Hilary: He did for a little while, but I don't think he likes them.
Nate: I don't even know what he's doing. [To me] So you have to understand, Peter's been working on the same project for six months-
Hilary: Has it been that long?
Nate: And he gives status updates in the broadest of strokes. Like for the first two months, it was just "working on CM!" and that was it.
Hilary: Or, "Doesn't work! Looking at it!"
Nate: He seems like a nice enough guy. I mean, don't get me wrong, I'd like to be able to work with him because I'm sure he's nice enough, but… [Nate trails off]

In this discussion, we see that the team is frustrated by the lack of information that they have about Peter – to the extent that they feel that they are unable to work with Peter because they lack awareness of his work activities. Lack of broadcasting often meant that team members needed to actively solicit awareness information. In a representative incident, Ryan, another member of the team, checked in without some of the changes he had discussed with the team on the day before. He also failed to notify the other team members that the addition of this functionality would be delayed. In the course of his work session following the check in, three other members of the team separately contacted him to inquire about the status of the check in. This inquiry from Fred, is typical:

Ryan glances at the CMC tool. He's received a message from Fred:

I do not see any of the rrtd based client fields in the cont_logic.c module and the cont.schema still has a lot of gunk from the tld.schema in it. Are you updating these now? It can not be released as it is.

Since Ryan is not making information about his work passively accessible to others, Fred must actively solicit this information. While not all awareness information on the non-XP team required active, conscious solicitation or transmission, in general, awareness development and maintenance on the non-XP team was a more effortful activity than on the XP team.

6. Discussion

In the previous section, we saw a portrait of social behavior exhibited by members of an XP team and members of a non-XP team. In this section, I explore the link between the adoption of the XP methodology and these social behaviors.

6.1 XP as organizing

XP is a framework of beliefs and practices that transform software development into a standard and transparent practice. These two features, standardization and transparency, play a crucial role in shaping the structure of an XP team and, subsequently, in shaping the ways in which members of an XP team go about the work of software development.

6.1.1 Standardization. When I claim that XP has standardized software development, I want to first make a clear distinction between software development and programming. XP has not standardized programming, in the sense of removing from it the knowledge and skill required to do the work of programming. Standardization does not imply a dumbing down of the fundamental task of programming here.

What XP has, instead, done is carefully delineated the role of the software developer in the software development process. While work

movements in the past (ex. TQM [1] and self-organizing teams [2]) have sought to broaden the role of the worker, in some sense, the power of XP comes from limiting the software developer role. XP organizes the software developer's role, in part by removing certain responsibilities from it and by giving a uniform structure to the tasks that remain.

For software developer at work on an XP team, there is a shared, reliable understanding of what one's work task is, the scope of one's work task, how the project's work tasks is allotted among fellow team members, how one should work on one's work task and what it means to complete a work task. In XP team observed here, for instance, team members understood their task to be defined by the words written on a set of colorful index cards that they selected off of a board or volunteered to take on during the morning standup meeting. They understood that their work involved cooperatively creating structures in code that exhibited interactions that matched the descriptions written on the index cards they selected, as well as helping others on their team do the same. The team members understood that working on their task meant being present in the same room as their teammates whenever possible and working, when possible, in pairs. They also understood that task completion meant demonstrating code behavior to the customer and having the customer believe that their task criterion was satisfied.

The clear definition of these understandings is the interpretive structure that the XP framework brings. This is not to say that every XP team will or should adopt the same understanding for each of these concepts. Each team must develop its own set of understandings, although team convergence is no doubt encouraged through the prescription and emphasis of particular practices (such as Open Workspace and Pair Programming) within the XP repertoire by the larger XP movement.

In the XP team observed in this study, these shared understandings manifested themselves in the consistent, uniform patterns of work in which the team members engaged. Their understanding of their work role under the XP framework required that they work at the same time, in the same place, and in largely the same ways. The non-XP team did not share this understanding of software development work and therefore exhibited a wildly different structure of work.

6.1.2 Transparency. The concept of transparency, as I use it here, comes from Lave and Wenger's influential work on situated learning [12]. Transparency to Lave and Wenger is a process rather than an attribute, one that is a fundamental to the practice of learning. Applied to the work of software development, transparency is the process by which the work because accessible, observable and knowable to others.

In one sense, this transparency is unsurprising; after all, XP sprang up in part due to a sense that available software development processes were too opaque. That a customer or a manager could be so unaware of a project status that schedule slips came as a surprise was viewed as a failing that XP might remedy [3]. But what I mean by transparency goes beyond creating and maintaining an awareness of project status in terms either features complete out of features desired or working pace. XP makes developing software visually and aurally available. On a very basic level, a software developer on an XP team can now regularly see and hear others on their team, chiefly due to their increased physical proximity and the continuous talk produced by pair programming. Pair programming, by its very nature, increases the accessibility (and hence transparency) of the programming component of software development; the practice creates a verbal, aural interface to cognitive act of constructing code.

These new forms of accessibility not only allows XP team members to easily make assessments of whether team members and activities are "on task", but it allows them, throughout the course of the workday, to build and maintain a continuing awareness of the work that is going on around them. For the non-XP team members, software development work is not nearly so transparent and consequently they must work consciously to acquire the same information.

7. Conclusions

While past research has emphasized the efficacy of pair programming, overlooked has been the social effects of the XP methodology and their role in the methodology's effectiveness. This study attempts to remedy, at least in part, this oversight.

The study's findings suggest that XP provides a framework for standardizing the work of software development and making this work more effortlessly visible and accessible to

members of a software development team. The transformations have implications for the social relations of team members, both in terms of behaviors enacted and understandings developed in the course of their work. It should be noted, of course, that the social behaviors reported in this study are situated, local practices that have been developed over time by the team members in response to both the general effects of the XP framework (or lack of it) and a specific set of arbitrary environmental circumstances. Therefore, while the roots of this behavior may be constant across teams that adopt XP (or teams that do not) and, indeed, many of the practices observed here may be also shared, the expectation that these behaviors be replicated perfectly across all teams that adopt XP is unrealistic. A major challenge of social science research has always been to separate the systemic causes and effects from the local ones; it is no different here.

Standardization and transparency both have implications beyond the behaviors explored in this paper. The transformation of software development into a more routine, recognizable and accessible form of work should result, for instance, in an equally fundamental transformation in the process of learning. A fruitful avenue for further study would be examination and comparison of these learning behaviors across XP and non-XP teams.

8. Limitations

While the conclusions reached in this work are a result of a systematic and careful exploration of a rich, descriptive data set, ethnographic work always faces the question of generalizability due to sample size. Replication of this work with other teams and other environments could stand to strengthen the findings.

In addition, this study closely examines the effects of a limited number of practices in the XP framework, while simultaneously attempting to consider the effects of the framework as a whole. Since the study's emphasis is on social interaction, I have implicitly focused on the practices and values most likely to shape social interaction: the value of Communication and the practices of the Planning Game, Open Workspace and Pair Programming. Software development, however, has a fundamental technical interdependence that has not been thoroughly explored here. Further work that looks at this technical interdependence and the effects of XP practices and beliefs on how teams structure and are structured by this technical interdependence may stand to add to our understanding of how XP "works".

9. Acknowledgements

This work would not have been possible without the support and participation of all the informants who so thoughtfully made their time and attention available to me. In addition, Vivian Neou, Rob Mee and Chris Sepulveda provided invaluable assistance. For their patience, comments and feedback, I'd like to thank Diane Bailey, Chris Gates and Somik Raha.

11. References

[1] Adler, P., B. Goldoftas and D. Levine, "Ergonomics, Employee Involvement, and the Toyota Production System: A Case Study of NUMMI's 1993 Model Introduction," *Industrial and Labor Relations Review*¸1997, p. 416-437.

[2] Banker, R., J. Field, R. Schroeder and K. Sinha, "Impact of Work Teams on Manufacturing Performance: A Longitudinal Field Study," *The Academy of Management Journal*¸1996, p. 867-890

[3] Beck, K, *Extreme Programming Explained*, Addison Wesley, 2000.

[4] Blumer, H., *Symbolic Interactionism: Perspective and Method*, Prentice Hall, Englewood Cliffs, NJ, 1969.

[5] Cockburn, A. and L. Williams, "The Cost and Benefits of Pair Programming" in *Extreme Programming Examined.* Addison Wesley, 2001.

[6] Dourish, P., and V. Bellotti, "Awareness and Coordination in Shared Workspace," *Computer Supported Cooperative Work 1992 Proceedings*, ACM, 1992.

[7] Garfinkel, H. *Studies in Ethnomethodology*, Prentice-Hall, Englewood Cliffs, NJ, 1967.

[8] Goffman, E. *The Presentation of Self in Everyday Life*¸ The Overlook Press, Woodstock, NY. 1959.

[9] Heath, C., M. Svensson, J. Hindmarsh and P. Luff, "Configuring Awareness," *Computer Supported Cooperative Work*¸ Kluwer Academic Publishers, 2002.

[10] Keefe, K. and M. Dick., "Using Extreme Programming in a Capstone Project", *Proceedings of*

the 6th Conference on Australian Computing Education, ACM, 2004. p. 151-160.

[11] Layman, L., L. Williams, and L. Cunningham, "Exploring Extreme Programming in Context: An Industrial Case Study", Agile Development Conference 2004, p. 32-41.

[12] Lave, J. and E. Wenger. *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press, 1991.

[13] McDowell, C., L. Werner, H. Bullock and J. Fernald, "The Effects of Pair-Programming on Performance in an Introductory Programming Course", *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, ACM, 2002.

[14] Muller, M. M. and W. F. Tichey. "Case Study: Extreme Programming in a University Environment", *23rd International Conference on Software Engineering*, IEEE, 2001.

[15] Nicolescu, R and R. Plummer. "A Pair Programming Experiment in a Large Computing Course", CITR Technical Report, 2003.

[16] Pentland, B. T, "Grammatical Models of Organizational Processes", *Organization Science*, INFORMS, 1995, p.541-556

[17] Perlow, L., "The Time Famine: Toward a Sociology of Work Time", *Administrative Science Quarterly*, Cornell Business, 1999, p. 57-81.

[18] Schmidt, K. "The Problem with 'Awareness'", *Computer Supported Cooperative Work*¸ Kluwer Academic Publishers, 2002, p. 285-298.

[19] Strauss, A., & Corbin, J., *Basics of qualitative research: Grounded theory, procedures, and techniques*, Sage, Newbury Park, 1990

[20] Williams, L., Krebs, W., Layman, L., and Antón, A., "Toward a Framework for Evaluating Extreme Programming", *Empirical Assessment in Software Engineering (EASE)* 2004, pp. 11-2