The "Pair" as a Problematic Unit of Analysis for Pair Programming

David Socha, Kevin Sutanto
Computing and Software Systems
University of Washington Bothell
Bothell, USA
socha@uw.edu, kevinsnr@uw.edu

***Abstract*—This paper explores the problematic nature of using an isolated pair as the unit of analysis in studies and evaluations of pair programming. Using empirical data from an observational case study within a software development organization, we show pairs spending 20% of their pairing time interacting with people outside the pair. These interactions, which are encouraged by this organization as part of its highly collaborative system, represent important value exchanges with people outside the pair. This suggests that research on pairs in isolation may not be indicative of how pair programming works in situ when enacted by teams accomplished in the practice, and may misrepresent the net value proposition of pair programming.**

***Index Terms*—Pair programming, software design, case study**

I. Introduction

How do we represent the work of software development to ourselves and others, and how might these representations influence how we study and evaluate such work? For instance, there is abundant research on pair programming, yet the research is not conclusive on whether pair programming is an effective practice. This paper suggests that one reason may be that the representation of pair programming, the conception of what is being studied, may itself be problematic, not in the definition itself, but in what it makes visible and what it glosses. In *Making Work Visible* [1], Suchman states that people who *do* particular forms of work have a special relationship not only to this work but to its *representation*, and that such representations always have political and social ramifications both inside and outside the settings in which the work is undertaken. How do researchers and practitioners represent pair programming, and how might those representations influence how we investigate the nature of pair programming and its value proposition? How does it influence the scope of our inquiry, the definition of what is "in" the scope of study, and what is "out"?

We explore these questions in the context of ethnographic data from an empirical study of software development practices that we have been investigating for the last two years, using analyses of pair programmers to highlight the mismatch between how work is sometimes practiced and how it is idealized and normalized in the literature. We argue that normalizing particular representations of practice may be problematic for research, education, and practice.

These results have implications for the methods by which we study pairing, for how we measure the value of pairing, for how organizations configure their workspaces, and for how we educate students about pairing.

In the next section, we summarize some of the literature on pair programming, highlighting its normative character. We then describe our data collection and analysis. We discuss potential ramifications, and conclude by reiterating our argument. In the balance of this paper we use the term "developer" as a synonym for "software developer".

II. Background

Much of the interest in pair programming can be dated to Kent Beck's formulation as one of the practices in Extreme Programming. Beck defines the key characteristic of pair programming as "[a]ll production code is written with two people looking at one machine, with one keyboard and one mouse" [2]. In a single sentence, this both widens the unit of production from a single programmer to a pair, and isolates this pair from other members of a software development team and the larger organization. Perhaps as a result, a considerable amount (though not all) of practitioner and academic research has studied pair programming solely in terms of inter-pair interactions, asking such questions as what roles the members take on in relation to one another [3], [4], what technological tools can support distributed pair programmers [5]–[7], the quality of code produced from pair programmers [8], and similar. That is, the unit of analysis is taken as identical with the unit of production: the *pair itself.* This is so taken for granted that it is never remarked upon, let alone challenged. For it might seem somewhat absurd to consider another unit of analysis for investigating *pair* programming.

Furthermore, this same focus on the pair as an isolated entity carries over to normative prescriptions about using pair programming for educational and training purposes. For example, Williams and Kesler begin their book *Pair Programming Illuminated* [9] ("written … for software development team members and their managers … [and] for educators who would like to try pair programming with their students") with: "At face value, pair programming is a very simple concept. Two programmers work together at one computer on the same task. Done." This is the alpha and the omega of pair programming.

Accepted for publication at CHASE 2015

But is it? Is the pair the most important unit of analysis that we can use, especially for pair programming in organizational settings? In Kent Beck's book on Extreme Programming, he stresses the importance of the team, of what Teasley et al. term "radical collocation" [10]: "*team members need to be able to see each other, to hear shouted one-off questions, to 'accidentally' hear conversations to which they have vital contributions.*" [2, p. 79]. Later in that book, Beck states that "*If you absolutely can't move the desks, or the noise level prevents conversation, or you can't be close enough for serendipitous communication, you won't be able to execute XP at anything like its full potential*" [2, p. 158]. Beck conceived of pair programming not as an isolated practice, but as one of a set of practices (the XP practices) supported by organizational structures such as radical collocation.

There is much less research on the interactions between members of a pair and other people in the organization, what we term as *extra-pair interactions*. Such research often uses terms such as interruptions [11] or disengagement [12] to frame such extra-pair interactions, representing these interactions as disruptions to the normal course of events, e.g., noting that workers "must contend with all manner of disruptions" [11, p. 29]. The implication is that such interactions are likely to be problematic to the pair's effectiveness, though Plonka notes that disengagement is not always an undesirable behavior [12].

In the next section, we undertake an analysis of pair programming where we widen the field of vision to include not only the pair, but other software developers within the same organization, all of whom are working jointly to produce the same software product. In doing so, we highlight the importance of extra-pair interactions, and thus call into question the normative representations of pair programming as simply a matter of "*two programmers [who] work together at one computer.*"

III. Method

1.1 Data collection

The data for this study come from a research project using video ethnography to explore *how* professional software developers collaborate on their *actual* work in their workplace [13], [14]. We collect and analyze videos and related data of software developers collaborating in their organization, such as when they are pair programming.

The data for this paper was collected at a 9 year-old software development company in the Seattle area of the United States. The company has about 50 employees and is owned by a non-US parent company. The organization's product is a software system that helps friends and family share information. It has over 13 million users, includes a significant backend Software-as-a-Service (SaaS) component, and has both a web-based version and client versions for Macintosh, Windows, iPhone, and iPad.

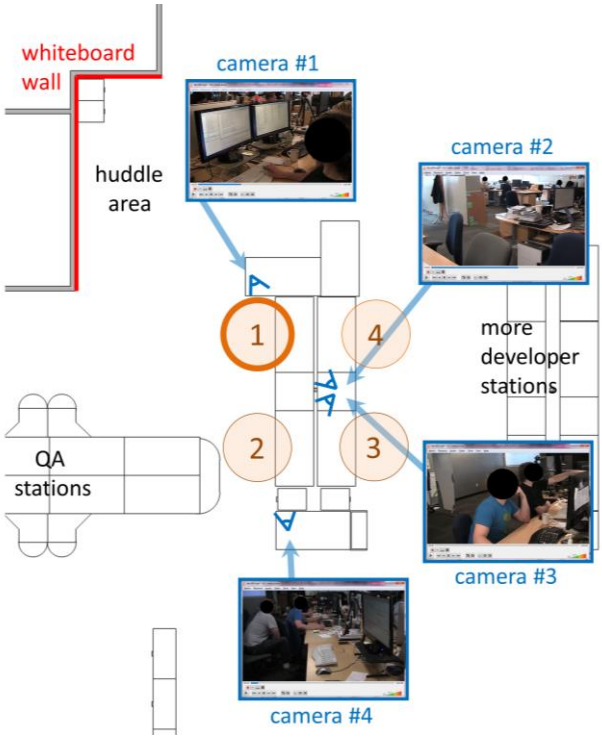

Fig. 1. Locations of four pairing stations (circles 1-4) located near the huddle (standup meeting) area and whiteboard wall (red lines). This paper focuses on pairing station 1 recorded by four video cameras (A-like shapes).

All employees work in a single large room with no dividers. The software developers in the organization use a mix of extreme programming [2] and Scrum [15], as well as other practices they have evolved. Fig. 1 shows a portion of the floor plan in the organization centered on the developer. That developer area has four "pairing stations" (circles 1-4), each with two keyboards and two mice controlling a single cursor and keyboard input on a single computer. When pairing, developers sit side-by-side. During our initial data collection, we configured fixed cameras at four different locations within the pairing stations, each configured to record for an entire day.

In total, we have collected approximately 400 hours of video data of software development work in this organization. The data include 24 hours of ethnographic observations done by the first author over the course of seven visits from October 2012 through January 2013, augmented by field notes and photographs. It also includes interviews of the VP of Engineering and one developer.

This paper focuses on the interactions at a single pair programming station (circle 1 in Fig. 1) over a 6:43 hour period on January 24, 2013. The activity at that station on that day was characteristic of the pairing activity we observed in this organization. The 6:43 hour period is long enough to show the nature of the types and variety of interactions we observed in this organization.

A. Data Analysis

This paper focuses on the extra-pair interactions between either of the two developers assigned to station 1 (see Fig. 1)

and any of the other developers in the organization. We used interaction analysis [16] to analyze the videos in order to discover and quantify things like how artifacts, gestures, and voice mediate the process of software developers forming what Clark calls *common ground*, i.e. the mutual knowledge that people have about the setting, the task, and one another's state of knowledge [17]. During our collaborative and independent analyses, we dynamically adapted the level of analysis based upon the particulars in the video in order to focus on the most interesting interactions.

We considered four different units of analysis: the individual developer, the pair, the station occupied by a pair for a pairing session, and the task being worked on by the people at the station. We chose the pairing station as our primary unit of analysis because our initial analysis indicated that this unit provides the clearest representation of the totality of interactions by the software developers as a whole. That is, focusing exclusively on the pair would miss a large number of interactions between software developers, many of which were essential to a pair in completing their work. In addition, this unit of analysis (the pairing station) represents the location to which a particular task or topic is assigned within this organization, even as developer pairings change and move. Using the station as the primary unit of analysis also did not make any a priori assumptions about the number of people working on a task, or whether a pair might swap individuals partway through a pairing session as described for some organizations [18].

We measured both the intra-pair interactions, where the members of a pair interact only with one another, as well as the extra-pair interactions. We take an *extra-pair interaction* to be the period of time when one of the developers at a pairing station is visibly or audibly communicating with, or engaged in the actions of, a person *other than their partner at the pairing station*. In almost every case, the audio and visual clues clearly indicated when an interaction started and ended. In this data, every interaction started with a verbal exchange between a developer at station 1 and another person. The people involved in an interaction almost always physically changed their body orientation or position to clearly be engaged in the joint conversation or activity. Simply looking at another developer was *not* used to indicate the start of interaction, though it might indicate a previously started interaction was not yet complete. We did not try to reason about the invisible activity happening inside the participants' heads. Instead, we focused on the "actual observable conduct" [19, p. 21] that participants make publicly available and use in structuring their joint work. These judgments, and our interpretation of actions and utterances, were informed by the first author's expertise in software development gained from 19 years as a full-time software developer, manager, and agile coach.

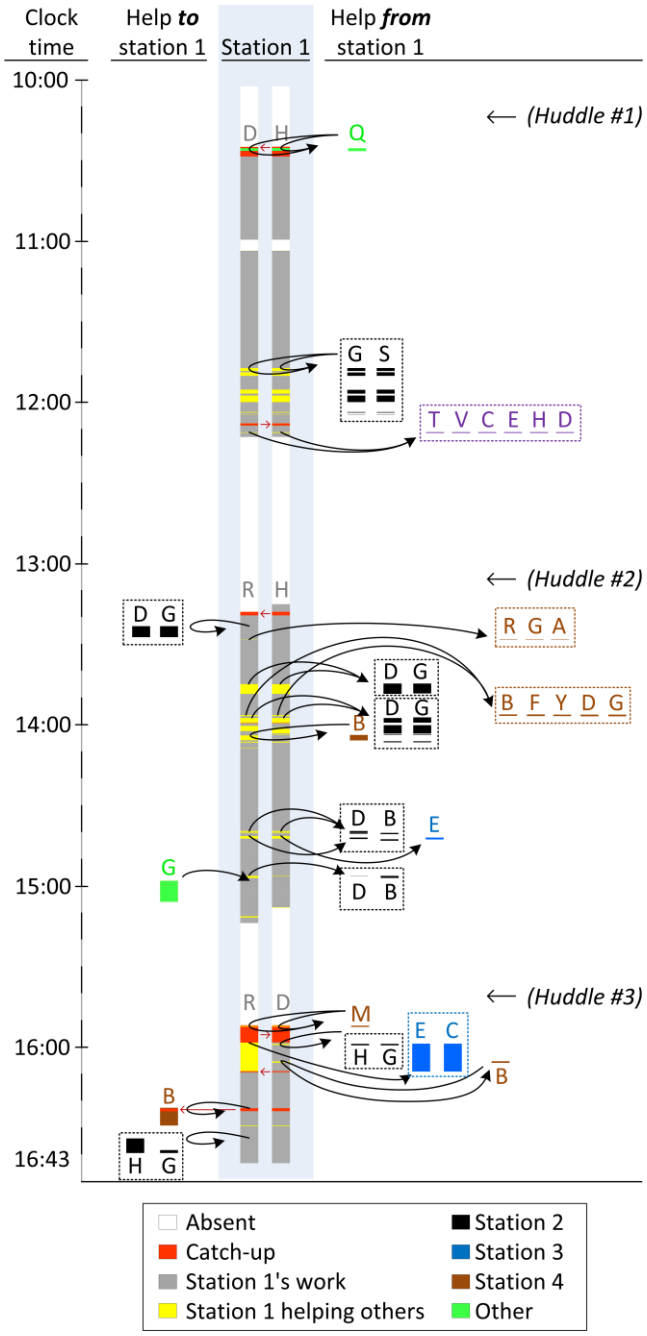

Fig. 2. An interaction score showing timing and frequency of extra-pair interactions involving pair assigned to station 1.

The 6:43 hours of video from this day captured 2.5 pairing sessions. The first two pairing sessions lasted 1:48 hours and 1:59 hours. The cameras caught the first 51 minutes of the last pairing session, before the cameras were turned off and removed. Each pairing session was preceded by a huddle, a standup meeting among all software developers. In total, the developers assigned to station 1 were active in pair programming sessions for 4:38 hours of this data.

IV. Results

Our main result is that a full 20% of the time, a pair was engaged in extra-pair interactions, with only 80% of the time being solely intra-pair interactions. From 10:02 in the morning to 16:43 in the afternoon (6:43 hours) there were 29 interactions between the people at station 1 and people outside of that pair, taking 56:25 minutes (20%) of the time spent in the pairing sessions.

The *interaction score* in Fig. 2 shows the distribution of these interactions over time and space. The height of each *activity bar* shows the duration of an interaction involving the developer whose initial is above the bar. The gray section labeled "station 1" contains two columns of bars, one for each of the pair of developers assigned to station 1. In this section, white bars show when the developer was away from the developer stations, perhaps on a break. Red bars show "catch up" periods when someone (pointed to by the red arrow) was being brought up to speed on the state of station 1's work. Gray bars show when the developer was active on station 1's work. Yellow bars show when the developer was helping a different station's work.

Activity bars to the left and right of the Station 1 section show other developers interacting with the developers at station 1. These bar colors indicate the station (see Fig. 1) that those developers had been working at, as noted in the legend. Sets of activity bars enclosed in dotted rectangles indicate multiple non-station-1 developers involved in the same interaction, and sequences of interactions related to the same topic. The score also shows when the three huddles occurred; one before each pair programming session.

Arrows originate at the person who initiated the interaction and reach out to those involved. For instance Q initiated the interaction with D and H at 10:26, and at 11:42 both G and S requested help from both D and H.

These interactions occurred on average once every 9:35 minutes, though Fig. 2 shows that they often were clumped together. Most of these 29 extra-pair interactions were brief (see Fig. 3), with an average length of 1:57 minutes. Of the 52% that lasted for less than one minute, some were quite short (see Fig. 4).

The number of interactants varied across these extra-pair interactions: 10 (34%) included only one person outside of the pair; 16 (55%) included both members of another; 3 (10%) included more than four people (see Fig. 5).

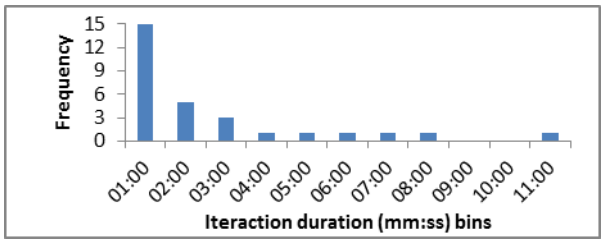

Fig. 3. Durations of extra-pair interactions. Bin labels show maximum duration for that bin.

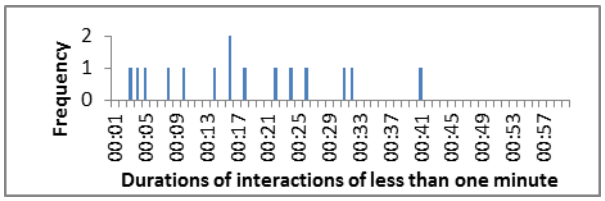

Fig. 4. Durations of sub-one-minute extra-pair interactions.

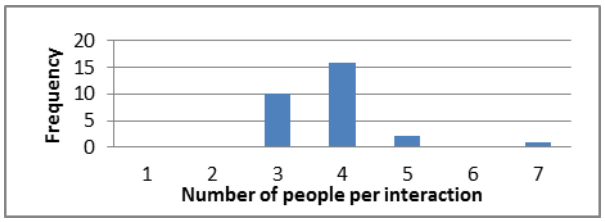

Fig. 5. Number of people involved in extra-pair interactions.

What are these interactions like? Here is a more detailed analysis of the simple interaction with Q starting at 10:26. It starts shortly after the first pairing session began:

D: I don't know how to do it.
H: You don't know how you are doing?
D: I know about what I am doing, I just don't know how to do it all.
[D sits down, clicks on his mouse, turning off the screensaver]
H: So, is the code that's [checked] out on this machine?
D: Ah.
H: Or with [something]? But there's been some work done on the story, right?

D is bringing H up to date with respect to the status of D's work on the task they will be working on, an activity we term a "catch-up". At this point, the QA engineer (Q) interrupts to ask for help with an issue she is having with a profile she uses. D and H turn their chairs around and move away from station 1 to talk with Q about her issue. H says he will provide her with a new profile, and the conversation concludes with:

Q: Okay.
H: Brilliant!
Q: Thank you. [Laughs]

D and H roll their chairs back to station 1 and continue to talk about Q's issue for a moment before returning to the catch-up:

D: [to H] That probably will work.
H: [laughs] [something…] test flight. You just sort of beat on it. Eventually it does what you want. Um.
H: Okay. What have you done?
D: Ah. There is a safe creation controller…
H: Yeah! We started on that. Cool. Yeah. And then…

The conversation continues with D largely answering questions from H about the work that D has done up to this point. After 2:21 minutes of this the conversation shifts to H answering questions from D about how the code works and possible ways to proceed, which we take as the point at which the collaboration shifts from catch-up into figuring out what how to proceed with the task.

What is the nature of this interaction? The catch-up portions can be considered part of the design activities included in Laura Williams's definition of pair programming as "a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code, or test" [20, p. 311]. This is the definition used in virtually all of the literature on pair programming. But what about the interaction with Q? The two developers are no longer working on their task, nor at their pairing station, nor just in a pair, so it does not fit the above definition of pair programming. It could be viewed as an interruption to the pairing, something that interrupts work and perhaps should be minimized. But do the people in this organization represent this interaction as an interruption to be minimized? And, as a researcher, are these extra-pair interactions part of the material to collect when studying pair programming?

In an interview, the VP of Engineering in this organization noted that "*Pair programming is a technique that is part of an approach. […] We tend to work in a highly collaborative fashion. And, yeah, pair programming is one of the core techniques that we do as part of it, but it is only a part of what is going on.*" Pair programming is a separate concept in their vernacular, but it is not represented as a separable activity; it is represented as highly intertwined with other organizational structures and practices.

For instance, they cluster the developer workstations in the center of their organization to "*maximize easy information sharing both within and between groups*" [21, p. 121] and thus enable the ad hoc push and pull of audio and visual signals across the pair's membrane to other pairs and the rest of the organization. The VP stated that "*As a team we actually made that choice. We said maximize interactions, ask questions right away. We didn't say wait until the two hour pairing session is over before you approach one of these pairs. […] we prefer to have the tradeoff go the other way. Pay the cost of context switch and interruption because we think that it makes us faster and better in the long run. It's better to not have the person who is blocked stay blocked.*"

The value they placed on these extra-pair interactions was further illustrated by a story the VP told about when they realized there were too few extra-pair interactions: "*We were trying to ramp up our availability, dev availability, to other parts of the company for questions and interactions 'cause noticed that the fact that we're pairing all the time actually created impedance to non-developers coming in and asking questions 'cause they felt that they were interrupting. […] they didn't want to interrupt and they didn't want to come in and ask even though they had important questions. […] So we made a sign called Ask a Dev. It was like a giant arrow. We […] put it on a giant clothespin and we would set it on the table […] and if there was a person who wasn't paired up who was a solo they would always make sure to sit it right next to themselves so that it was immediately visible to anybody walking by. 'Oh I can talk to that person.' But if there was a pair and we didn't have a solo we would make sure one of the pairs always had it set next to them too. […] And then we'd broadcast around the company 'Hey. Come and ask us stuff any time you want. It's no big deal.' That's part of the good things of pairing is that actually one of the persons can just split off and talk to you and answer your questions and the other person gets to keep going. Right?*"

The "Ask a Dev" sign represented their valuing of extra-pair interactions, even when the sign was next to a pair. Instead of representing Q's interaction as an interruption of D and H's work, this organization recognized that for Q this interaction is a *continuation* of her work; that for her to wait to get assistance would be an interruption. And that pairing helped enable these interactions.

Most of the interactions were more textured than the above example, fulfilling multiple purposes per interaction (e.g., coordination, design, accounting, and humor), and aligning to multiple of the organization's core values of "Consumer Centric", "Creative and Technical Excellence", "Family Oriented", "Accountable", "Transparent", "Collaborative" and "Fun" [21] which, themselves, are prominently written along the top of the whiteboard wall next to the huddle area [22] in a very visual representation of their value to this organization.

V. Discussion

At one level, our results demonstrate that in this organization pairs do not work in isolation, and are not intended to work in isolation. There are frequent and ad hoc interactions between one or more of the people in a pair and people outside of the pair. In our study, pairs spent 20% of their time interacting with people *outside* of the pair. This suggests that studies of pairs in isolation may not reveal important aspects of how pairs function, and that measurements of the effectiveness of pair programming should take into account these extra-pair interactions.

While these extra-pair interactions not "part" of pair programming, this organization considers them an integral part of what it means to *do* pair programming in an ecosystem of practices. The value exchanges embedded within these extra-pair interactions represent significant enough value for this organization that they build structures and practices to encourage them. The developers do not follow normative notions that pairs are isolates; rather, they take a pair as a typical starting point and a common stable point for the complex work that they do. They treat the conceptual boundary around a pair as a permeable membrane readily communicating across it, create practices to encourage such interactions, and naturally adapt the "membership" of the "pair" in response to emerging needs in order to enable effective decisions in a timely manner. This allows the developers to readily take advantage of the larger organizational context that informs and enables their work. Given this, how could the value proposition of pair programming in this organization be fully measured and understood without attending to these extra-pair value exchanges?

Furthermore, for these developers, communicating with another pair, far from being an *interruption*, as so often characterized in the literature on pair programming (e.g., [11]),

is considered to be part of the collaborative joint work for which the entire team is collectively responsible. This is in stark contrast to claims from others, such as Steve McConnell's Construx, whose list of "The 10 most deadly mistakes in software development" [23] includes "*Noisy Crowded Offices: Developers are most productive in quiet, private workspaces. Help them stay in the Zone by minimizing distractions, interruptions and multi-tasking.*"

How might we resolve the contradiction between these two different views of interruptions? Consider the way that both groups represent work. Steve McConnell supports his assertion (Steve McConnell, personal communication) by citing evidence reported in Peopleware [24], including a commonly cited statistic that it takes 15 minutes or more to be able to reenter the psychological state of *flow* after an interruption. This evidence comes from studies of individuals doing individual work, which is a quite different representation from what we see enacted in the organization we studied. Different representations lead practitioners and researchers to take different stances with respect to what constitutes work, how to measure effectiveness, what and how to study it, and what evidence to collect.

It may be that the research focus on individuals doing individual work creates a blind spot around how flow and interruptions function in highly collaborative situations like pair programming. Chong and Siino [11], for instance, show that when developers worked in a pair, instead of alone, interruptions were shorter and the interrupted developer more quickly returned to their task (instead of forgetting what they had been working on).

Nor does the organization we studied represent pair programming as an isolated practice; they represent it as part of a complex interdependent system of structures and practices designed to support highly collaborative work. As the VP of Engineering stated, "*pair programming is one of the core techniques that we do as part of it, but it is only a part of what is going on.*" They configure their workspace to support the permeable nature of the conceptual boundary between the pair and the rest of the organization. Rolling chairs allow individuals to easily connect with others and physically "enlarge" a pair in an ad hoc manner. Thus, this organization embodies a principle that the *physical configuration of the workspace symbolizes the space of interactional possibilities* within the software development team. This is in contrast to workspace configurations that physically isolate developers; while such isolation does not mandate that developers work alone and in private, it nonetheless symbolizes the normative form of desired interaction within the organization. Similarly for isolated pairs.

We are not claiming that our statistical results generalize outside of the organization that we studied. What we claim instead is that these results show that treating pairs as normative, prescriptive units of production (for practitioners) and analysis (for researchers) may misrepresent the essential character of human interaction within a setting. Consider, for instance, the difference between four pairs working radically collocated, and a single pair working in an individual office. The nature of how they work and their value exchanges might be substantially different, with substantial differences for the net effectiveness of the practice.

In summary, our intent here is to challenge not the definition of pair programming, but rather the nature of *how* pair programming is *studied*. Focusing only on interactions between the two members of a pair, and only on the types of activities covered by the common definition of pair programming, will miss many other value exchanges between members of a pair and their enclosing organizational fabric. These other value exchanges may be important to how pair programming works, and whether pair programming can be effective.

VI. Conclusion

This empirical study complements and resonates with other empirical studies of professional software developers that have used observation and videos to reveal nuanced and fine-grained behavior of developers in pair programming sessions [3], [11], [25], [26]. Such studies are starting to unpack the nature of pair programming as it unfolds in situated dynamics of industrial settings.

This study adds to the discourse clear evidence of the frequency, duration, and nature of extra-pair interactions between developers in a pair and other people in their organization. Ad hoc meetings are frequent during the course of pair programming sessions, and the sociality of the pair programming practice extends outside of the pair. The professional software developers in the organization we studied do not create artificial boundaries around the pairs. They intentionally configure their workspace and social conventions to enable peripheral awareness of what is happening nearby, and their pairs do not work in isolation.

This study highlights the need to consider pairs as part of a larger organizational setting. To understand the total value proposition of a practice like pair programming we may have to expand our unit of analysis to attend to the value exchanges between a pair and their containing contexts as they are enacted in situ. Studying pairs in isolation misses important aspect of how pairs function.

Acknowledgment

We thank our study participants who have been so generous to extend to us the level of trust that is critical for this type of research. We thank Elizabeth Davis, Josh Tenenberg, and Skip Walter for their critical comments on this paper. This work was partially funded by a 2012-2013 Worthington Distinguished Scholar award, and a UW Bothell CSS Graduate Research award to the primary author from the University of Washington, Bothell.

References

[1] L. Suchman, "Making work visible," *Commun. ACM*, vol. 38, no. 9, pp. 56–64, Sep. 1995.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley Professional, 1999.

[3] S. Salinger, F. Zieris, and L. Prechelt, "Liberating pair programming research from the oppressive driver/observer

regime," in *International Conference on Software Engineering (ICSE '13): New Ideas and Emerging Results*, 2013, pp. 1201–1204.
[4] S. Bryant, P. Romero, and B. Boulay, "The collaborative nature of pair programming," in *7th International Conference on Agile Software Development*, 2006, pp. 53–64.
[5] S. Salinger, C. Oezbek, K. Beecher, and J. Schenk, "Saros: an eclipse plug-in for distributed pair programming," in *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, 2010, pp. 48–55.
[6] D. Stotts, J. M. Smith, and K. Gyllstrom, "Support for distributed pair programming in the transparent video facetop," in *Proceedings of the Fourth Conference on Extreme Programming and Agile Methods— XP/Agile Universe*, 2004, pp. 92–104.
[7] B. Hanks, "Empirical evaluation of distributed pair programming," *Int. J. Hum. Comput. Stud.*, vol. 66, pp. 530–544, 2008.
[8] H. Hulkko and P. Abrahamsson, "A multiple case study on the impact of pair programming on product quality," in *Proceedings of the 27th international conference on Software engineering - ICSE '05*, 2005, pp. 495–504.
[9] L. Willaims and R. Kesler, *Pair Programming Illuminated*. Addison Wesley, 2002, p. 288.
[10] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson, "How does radical collocation help a team succeed?," in *CSCW'00*, 2000, pp. 339–346.
[11] J. Chong and R. Siino, "Interruptions on software teams," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*, 2006, pp. 29–38.
[12] L. Plonka, H. Sharp, and J. van der Linden, "Disengagement in pair programming: does it matter?," *2012 34th Int. Conf. Softw. Eng.*, pp. 496–506, Jun. 2012.
[13] D. Socha and J. Tenenberg, "Navigating constraints: the design work of professional software developers," *ACM SIGCHI Conf. Hum. Factors Comput. Syst.*, 2013.
[14] D. Socha and J. Tenenberg, "Sketching software in the wild," in *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, 2013, pp. 1237–1240.
[15] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall, 2002.
[16] B. B. Jordan and A. Henderson, "Interacton Analysis: Foundations and Practice," *J. Learn. Sci.*, vol. 4, no. 1, pp. 39–103, 1995.
[17] H. H. Clark and S. E. Brennan, "Grounding in communication," in *Perspectives on socially shared cognition*, vol. 13, no. 1991, L. B. Resnick, J. M. Levine, and S. D. Teasley, Eds. American Psychological Association, 1991, pp. 127–149.
[18] A. Belshee, "Promiscuous pairing and beginner's mind: embrace inexperience," in *Agile Conference*, 2005, pp. 125–131.
[19] K. Schmidt, *Cooperative Work and Coordinative Practices - Contributions to the Conceptual Foundations of Computer-Supported Cooperative Work (CSCW)*. Heidelberg, Germany: Springer-Verlag, 2011.
[20] L. Williams, "Pair Programming," in *Making Software: What Really Works, and Why We Leave It*, A. Oram and G. Wilson, Eds. O'Reilly Media, Inc., 2010.
[21] P. Ingalls and T. Frever, "Growing an agile culture from value seeds," in *2009 Agile Conference*, 2009, pp. 119–124.
[22] D. Socha, T. Frever, and C. Zhang, "Using a large whiteboard wall to support software development teams," in *Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS'15)*, 2015, pp. 5065–5072.
[23] Construx, "The ten most deadly mistakes in software development," 2013. [Online]. Available: http://info.construx.com/rs/construx/images/Construx-10-Most-Deadly-Mistakes-in-Software-Development.jpg.
[24] T. DeMarco and T. R. Lister, *Peopleware: Productive Projects and Teams*, First. Dorset House Publishing Co., Inc., 1987, p. 188.
[25] L. Plonka, "Unpacking collaboration in pair programming in industrial settings," The Open University, 2012.
[26] J. Chong and T. Hurlbutt, "The social dynamics of pair programming," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 354–363.