Face-to-Face Collaboration in Formative Design: Working *together* at a whiteboard

Design at a Whiteboard

John Rooksby, University of St Andrews
Nozomi Ikeya, Keio University & Palo Alto Research Center

Abstract To successfully collaborate in a creative design session, software developers need to achieve and maintain a shared focus, need to encourage and challenge each other, and need to manage their working relations in what can be a stressful situation. This paper describes six key ways professional software developers do this. These are illustrated using examples from a video study of professional developers designing at a whiteboard.

Introduction

Software development is not a matter of applying technical procedures to solve technical problems[1,2]. On the contrary, it requires creativity, discipline, debate, cooperation and collaboration; it requires developers to work on what are often ill-defined problems and to identify and address the needs of customers and users. This is particularly evident in the formative stages of software design. Formative design is often highly discursive, involving brainstorming and the exploration of ideas and alternatives. As Linden Ball et al[3] explain, when beginning work on any new problem, the problem itself is likely to be ambiguous; the work will be as much concerned with understanding and defining the problem, as it will be with establishing potential solutions. It is necessary for developers to work together at this stage. Only when the problem has been established can developers begin to divide the work into separate tasks.

In this paper we focus in detail upon the forms of interaction and collaboration that underlie reasoning in formative design. We look not at the products of collaboration (the ideas and decisions) but at the nature of collaboration itself (conversation, writing, and embodied interaction). We argue it is important for developers to recognise and build upon their skills in interaction and collaborative work.

Suggested call-out

Studying Collaborative Work

Our research is at the intersection of software engineering and sociology. Our interests lie in an area known as ethnomethodology (the study of people's methods[8]). While many sociologists and psychologists study what lies behind the scenes, searching for the hidden structure of phenomenon, such as cognitive processes, power relations and so on, our own interests bring us to study what

happens in a real course of action. We are interested in what is necessary for participants in an activity to pay attention to, to be aware of, and to do in the presence of others in order to achieve something they are to accomplish (in this context, software engineers' work of collaboratively thinking through ideas for the initial system design). As Ian Sommerville[9] has explained, such detailed descriptions of how people work and interact with each other can provide useful insights for the improvement of technologies and methods. Key ethnomethodological studies of software development in practice include studies of design reasoning by Wes Sharrock and Bob Anderson[10], work looking at configuration and testing by David Martin et al[11], and work looking at project management by Erik Vinkhuyzen and Nozomi Ikeya[12]. Ethnomethodological studies of whiteboard and blackboard use include work by Lucy Suchman[6] (on whiteboard use in cognitive science) and Christian Greiffenhagen and Wes Sharrock[7] (On blackboard use in mathematics). Ethnomethodology and ethnography are sometimes confused. Crudely put, the former can be considered an analytical viewpoint, the latter a research approach. Not all ethnographers are ethnomethodologists, for example ethnographic work by Helen Sharp et al[13] focuses on culture rather than people's methods. Not all ethnomethodological work is ethnographic in the strict sense, for example this paper is solely based on recorded video data.

How do developers collaborate?

Few software professionals will need convincing *why* they need to collaborate, particularly during the formative stages of design. Briefly, collaboration is important because a) where a problem is ambiguous, talking it through should help articulate it, and b) it is important to agree with others what the problem is, and how it will be solved, particularly if more than one person is to work on solving it. Collaborative work needs not just to occur, but also to be done effectively. If, during formative design, collaboration is not done well, developers may: withhold ideas or withdraw completely from debate, fail to elaborate or explain their ideas, lack a sense of ownership of the design, and end with different ideas of what has been decided. Failures in collaboration may lead to a complete breakdown of a design session. However, smaller, less noticeable problems are potentially more harmful; it may not be noticed when potentially important perspectives are lost, when decisions are made without enough consideration, and when there are unnecessary ambiguities in the outcome. Small, unnoticed failures can lead to problems that are expensive to correct later.

It is not our intention to argue the case for *why* formative design benefits from collaboration. Rather, we believe more attention needs paying to *how* collaboration is done in practice. Collaborative work in software development can take different forms (face-to-face or remote, synchronous or asynchronous, between small or large groups, and so on). In this paper we will specifically explore face-to-face interaction.

We will use examples of interaction around whiteboards, a medium commonly used in formative design[4]. We are not going to present an alternative to current whiteboard practices, but want to unpack the detail of common but taken-for-granted practices; we believe it is important to unpack what Janet McDonnell[2] terms "the phenomena that serve collaboration itself".

We have obtained and analysed videos originally produced by André van der Hoek and Alex Baker[4], and will use these to illustrate six practices professional developers engage in during formative design. In the study, professional developers were asked to work in pairs to design traffic simulation software for educational use. Van der Hoek and Baker judged the three best performing pairs in their study to be those from AmberPoint, Intuit and Adobe. Each produced an imperfect[5] but respectable design. For us, the most interesting achievement of these three pairs is not their product, but the fact that each pair was able to work productively over a sustained period. We are interested in how they remained coordinated and focused, and so we have closely examined how they talk and orient to each other through the sessions. Six practices we believe underlie their successful collaboration are:

1. They pay attention to each other
2. They maintain a shared focus
3. They talk through and sketch out ideas
4. They are open to each other's ideas
5. They seek agreements and acknowledge disagreement
6. They maintain a sense of humour

These practices are verbal, visual and embodied and hence we will draw on quotes and images to illustrate sequences of action. At a high-level, none of these practices should be a surprise. It is the details of these practices that we find interesting. We explore these details below.

1: Paying attention

Paying attention is an important part of collaboration, this much we hope is obvious. But what does it take to pay attention in a fast moving, complex, interruption prone design session? For the professional developers in the study, paying attention was an active not passive activity. They didn't listen to each other passively, but also interjected with comments ("yeah", "I like that"), made regular eye contact, and give supporting gestures (such as nodding). They didn't watch each other passively; they didn't just move their eyes but physically positioned themselves where they could attend to what the other was doing. Paying attention was a two-way activity, the person talking or writing at the board also listened to and often make eye contact with the other. Attention was also paid very conspicuously; it seems important that the developers paid attention but also that they were seen by the other to be paying attention. This became particularly apparent in their the handling of interruptions from mobile phone messages (something that happened in two of the sessions). Phones were quickly silenced; attending to them was treated as unimportant. On the one occasion when a developer did check their phone, this was done swiftly in a

movement lasting less than 2 seconds, and this was timed to coincide with a point where their partner had turned their back to write something.

2: Maintaining a shared focus

Another thing the professional developers did was to focus together on the same issues. The talk in each design session was conversational, with the developers not simply blurting out anything that came to mind, but discussing and exploring topics together one at a time. There was not just a verbal but also a physical, embodied dimension to the way focus was maintained. For example, when one moved to a particular place on the whiteboard to focus on a particular section of a diagram or list, the other (unless they were sitting) would usually move towards that place too. When one developer stepped back from the whiteboard, the other would often step back too. As Lucy Suchman[6] and Christian Greiffenhagen & Wes Sharrock[7] have noted of cognitive scientists and mathematicians working at white and blackboards, such a simple thing as stepping back from a board can be significant; when the developers did so it would often accompany a turn in conversation away from a particular detail and to a broader issue. So in this situation, to step back when your partner steps back can give a clear signal that you too are moving from a particular detail and onto something else; coordinated body movements can conspicuously signal the retaining of a shared orientation to a problem. Figure 1 shows a sequence in which the Adobe pair both focus on the board (frame 1), then make eye contact (frame 2), and then step backwards one after the other (frames 3 & 4). Their conversation during this sequence went from a focused to a more general discussion; they talk about the area of the board they are stood near to (frame 1), and then step away as they talk more generally about the design (frames 3 & 4).

*** fig 1 about here ***

figure 1: Developers talking at the whiteboard

3: Talking through and sketching out ideas

For the professional developers, the whiteboard was not simply a device for logging ideas and decisions but also acted as a device for supporting discussion. Many markings on the whiteboard were discursive devices that could be erased or transformed later. For example all three pairs used a diagram of a traffic intersection as a discursive device, using this to work through the properties of intersections and drivers' behaviours. Sketching and writing was not the only activity at the board. As Lucy Suchman[6] has discussed, whiteboards can serve as an object of mutual orientation irrespectively of whether they are currently being written on. Certainly, the developers spent long periods in which they discussed what was on the board, adding nothing or little to it. During these periods, the discussion took place at the whiteboard and was kept relevant to what was already on it. In talking about what was on the board, the developers employed a repertoire of gestures. As Christian Greiffenhagen and Wes Sharrock[7] have noted of mathematicians, gestures suit several purposes. They can be used connectively, for making specific relevancies clear. For the developers, this included standing near

the items being discussed (even if nothing is to be added), pointing, waving to an area, masking with hands, and circling items. Gestures can be used to integrate items on the board. For example it was often necessary for the developers to physically point out a connection between two or more items on the board. Other gestures were more illustrative, for example waving a hand to indicate the flow of traffic.

4: Being open to ideas

Being open to other people's ideas is often easier said than done. The professional developers managed to be receptive to and encouraging of each other's inputs and opinions, and tried to make sure they raised any objections they had. The developers seemed not to want to show off or impose their position but to discuss and explore it. The developers encouraged ideas from the other, through doing things like asking questions or telling the other to take a lead in writing at the board. Interruptions and interjections from the other were rarely talked down or dismissed but given consideration. More subtly, we noticed the developers introduced ideas in a tentative way, and only solidified these if there was agreement. Something the Intuit and AmberPoint pairs did was introduce ideas using the word "I" or "my", and treat them as tentative until agreed as something "we" need or want. For example when dismissing an issue as beyond the scope of the design one of the Intuit developers used the phrase "I think we don't really care". Here we can see a personal opinion "*I* think" used to introduce a candidate decision "*we* don't really care". The other developer then confirmed this as shared decision with the phrase "right, *we* don't care". Tentativeness is valuable as it gives room for questioning and contradiction, positions are not forced on the other but talked through. Janet McDonnell has explored this issue more widely[2].

5: Seeking agreement and acknowledging disagreement

Something we found striking about the professional developers' working was the pervasiveness of agreements. Even when the developers did not see eye-to-eye on a matter, they spent much more time finding what they agreed on then what they disagreed on. Confrontation was constantly downplayed in order to find common ground. Disagreements seem to have been made very reluctantly and, strikingly, the developers 'agreed their way through disagreements'. So, for example, when one developer disagreed with the other about the probability of cars making particular turns needing to vary according to the time of day, this developer still found positive things to say. Rather than confront this idea, the developer said "potentially" and then complimented the idea as "it's realistic". If disagreements could not be worked out, the subject would eventually be changed. Disagreements could be put to one side, but did tend re-emerge later. There certainly are risks if disagreements are never worked out, but it seems more important during formative design, particularly during what Linden Ball et al[3] describe as "the first pass" of ideas, to articulate ideas and alternatives than it is to jump to positions. In his study of the same videos, Michael Jackson[5] points out problems including a failure to adequately challenge unnecessary complexities. Complexities such as varying the probability of turns according to the time of day really ought to be challenged, but from the above

it should also be clear that if we want developers to challenge each other more, these need to be made in an appropriate way that is well timed and does not harm the accumulation of ideas. As Janet McDonnell[2] points out "accommodating disagreements" can be one of the things designers must do in order to get anywhere.

6: Keeping a sense of humour

For the most part the professional developers remained serious and focused, but from time to time they would laugh and joke about their design. Academics tend to ignore workplace humour, but for the developers this was an intrinsic part of getting the work done. One reason for laughing and joking was seemingly to mask embarrassment and awkwardness. Two of the pairs seemed to find it awkward to get started and so both employed humour in their first moves. This enabled them to get a footing and move on to serious working. Another function of laughing and joking between developers was to deal with mounting pressure and complexity. For example, humour often followed a period of intense discussion, and one of the pairs began to laugh and joke when it become clear they were running short of time. On all but one occasion, when one developer initiated laughing and joking the other would reciprocate it. Initiation seemed to occur at appropriate times where this would not interfere with serious discussion. On several occasions initiation was noticeably preceded by a look and a smile, as if one is checking that the other will act reciprocally if they do joke or laugh. The laughing and joking was always brief and always seems to relax the participants. Laughing and joking is an important aspect of collaborative work, it is something that helped the participants manage the pressure of the task and the difficulties of working together.

Collaboration and Professional Practice

Researchers often delve deeply into the ways ideas are structured and navigated but rarely discuss the forms of collaboration and communication that underpin these. Under six ostensibly simple headings, we have delved into just how it is developers manage to sustain a productive collaborative activity. We believe it is valuable to draw attention to this detail for several reasons, primarily:

- Students often do not come ready equipped with these skills, they need to learn about these as much as they do about procedures and technologies.

- Professionals can improve through reflection on practice. The developers featured in this study collaborate well but not perfectly. For example, their mistakes include a failure to adequately challenge unnecessary complexities[5].

- Researchers involved in the innovation of methods and technologies to support software design need to understand these practices in order to support and change the ways designing is done.

In this study, we have analysed what we have come to think of not as expert behaviour, but as professional behaviour. Software development is too broad a field

for any person to be expert in every aspect of it. The participants in this study are clearly talented, but their expertise is not specifically in the development of simulation software (a paper by Michael Jackson[5] gives a more expert view on how the task could have been best completed). The developers were also put in a difficult situation, designing in front of a camera, with a two-hour limit, with no contact with others, no Internet, and working completely from scratch. Professionalism is not the same as expertise. A professional does not (or at least ought not to) stop being a professional when working under pressure and outside of their area of expertise. We think the ways the participants did their best to work together and work productively in a difficult situation exhibits such professionalism. Their designs were not necessarily expert designs, but they were respectable, and their designing was done in a disciplined, creative way that we all have much to learn from.

Acknowledgement
Results described in this paper are based upon videos and transcripts distributed for the 2010 international workshop "Studying Professional Software Design", as partially supported by NSF grant CCF-0845840.

References

1. C.R.B. de Souza, H. Sharp, J. Singer, L. Cheng, and G Venolia "Cooperative and Human Aspects of Software Engineering." IEEE Software, Nov/Dec 2009, pp. 17-19.
2. J. McDonnell "Accommodating Disagreement: A Study of Effective Design Collaboration." Design Studies, 2011, in press.
3. L. Ball, B. Onarheim, and B. Christensen "Design Requirements, Epistemic Uncertainty and Solution Development Strategies in Software Design." Design Studies, 13, 2010, pp.567-589.
4. M. Petre, A. van der Hoek, and A. Baker "Editorial (Studying Professional Software Design)". Design Studies, 31, 2010, pp.533-544.
5. M. Jackson "Representing Structure in a Software System Design." Design Studies, 13, 2010, 545-566.
6. L. Suchman "Representing Practice in Cognitive Science." In M. Lynch and S. Woolgar (Eds.), Representation in Scientific Practice. Cambridge, MA: MIT Press, 2011, pp.301-321.
7. C. Greiffenhagen, and W. Sharrock "Gestures in the blackboard work of mathematics instruction". In Proceedings of the 2nd Conference of the International Society for Gesture Studies, Lyon, France, June 15-18, 2005.
8. M. Rouncefield, and P. Tolmie (2011) *Ethnomethodology at Work*. Farnham: Ashgate, 2011.
9. I. Sommerville, T. Rodden, P Sawyer. and R. Bentley "Sociologists can be Surprisingly Useful in Interactive Systems Design." Proc. People and Computers II (HCI92), New York, 1992, pp.341-353.
10. W. Sharrock and B. Anderson "The User as a Scenic Feature of the Design Space." Design Studies 15, 1, 1994, pp.5-18.

11. D. Martin, M. Hartswood, R. Slack, and A. Voss "Achieving Dependability in the Configuration, Integration and Testing of Healthcare Technologies." Computer Supported Cooperative Work, 15, 5-6, 2007, pp.467-499.
12. E. Vinkhuyzen, and N. Ikeya "Rethinking How Projects are Managed." In P. Szymanski and J. Whalen (Eds.), *Making Work Visible: Ethnographically Grounded Case Studies of Work Practice*. Cambridge: Cambridge University Press, 2011.
13. H. Sharp, H. Robinson, and M. Woodman "Software Engineering: Community and Culture." IEEE Software, Vol.17 No.1, Jan/Feb 2000, pp.40-47.

John Rooksby is a Research Fellow in Computer Science at the University of St Andrews, UK. His research interests lie in human and organisational factors in the development and operation of software. He holds a PhD in Computer Science from the University of Manchester. Contact him at jrnr@st-andrews.ac.uk

Nozomi Ikeya is a Professor at Keio University, Japan and also a Senior Research Scientist at Palo Alto Research Center. She conducts ethnographic and ethnomethodological studies of knowledge management, focusing on "knowledge in action". She holds a PhD in Sociology from the University of Manchester. Contact her at nozomi.ikeya@a8.keio.jp

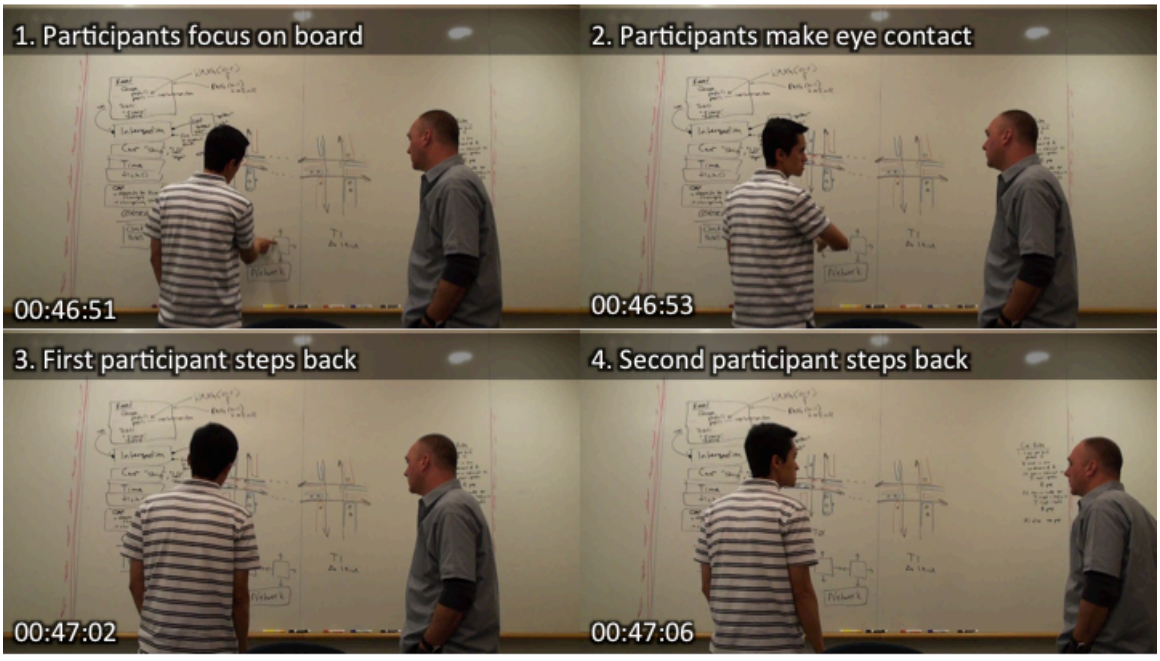

fig.1