Learning Processes in Distributed Software Development: A PhD Thesis Synopsis

Anders Sigfridsson
Interaction Design Centre
University of Limerick
ERB-1002
+353-86-2333625
anders.sigfridsson@ul.ie

ABSTRACT
This paper presents a brief synopsis of a PhD thesis currently being written on the topic of collaborative software development practices in the context of globally distributed projects. It explores issues related to knowledge in software developers activities when dealing with particular problem situations in practice. The research builds on participant observation in a global corporation and a free/libre open source community. This paper discusses the research purpose and methodology, and presents a selection of empirical episodes to exemplify the practical issues investigated. It concludes with some initial analytical points about these episodes.

Categories and Subject Descriptors
H.1.2 [**Information Systems**]: User/Machine Systems – *human factors.*

General Terms
Human Factors, Theory.

Keywords
Distributed software development, practices, collaboration, knowledge, learning.

1. INTRODUCTION
In contemporary society there is a trend towards increasing collaboration across distance which affects work for many people [1]. Settings where people actively collaborate with remote colleagues are therefore significant contexts to consider for researchers studying how people are using modern information technology. One domain where distribution is particularly far-reached is software development. Outsourcing relationships, global organizations, inter-organizational alliances and dispersed free/libre open source (FLOSS) communities are since long commonplace [2,3]. Work in these distributed settings are both value-creating in a society pervaded by computer technology and a salient example of modern collaborative activities mediated by information technology. In this paper I present a brief synopsis of my PhD thesis currently being written about collaborative software development practices in the context of distributed projects.

My thesis investigates a particular class of problems that I've identified in distributed software development practice. I call this a problem of *emergence*, as it is about how software developers respond and adapt to new situations while accomplishing their daily work. What faces software developers who work in distributed projects – be it in a global organization or in a FLOSS community – are constantly emerging conditions and goals. The technology being implemented in the application, but also used for accomplishing the work, is autonomously evolving and branching. An increasing turn-over and competition due to border-crossing dispersion also contribute to a constant evolution of requirements, resources and relationships. My central concern is how software developer teams in global projects are dealing with this inherent emergence. Characterizing this particular aspect of software developers work in distributed projects is the main objective of the thesis.

I analyze this aspect of their work in terms of knowledge development. Problems related to knowledge development and sharing have been highlighted as fundamental in distributed software development [4,5]. My perspective is that working and learning are inherently interrelated in practice [6]. To further our characterization of knowledge-related issues in distributed software development practice, I contribute an empirical analysis of how this convergence occurs in particular work episodes. My argument is that the processes through which software developers deal with the problems I'm investigating is about learning in practice.

To address these questions I have adopted a qualitative, empirically grounded approach. I've performed participant observation in two different settings: a global corporation and a FLOSS community. My unit of observation was software development teams in distributed projects, giving me a local perspective on the phenomenon. In parallel, I employed grounded theory coding to process the empirical material. My analytical approach in the thesis is to produce thick descriptions of certain episodes extracted from this material. What emerged as significant analytical categories during the coding process is rendered into

I-HCI '09, September 17-17, 2009, Dublin, Ireland.

vignettes illustrating certain idiosyncratic chains of events and activities as the teams encounter novel situations. This allows me to show what the common problem is across the different episodes and to analyze the process that the software developers went through as they encountered it. At the time of writing, the field studies and major analysis of the empirical material is complete, while thick descriptions of observed practice are still being developed.

1.1 Knowing in distributed software development

Knowledge is, of course, a subject in many different academic discourses. We can speak of knowledge-related issues in organizations, computer-supported collaborative work, human-computer interaction, as well as software engineering as a particular activity. For the present purposes, let us look at two major knowledge-related issues that researchers have discussed in distributed software development. A concept that has become central is "mutual knowledge" [7]. It refers to what is known by a person, but, more importantly, also to her awareness of who else knows it. It's advocated that building up and maintaining mutual knowledge is an essential problem in distributed projects and this has been the attention of much empirical research [7]. The frequently highlighted issue of "embededness" [5] is closely related to mutual knowledge in researchers thinking about the phenomenon. It is argued that knowledge is embedded in specific local contexts and that this impedes the sharing of knowledge among participating groups in a distributed project. This notion has been used to explain a number of empirically investigated problems [4].

The purpose of my research can be further defined against this backdrop, because there is a difference in the unit of analysis. As we can see in the examples above, the main concern in previous research on knowledge-related issues in distributed software development has been the distribution and sharing of knowledge within projects. In contrast, I am bringing out for inspection the way knowledge is enacted and developed in the actual activities through which distributed software development is accomplished locally. This is taking a practice-based perspective on knowledge, emphasizing how it is enabling practice [8]. Understanding how knowledge is communicated and co-ordinated within distributed projects is important, but there is also a need to delve deeper into how it is enacted in practice. In contrast to training and other institutionally sponsored efforts, what I focus on is informal learning [9]. My contribution to our understanding of distributed software development, then, is a practice-based perspective on how knowledge is enacted and developed in dealing with the practical problem of dealing with novel situations.

2. METHODOLOGY

The research challenge, then, is to investigate how software developers in distributed projects deal with novelties while they engage in their daily practice. This is to be analyzed in terms of knowledge and learning in practice. My strategy has been qualitative and empirically grounded. I've adopted an open-ended approach, characterized by exploratory investigation and dialogue between posed questions and empirical findings. Some setting and certain activities have been studied and analyzed, iteratively leading to more developed research questions and further investigation of certain aspects. I've achieved this through extensive field studies using participant observation in combination with grounded theory coding. This allowed me to access the actual practices through which the work was accomplished in these settings and to tune the analysis to issues that were important in the particular contexts studied (rather than prescribed issues derived from some theoretical or empirical framework).

Two cases have been studied. First, a development team that was part of a globally distributed project in a multinational organization. Second, a core group of developers in a FLOSS community. The reason for studying these two particular cases was not to compare proprietary software development in organizations to open source software development in volunteer-based communities. Instead, the argument is that corporations and FLOSS communities essentially are different settings for the same substantial activity. This selection of cases allow me to show that dealing with emergence is a significant problem to distributed software development practice in different settings. The difference in setting also give rise to contrasts that are useful in the analysis of what characterizes this problem.

The field studies are completed at the time of writing and what remains is working up the produced material into a thesis. What emerged as significant analytical categories during the field studies will be rendered into coherent vignettes that illustrate certain episodes. Based on these thick descriptions, I will first show the common characteristics of the problems across the different episodes. Then I will analyze the observed processes in terms of knowledge being enacted. My analysis is thus narrative in style, as described by Becker [10], its goal being to unveil the steps of the process by which something is dealt with in particular situations (ibid. pp. 57-58 and 60-63).

The remainder of this paper is meant to give the reader a few examples of vignettes being developed. They'll give a sense of what characterizes the practical problems that I am focusing on. I'll also discuss some preliminary analytical points to conclude the paper, though it has to be pointed out that these are still very early on and should be considered more as topics for discussion than finalized conclusions.

3. EMPIRICAL EPISODES

3.1 Encountering extraneous change

One of the observed software development teams worked in the Dublin office of a multinational organization. They were the development team of a globally distributed project, collaborating with other teams and individuals in four other sites of the organization. For simplicity, I'll give the project the pseudonym LPAS and the team in focus will thus be called the LPAS development team. This first vignette illustrates an episode where the LPAS team had to adapt the application under development following an update in the software framework it built on.

The main events of this episode took place over a period of about two weeks (in addition, there were rippling effects during the following two months). A patch was expected at the end of those two weeks. At the same time, an update of the software framework was imminent. We'll jump in towards the end of the episode, taking a look at the key events, a turning point: at a team meeting two days before the patch, they were discussing whether to support the new framework version or not in the upcoming patch. They knew there were would be some issues integrating it and

because it wasn't in use by any customers at the time, it was argued that they would not support it. The local program manager corroborated the decision, saying that they would implement the new framework version in the next development phase instead. However, on the afternoon of that same day, the projects release manager – who was located in another site in the US – reversed the decision and demanded support for the new framework version.

Suddenly, the LPAS development team needed to learn in detail what was new in the framework update and test it with their application to see how it needed to be adapted. Since they knew there were issues and that it related to the installation process, it fell to the developer responsible for this part to investigate exactly what the issue was. As it turned out, it was quite serious. A "perk" in the new framework version caused the installation process of their application to fail. But since time was short – the patch was due the next day – they had no other choice than to find a solution in their application. The developer responsible for investigating the issue took help from one of the teams technical leaders and together they traced the error and came up with a temporary fix for this patch. The patch was finished on time and shipped with instructions on how to apply the fix. Interestingly, there were also some rippling effects of this event during the following 2 months in the form of discourses regarding strategies for how to deal with updates of frameworks and structures.

This vignette, then, talks about how an extraneous change was dealt with. First there was uncertainty about whether to incorporate the change, but once the decision was taken the problem turned into *how* to incorporate it.

3.2 Encountering a novel system

We stay with the LPAS team for a second vignette. What is illustrated in this one is an episode where the team utilized a new collaboration platform. This system integrated support for many different aspects of collaborative software development, including source code management, task tracking, and project planning. When I began my observations with the LPAS team in early September 2007, it had been decided that the team was to switch to this new system. But to what extent was still uncertain. At the time, the system itself was also under development and only in a beta stage. The LPAS teams lead architect championed its introduction, while primarily two of the technical leaders were the pragmatics, concerned with discussing how it would actually work for them. What happened was that the team began using the new system for task tracking and project planning, while the question of whether to use it for source code management remained open.

The LPAS developers, then, began using the system for task tracking in their daily work during the later half of September. There were no noticeable hick-ups in terms of work flow, as everyone seemed able to pick it up and work with it from the start. However, after a few weeks, some troubles began accumulating. People had begun using the system based on their experiences with similar systems in the past. But this also meant that people had slightly different ways of entering and managing tasks. There was also a problem with people not incorporating the system into their daily routines, meaning that data in the system would often be out-of-date. The result of this was an accumulating mess and much of the LPAS teams efforts during the following period was aimed at attempting to disseminate a shared practice for using the system.

The problems that this vignette talks about had two major parts. One was the uncertainty about to what extent they were going to use the new system. The other was the actual practical problems that began accumulating once they began using some of the systems services. In the end, the key in utilizing this novel system was about finding shared practices for using it.

3.3 Encountering new colleagues

Lets turn attention now to the second setting in which I performed field studies. In doing so, we move from global organizations to FLOSS communities. A first episode from this setting describes how a newcomer began participating in the community. The community in question is called PyPy and, while in many respects being a regular FLOSS project, they also employ a novel sprint-driven development methodology. While being a way to accelerate development work, it is also used as a strategy to attract newcomers and help them get started on their work in the project. When studying this community, we wished to build an understanding of what the sprints meant for someone joining the community. To get an insiders perspective on the process, I performed participant observations, joining the community as a participant and eventually attending my first sprint.

The first steps of joining PyPy were taken online, subscribing to the mailing list and announcing ones interest in the project. I explained my dual role as participant and researcher, spoke briefly about my technical background and current interests (relating to the project at hand). There was a brief discussion about my background and interests, after which others suggested relevant online documentation to look into. I was also encouraged to ask questions and begin participating in the available online forums. Together, this gave me an idea of technological architecture and the practices for contributing to it, but also about who was who in the community.

But it was well at my first actual sprint that the preparatory function of these online activities became apparent. At the sprint, we newcomers (there was myself and one other person) were effectively taken by the hand. Tutorials were organized based on our then current knowledge and what were were interested in working on. What's particularly interesting is how this was really a direct continuation of the online activities. We knew roughly what topics we were interested in, while the other members had a sense of our knowledge and areas were we needed improvement. After the tutorials, we were allowed to pair up with experienced developers to work on topics of choice. This helped us get a direct, hands-on experience with working on what we were interested in.

The key point illustrated in this episode is how the PyPy community's practices for receiving newcomers focused on enabling them to achieve both technical competency and community membership. It's also important to note how the online activities and the collocated evens were interwoven in this process.

3.4 Encountering uncertainty

Lastly, the fourth episode we will look at in this paper illustrates how the FLOSS community dealt with ambivalence regarding the future focus of the project after a major release. A major part of my studies of this community was analysis of online activities, primarily in the developer mailing lists. The episode at hand took place just after the release of the 1.0 version of the system being

developed by the community. Following this there were discussions in the online forums regarding the future direction of the project. This was essentially a debate about what the purpose and intended users of the project was, as peoples opinions about what to do next stemmed in their perception of those questions. However, while several different (many opposing) viewpoints were presented, there were no decisions being made. At this point, a peripheral follower of the PyPy project writes a long entry in the debate, criticizing the current proposals and making some radical suggestions. This person, while only a peripheral follower of the PyPy project, had significant experience and celebrity in the FLOSS domain. His entry, though, was followed only by silence.

After a few days, he writes another email to the list, apologizing for the critical tone of the previous entry, but maintaining and still pressing his opinions. Again, there was only silence. When a response finally arrived, it was a core member who apologized for the silence, but also explained that people probably felt the issue hadn't been discussed properly among themselves yet. They therefore preferred to talk about it when all the core members could gather face-to-face at an upcoming sprint before discussing these issues any further with the broader community.

What's illustrated in this vignette is how the PyPy community responded to uncertainty in connection with external pressure. The core members of the community had learned over time the benefits of deferring complex or controversial discussions from the online forums to face-to-face meetings at sprints.

4. DISCUSSION

This has been a very blunt presentation of four of the episodes I am currently developing, but it has highlighted some of the key issues that I am concerned with. To conclude this paper, I'll discuss some preliminary analytical points about them. The first question to be dealt with in the thesis will be how the problems of the different episodes can be characterized in general terms. Consider the commonalities between them: though we've only seen brief glimpses of the full episodes here, I hope the reader can see how all the cases are characterized by change, uncertainty, and extraneous influence of various kinds. An important realization is that they all stem – to some extent – in the fact the these projects are situated in a broader context. The project participants encounter extraneous influence as they're working and this sparks changes that must be negotiated. While a lot of analysis and conceptual work remains, it's suffice to say that the problem in focus can be described as being about dealing with emergence.

The problems that we've seen is how software developers in distributed projects are to deal with emergence while engaging in their daily practices. The remaining question, then, is what characterizes the process by which they do so. One interesting aspect is how what happened in the episodes revolved around achieving a shared practice. Another point to make is that all of these episodes occurred – not as separate, distinct activities – but during or as part of regular work. As such, they were in themselves collaborative activities. Additionally, the episodes also show us how dealing with the problems required knowing about how to work with independently evolving platforms. Maintaining a shared understanding of exactly what platform for which the application was being developed was vital. I argue that what we're seeing in this are practice-based learning processes. It is about engagement in a community of practice and about enculturation [6].

There are a number of areas in this analysis that need to be further elaborated, but in addition to what I've mentioned above, another important question is what role the global dimension played in these processes. For example, one must ask oneself what in the problems analyzed was caused by the existence of remote colleagues. But also, what this enabled in dealing with them. Considerations like these, relating directly to the distribution of the settings, will need to be further elaborated upon in the thesis.

With that I conclude this brief synopsis of my thesis argument, hoping to have presented some interesting points for discussion, both regarding the logic of the argument and the specific topics of the examples. Again, let me point out that the vignettes I've talked about and the analytical points that I've mentioned in this last section are still very much under development.

5. ACKNOWLEDGMENTS

Work performed in the socGSD project, funded under under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI). Thanks to my colleagues in the IDC for valuable feedback and to the reviewers for their constructive comments!

6. REFERENCES

[1] Hinds, P.J. and S. Kiesler, Distributed Work. 2002, London: MIT Press.

[2] Sahay, S., B. Nicholson, and S. Krishna, Global IT Outsourcing: Software Development Across Borders. 2003, Cambridge, UK: Cambridge University Press.

[3] Fitzgerald, B., The Transformation of Open Source Software. MIS Quarterly, 2006. 30(3): p. 587-598.

[4] Imsland, V. and S. Sahay, 'Negotiating Knowledge': The Case of a Russian-Norwegian Software Outsourcing Project. Scandinavian Journal of Information Systems, 2005. **17**(1): p. 101-130.

[5] Nicholson, B. and S. Sahay, Embedded knowledge and offshore software development. Information and Organization, 2004. **14**: p. 329-365.

[6] Brown, J.S. and P. Duguid, Organizational Learning and Communities of Practice: Towards a Unified View of Working, Learning, and Innovation. Organization Science, 1991. **2**(1): p. 40-57.

[7] Cramton, C.D., The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration. Organization Science, 2001. **12**(3): p. 346-371.

[8] Cook, S.D.N. and J.S. Brown, Bridging Epistemologies: The Generative Dance Between Organizational Knowledge and Organizational Knowing. Organization Science, 1999. **10**(4): p. 381-400.

[9] Marsick, V.J. and K.E. Watkins, Informal and Incidental Learning. New Directions for Adult and Continuing Education, 2001. **89**: p. 25-34.

[10] Becker, H.S., Tricks of the Trade. 1998, Chicago: University of Chicago Press.