Knowledge Work Practices in Global Software Development

Gabriela Avram
University of Limerick, Ireland
gabriela.avram@ul.ie

Abstract: This paper is an exploration of knowledge work practices in a distributed software development setting. The author has undertaken an empirical study in the Irish subsidiary of a multinational company over a 16-month period. Our methods were inspired by ethnography; by spending an extended period of time with a software development team working on a specific project, we had the opportunity to observe real work practices in a real work setting in the specific circumstances of distributed work. The purpose of the current study is to highlight the ways in which technical and social factors are inextricably entwined in distributed work settings.

Keywords: collaboration, work practices, distributed work environments, global software development, knowledge work, mutual knowledge, transactive memory

1. Introduction

Software development is a knowledge-intensive, complex activity. In addition to its intrinsic complexity, nowadays software development has increasingly become a multi-site, multicultural, globally distributed undertaking (Herbsleb 2001). The forces behind the global distribution of software development activities are predominantly of an economic nature, turning national markets into global ones and spawning new forms of competition (Prikladnicki 2004). The phrase ‘global software development’ (GSD) has been coined to refer to “*software development that is geographically, remotely or globally distributed with the aim of rationalising the development process and products”* (Sarker 2003). The variety of settings in which GSD takes place include anything from globally distributed subsidiaries of the same organization to outsourcing arrangements, open source communities and virtual small companies with employees distributed throughout the world.

The ACM Job Migration Task Force report on globalization and software offshoring (Aspray 2006) shows that globalization of the software industry is expected to increase in the coming years, fuelled by both information technology itself, by government actions and by economic factors. The reasons for the global distribution of software development activities stay the same: cost reduction, flexibility, availability of a large and competitive labour pool, access to special expertise, good quality of work, closeness to particular markets. But at the same time, global software development poses specific challenges related to communication, coordination and collaboration that can hinder the success of distributed work (Carmel 2001). The conclusions of a workshop dedicated to global software development in 2003 listed as the main challenges for the field: the lack of informal communication, the cultural differences between distant sites and the difficulty of building trust among remote developers (Lanubile 2003).

Knowledge management issues within global software development have been highlighted in several studies. Herbsleb and Moitra (Herbsleb 2001), for example, categorize the effects of physical separation among project members along several dimensions:

- strategic,
- cultural,
- technical,
- communication related,
- knowledge management related and
- project and process management related.

Among the knowledge management issues, they mention the inappropriate level of knowledge sharing and documenting in distributed software projects. For the authors, knowledge management is just one of the aspects challenged by work distribution, seen in its concrete manifestations.

De Souza et al. (De Souza 2006) consider the management of knowledge in global software development projects absolutely essential in order to cope with the coordination and integration of multiple knowledge sources under time pressure and budgetary constraints. Building on the two dominant knowledge management strategies described by Hansen, Nohria and Tiernen (Hansen 1999), De Souza and his colleagues focus on distributed software development. The two initial approaches are: codification(focused on building central knowledge repositories, where knowledge is detached from its owner, put in context and made available across the organization), and personalization(where knowledge sharing is fostered through people-to-people interactions and dialogue). De Souza's study is based on knowledge management practices observed in more than 50 software organisations, and after taking into account three factors (the focus of the organisation, the degree of structure and the knowledge repositories in place), the authors describe three models for distributed knowledge management architectures: client-server (corresponding to codification), peer-to-peer (corresponding to personalisation), and a hybrid model, which is a combination of both.

While we agree that, taking into account the large variety of settings for global software development, choosing suitable approaches for each organisation is important, our work is not focusing on the strategic level. We have instead chosen to explore knowledge work – "the work of producing and reproducing the knowledge" (Schultze 2000) – focusing on "what people do, i.e. their work practices, rather than on what they know" (Blackler 1993). A better understanding of knowledge work practices can provide useful insights in the role of the organisational culture and of a stimulating work environment as vital for the successful organisation of distributed work.

Speaking about knowledge in the context of software organizations, Davenport and Prusak describe it as "a fluid mix of framed experience, values, contextual information, and expert insights and grounded intuitions that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of the knower. In software organizations, it often becomes embedded not only in documents or repositories, but also in organizational routines, processes, practices, and norms". (Davenport 1998, pg.5)

We regard knowing as an essentially human feature and thereby maintain that knowledge cannot be stored using technical means. What is actually stored is information about knowledge (Rus 2001). The human actors are the only ones who can process, share and create new knowledge by interacting with each other and with the technologies. Regarding knowing as something people do (rather than looking at knowledge as something that people have) "draws attention to the need to research ways in which the systems that mediate knowledge and action are changing and might be managed" (Blackler 1995). Instead of speaking about knowledge being created, shared, and used, we prefer to look at the human actors who perform all these actions in a social context.

Our work is meant to extend the existing body of studies focused on work practices in Software Engineering, by contributing an insight on the particularities of knowledge work in distributed software development environments. The purpose of our paper is the exploration of knowledge work practices in a global software development setting. In order to do so, an empirical study in an Irish subsidiary of a multinational company was undertaken. The study covered a 16 month period and employed ethnographically-informed methods. What differentiates our approach from other global software development studies of distributed work settings are: the focus on social practices rather than on technical issues, the bottom-up approach – we are observing what happens in the real setting, using a variety of methods ranging from participant observation to interviews and document analysis - and the extended span of time of our studies.

The objective of the current paper is to discuss the impact of the global distribution of software development on collaborative work practices and knowledge creation, transfer and retention. The next section includes a description of the wider context of our work. Section 3 introduces the software development team we had the opportunity to observe. Section 4 presents a number of knowledge work practices that were noted during our time with the team, and the following section is dedicated to a discussion of these practices, with an emphasis on the impact of distribution; the final section presents our conclusions.

2. The context of our research

2.1 The socGSD project

At the University of Limerick, Ireland, a group at the Interaction Design Centre was established as part of a software engineering research consortium to study the social, organisational, and cultural aspects of global software development (the short name of the project is socGSD). This project aims to explore how organizations attempt to manage the coordination of engineering work via a variety of mechanisms. In our approach, we build on earlier Computer Supported Cooperative Work (CSCW) studies dedicated to issues such as articulation and coordination work, information sharing and knowledge management practices in distributed work, and the role of organizational memory support tools. The socGSD project is investigating the nature of these issues through a variety of analytical and empirical methods highlighting both theory and practice in this domain.

Over the past 18 months, the socGSD team has been engaged in field work in several sites in Ireland where software development is being conducted involving geographically distributed teams. Our research methods mainly rely on an interpretive, naturalistic approach to data collection and analysis. This means that we study the phenomenon in the actual settings where the work activity takes place, attempting to make sense of the work through the eyes of those actually doing it.

We attempt to bring into light the diversity of ways in which distributed teams shape their work practices and achieve a shared understanding of their objectives. Another topic of interest is constituted by the ways in which people involved in software development manage to either cope with or re-shape the organizational rules and formalizations they are required to use through their work practices.

2.2 Research approach

In the ensemble of studies looking at global software development, our approach is relying on field studies of work practices over an extended period of time. Our work is informed by a long tradition of workplace studies presented in the CSCW literature.

We believe our approach complements studies undertaken from a macro-economic or strategic perspective by looking at participants in real workplaces in various global software development settings. These participants are studied in real work circumstances (as opposed to experiments), and are engaged in continually evolving working arrangements. By shedding light on the hidden aspects of these practices, we attempt to show what people actually do to get their work done in contrast with the prescribed processes.

Our data collection and analysis methods are mainly informed by ethnography and include observation, document analysis, in-context interviews, audio recording, focus groups and workshops. In this context, our study is not intended to be a simple description of the developers' perspective and experience explained in their own words, but also an interpretation about how these people's experiences can be understood in terms of the interplay between members and the researcher (Dourish 2006).

The purpose of our study is to contribute to a better understanding of the impact of global distribution of software development on knowledge work practices. How do software developers cope with distance when it comes to knowledge creation, sharing, transfer and application? What are the artefacts, tools and social practices facilitating distributed knowledge work?

2.3 The fieldwork at the described site

One of our field sites was the Irish subsidiary of a multinational company involved in software development. We were granted the opportunity to follow a software development team starting with January 2006. In the first 3 months, we participated in a number of team meetings, studied the documents in the project repository and interviewed the team manager and a few developers, in order to familiarize ourselves with the context and the work being done.

After that, we spent more than 50 days in the field, observing the activity of the team in its own work environment, participating in meetings and group activities and occupying a desk in the open plan next to the team's area. The periods of time spent in the company varied between 6 consecutive days when the team was approaching an important milestone or release, to one day weekly in order to maintain contact and awareness.

The research team was granted access to the company intranet, to the project's document repository and to the team's mailing list. The author was also allowed to make use of the company's own instant messaging system, useful both as an awareness mechanism and as communication channel with the members of the observed team. This allowed her to reach a better understanding of the ongoing activities and carry on her observation even when not present on site.

During the 16 months of observation, the author developed a good relationship with all the team members and found opportunities for conducting both formal interviews and informal discussions on various topics.

One of the best ways to observe directly the team members' interactions with people in various other locations (US, Germany, India) was the researcher's silent participation in teleconferences. The team manager, who granted us this opportunity, was extremely supportive not only by allowing us to follow the information displayed on his screen, but also by answering to our questions at the end or commenting for us whenever possible. The remote participants were made aware of our presence in the room on these occasions. The author also kept a diary and took detailed notes each day that was spent in the field.

10 months into the project, the author also travelled to one of the company's sites in Germany and interviewed five people with different roles in the collaboration between the two sites (managers, architects, technical planners).

The research team organized two workshops (one with the development team and the quality engineers, and the other with managers and some of the remote collaborators of the team) in the early phases of observation. The purpose of these meetings was to explain our methods to the team, and also to discuss our research-in-progress and illuminate some of the topics and situations we found of interest.

The data collected from the field was periodically discussed and analyzed by the extended research team, in order to identify topics, trends and problems and compare the findings to those from other similar sites where fellow research team members were observing similar processes and activities.

3. The observed project team

The project started in January 2005 with 8 developers and 2 software architects (one on site and the other one working from his home in another European country). 5 of these developers had worked together before (lead by the same development leader) on a specific component of a similar product, bringing valuable expertise to the team. A team of 4 quality engineers started its work on the product and collaboration with the developers later that year.

The development leader reported to both a local manager (dealing with administrative and HR matters) and to a second one, located on the East Coast of the US (dealing with business issues related to the new product). A user interface designer and a technical writer, both located in the US, were also assigned to the team.

Over time, the team grew to 15 people, with a high turnover during the 2 ½ years period of project duration. Only 4 members of the initial team stayed with the team until the end, both the local software architect and the development leader having to move to other projects before its end. Despite this, the team reached a good cohesion level after their first release (approximately 18 months into the project), and learned how to quickly adopt newcomers (either from inside or from outside the company) and deal with the departures.

The collocated team included people of various nationalities and with different backgrounds. Some of them had worked in the company for all their professional life, while others were student interns or had just finished their studies obtaining graduate/postgraduate degrees. The team was regarded by the other teams on site as being over-disciplined and extremely hard working, but having a good social life as a group. The cultural differences were leveraged as an advantage in the interactions with colleagues from other geographies – we witnessed phone and instant messaging conversations in no less than 5 languages other than English. It was a well-established procedure that senior members of the team worked from home 1-2 days a week to avoid commuting to work, using the same interaction mechanisms with their team as the members located in other countries (e-mail, instant messaging, phone, project databases).

While in the first year people worked on isolated components, the integration effort and the work on their first release brought people closer and contributed a lot to the formation of an efficient and flexible team. A post mortem analysis was organized after the first release, discussing the challenges encountered, the achievements, and the lessons learned. This was followed immediately by the kick-off of 2 new projects, building on the code of the previous release, but involving different technologies. In the next 9 months, the team encountered plenty of difficulties, most of them due to the immaturity of the new in-house technologies they were building on. In May 2007, the top management took the decision to relocate the work in another European country, where 2 of the underlying technologies were developed. The decision was based on a realignment of corporate priorities and on the need to reduce interdependencies between sites. The project came to a normal closure on the Irish side after the next milestone, one month after that.

The activity of the team was centred on a number of shared repositories:

- version control system for the code developed,
- defect tracking system for the problems encountered in the functioning of the code,
- project database for the architectural guidelines, specifications, important decisions in the course of the project,
- project documentation,
- work schedules and
- email archives.

For communication between the members of the team, face to face interaction (where possible), instant messaging, e-mail and phone were the most important means used. The team met weekly in a meeting room, joined by the remote members using the internal conferencing system.

4. Exemplars of practices

In this section, we will present three vignettes that illustrate work practices established by this specific team to cope with distribution. For each of them, we will mention the artefacts people use, the corresponding knowledge repositories and the role played by human actors. The vignettes are focused on knowledge creation, sharing, transfer and documentation practices that were observed by the author during her time spent with the team.

4.1 Talking the developer through his assigned work

The project started with an architect located abroad and a core group of developers in Ireland, who in the first phase developed prototypes to illustrate the product idea and its functionalities. The architect had worked for years on developing 5 similar products for different platforms during his career and had accumulated a lot of expertise in the field, being considered a “guru”. While re-developing the same ideas for an enterprise portal environment, he also had the chance to introduce new features and bring the product up-to-date. Instead of spending an extended period of time developing product specifications working in isolation from his home office in another European country, in the early stages of the project the architect preferred to interact with the developers’ team in Ireland, testing the opportunities offered by the underlying platform and on the same time building mock-ups to enable him to demonstrate his ideas to the top management and gain their support for the project. Instead of writing specifications for the developers, the architect used to develop his ideas in one-on-one phone meetings with various developers or in specially designated team meetings.

In the one-on-one meetings, the architect explained his ideas to the developer, who had then to write the code. “He is telling me what to do and how to do it”, one of the developers said to us.

Sometimes, artefacts such as UML or box & arrow diagrams, tables, text, screen shots or PowerPoint slides were sent to the developer beforehand, and the explanations were based on them, but this wasn’t the rule. Most of the time, the specifications remained very high level and the functionalities to be added were specified by the architect in emails sent to the developers assigned to each component.

After a first iteration, a code inspection phone meeting followed, where the developer shared his screen with the architect. In some of these meetings, the American User Interface designer was also invited. Speaking about one of these meetings, a junior developer confessed: ”it was very good, because it clarified a lot of problems. My last week’s work will have to be thrown in the bin, but I am much closer to a solution now. It’s basically a trial-and-error process, and it’s important to have the code inspected as early as possible”.

After the functionalities of a specific component were discussed, the general specifications for the component had to be drafted by the developer. These specifications were further used by testers (Quality Engineers) for running the functional tests, and by the technical writer for writing the product documentation. This agile practice might be pretty frequent in collocated settings, but this particular team has adapted it to work over distance. While the product expertise belonged to the architect, he found that the most rapid and efficient way to check his ideas and get the work done was “to sit down with the developer and talk him through the specific component.” During a phone interview, he admitted it was difficult (and frustrating at times) to work with junior developers, who “were missing the big picture”, unable to understand the product beyond the functionality they were supposed to develop, and lacking the chance to have seen any similar product working in a customer site. “They are like blind, and we have to guide them.” From his experience, working with a remote team he knew well for several years was not a problem – but in this case, the distance was a real challenge. The architect, coming from an agile development background, found his own way to deal with marketing driven deadlines imposed by the corporation – apparently impossible to meet.

This is an illustration of a knowledge transfer practice making extensive use of tools (phone, screen sharing, instant messaging) for alleviating distance. It does not coincide with the recommended corporate processes, but it proved efficient and helped the team deliver on time. In order to avoid e-mail exchanges that could have taken weeks to reach the same result, synchronous communication and collaboration were preferred.

4.2 Surviving the Babel Tower

Before the first release of the application in July 2006, the product under development had to be translated from English into 29 other languages, in order to make it available in all 30 languages on the date of the release. To make this happen, the company had a set of procedures, policies and tools. The role of translation testing coordinator was assigned to a software engineer who joined the team only 3 months before the localisation process started. His excellent communication and social skills, together with his eagerness to learn, designated him as the right person for the job – the development leader told us. Not only he was speaking several languages and had lived on three different continents before joining the team, but he was also able to give clear explanations and react appropriately in tense situations.

He was given time to familiarise himself with the tools and the procedures and to contact the corporate team involved in the translation process, whose members were located in different places around the world. He was in an uncomfortable situation: he had to become the team’s only specialist in a specific task, without having anyone around he could learn from. “It’s more like learning by doing”-he told us, “there’s plenty of stuff out there (on the company’s intranet), but information is difficult to find”. The solution? Networking! “If you know who knows something – they point you to the right resources! And nobody ever turned me down when I asked for information!”

The strings belonging to the user interface (including help files and documentation) had to be separated from the actual code and sent to the translation team 10 weeks in advance. In order to understand the new product and make sense of the context of the text strings that needed to be translated, the individual translators were given a prototype of the application and the corresponding documentation. During that 10 weeks period, the translation coordinator had to dispatch all the requests coming from various translators regarding specific strings. When a translator had problems understanding a specific situation when a particular text had to be displayed on the screen, the coordinator tried to help him and discussed the context via instant messaging. Sometimes, he had to involve in the instant messenger conversation the specific developer who had written that specific part of the code, to save time and give an immediate answer to the translator who could continue his work after that. The alternative would have been to check the code repository and try to figure out the context himself, but obviously this would have required more time. The working time overlap between the team in Ireland and the various translators was also, in some cases, very short. The developer who had written the code was the most knowledgeable about specific occurrences of a string being displayed on the screen; for him, it was almost obvious. But taken out of context, it was difficult for the translator to make sense of it and find the right equivalent. The translation coordinator facilitated the explanation on the context and the knowledge transfer to the translator by bringing together the developer and the translator in an ad-hoc instant messenger conversation. Not only did he learn the answer to specific questions (usually the ambiguous messages were rapidly detected), but he also retained these answers for future use in his instant messenger archive.

This case is interesting because it illustrates how a team member has temporary become a central hub, connecting the members of his own team with the members of the distributed localisation team and facilitating the transfer of contextual knowledge related to string translation. During this process, the coordinator also learned, both about the localisation process and about the product under development. This enabled him to answer some of the questions himself. The instant messenger was used for solving the problems in real time, and its archives were used as an ad-hoc repository, capturing the content of past conversations.

4.3 Inventing a new role: The integrator

The early stages of the project were spent exploring different alternatives and developing mock-ups. After a second software architect joined the team on site, the development efforts became more systematic. At the same time, the top management decision to integrate the software product developed by the team in a bigger package that had to be released in July 2006 accelerated the pace of the development efforts. Until then, the developers had worked on their specific components and tested them separately. At that moment in time, the modules needed to be integrated and tested together on the underlying software platform, still under development itself and unstable. One of the developers volunteered to take up the integration task, in addition to the development of a module.

He had worked with the team for two years, and had a few years of experience in customer support in the same company. Some of his more experienced colleagues were expected to do this – the development leader told us, but he volunteered and in a short while, his role became a quasi-official one. “I am the team’s integrator” he introduced himself. “And what we’re doing here looks like building a house using bricks that are not yet ready “.

“There was never such a role on any of my teams before” – the development leader said, “it was just a task – but he added a new dimension to it, taking responsibility for the result and doing his best to solve the problems”.

The “integrator” was the one to speak to all the other team members, becoming aware of the gaps, problems and critical issues. Not only that he was integrating the modules, but in a way, he was also a people connector. While working on isolated components, the team members didn’t have to interact a lot for professional reasons – and their social interactions were not restricted to team mates. The more the integrator spoke and listened to them, the more they were passing him information relevant to the others in the group. Soon, they started to interact much more with each other – not only because of the interdependencies between modules, but also because the integrator was now more aware about who knew what, and he was asking specific people to go and help their mates when incidents they once solved with success reoccurred.

In time, the integrator got to know not only the components of their application very well and the way they were supposed to function together, but also the underlying technologies, and – the most important thing - the people working on specific components, both inside and outside the team. 20 month into the project, the collocated architect, who until then was dealing with the external contacts, was assigned to another project. The integrator did his best to replace him in this role until another software architect joined the team and ensured the interface with the other teams they had to work with.

Then, a system testing phase that was supposed to end 3 months after the first release went on for another couple of weeks. The integrator was involved in the efforts of fixing complicated defects resulting from the coupling of different technologies, but his expertise was also required for the current development efforts. He realised he was becoming a bottleneck, and no matter how much he wanted to be everywhere and contribute, his time was limited. He initiated weekly meetings he named “knowledge transfer sessions”, where he presented topics related to the product under development and the underlying technologies and encouraged his colleagues to share their specific knowledge as well. When two new developers and an architect joined the team, he found time to give them a crash-course on the product itself and the technologies involved, for bringing them up-to-speed.

Around Christmas 2006, while he was on vacation abroad, a critical bug occurred. The development leader decided to ask for his advice, and the integrator had to communicate with another developer via phone on how to approach the problem; two days later, the problem was solved.

This last vignette highlights the contribution made by a developer with excellent technical and social skills to the timely delivery of the software product. As the coordinator in the previous example, he became a central hub, first inside his team, and later on at its interface with other teams. Through the integration process, he gradually learned about the various modules, their internal and external interfaces, the underlying technologies and the corporate procedures that had to be followed in the development and testing process. All his activity turned around the code repository, driven by product architecture, detected defects, corporate rules and procedures. His deep knowledge about the various code components - acquired through an extensive practice, together with his social skills and willingness to share, enabled him to support his mates, understand their knowledge gaps and initiate knowledge transfer.

5. Discussion

In the examples presented in the previous section, we described practices that were developed by the developers' team to help them cope with distribution. The company had, of course, recommended corporate processes for dealing with those specific situations. But, as Brown and Duguid show, "process deals with prescription and formality, whereas practice deals with all the variations and disorderliness of getting work accomplished. Process may be the ideal, but practices define how actual work gets done."(Brown and Duguid 2000)

Our goal was to look specifically at practices involving knowledge sharing, transfer and creation with an emphasis on the role of human actors. In the first vignette, the software architect has deep knowledge about the product to be developed, the opportunities offered by the new underlying technologies the product will be delivered with, the customers' needs and perspective. He has been involved in the development of 4 similar products, has visited customer sites, and was fully aware of the competition in this segment of the market. He went through the same process before, but in most of the cases worked with a collocated team, or collaborated at distance with a team he already knew after a period of collocated work. He was asked to do highly creative work under extreme time pressure, and as a consequence, he decided to assume the role of the seer who has to guide the blinds. Telling each developer what to do and how to do it and revising the results rather sooner than later became his way of dealing with the time constraints. Even if general presentations have taken place, there was not enough time to discuss the whole philosophy of the product with everyone; besides, the software architect himself didn't know precisely how the product was going to look like in the end. This case illustrates knowledge transfer and co-creation – a "chunk of knowledge" regarding the functionalities of a specific component was transferred – but the structural knowledge remained hidden to the developers until later. It was only during integration (and later on through the knowledge transfer sessions) that they become aware of their product as a whole. During the one-on-one meetings for component development, architect and developer built a shared understanding of specific components. Documenting is well known as being something software developers would do only if they consider it relevant for their work (Lethbridge et al 2003). In the example presented, documenting occurred after that shared understanding was made explicit. The people who documented the product functionalities were actually the direct beneficiaries of that specific documentation. Even if a whole range of tools, models and methods have been developed to help architects capturing the design rationale, they have failed to transfer to practice. Capturing assumptions and decisions doesn't appeal to the architect, because he is not the one who benefits from it (Kruchten 2006). This practice enabled the architect to hand over the documenting task to the developers.

In the first vignette, the developer was part of the architectural development process and was given the chance to build an understanding of both the emerging specifications and at least some of the reasons behind every specific decision. In the second example, once a specific context was explained to someone else in his presence, the translation coordinator was able to answer similar questions in the future without having to involve the developer. People who have spoken to the integrator in our third vignette about specific problems and managed to solve them with his support are later on capable of replacing him and help their colleagues in similar cases. In all these cases, the participants construct *mutual knowledge*. Mutual knowledge as defined by Cramton is "the knowledge that communicating parties share in common and know they share" (Cramton, 2001) and is considered to be a central problem of geographically dispersed collaboration. Distributed communication tends to get fragmented, and even with the information available to the whole group (on mailing lists or in project databases), it is actually difficult to retrieve it. *Boundary spanners* (Curtis 1988) such as the guru architect, the translation coordinator and the integrator in our three examples are the ones who maintain a big picture of the situation in its progress and of the information available, and are able to connect people to other people, and people to information. As the translation coordinator put it, the first and most difficult step is "to find someone who knows". Inside the team, people usually know where to turn to for getting specific information. Wegner has coined the term *transactive memory* for describing this phenomenon: after becoming part of a well-integrated transactive memory system, one can access information "well beyond his or her own internal storage" (Wegner 1986). More than that, information is naturally delivered by the group to the appropriate expert, contributing to an even more accentuated centralisation of information. What happened to the integrator when he understood he was becoming a bottleneck is also described by Wegner in terms of transactive memory: "Questions about the domain are typically directed to this person by default and it is sometimes difficult for the person to escape continuing responsibility for storage in the domain once expertise is generally acknowledged."

The remote architect knew who developed each component and how efficient they were at transposing his ideas into practice. Both the architect and the integrator had a feel of where the weak links were and which components or interfaces were likely to cause more problems than the others. This enabled them "to guesstimate" when were defects caused by architectural decisions or by the actual code, when a problem occurred in a component and when it was caused by the interface between the product components.

The translation coordinator in our second example had to deal with people belonging to different teams distributed globally and was able to dispatch the requests because he possessed relevant information about the team members who developed each component.

One interesting aspect is that the infrastructure enabling all these people to look for the relevant information, discuss alternatives and make quick decisions is a very common tool: instant messaging. What really makes a difference in this case is the organizational culture that encourages its use, providing every employee with a straightforward mechanism for accessing the right people at any level in the organisation and in any location, in the shortest time. Instant messaging was turned into a highly effective transactive memory system, enabling people to maintain lists of contacts grouped according to expertise, team or location. Instead of having to look for contact names in directories and email archives, once a contact had been found, it was stored in the instant messenger list, providing a shortcut for interacting with people who were infrequent or temporary collaborators (as in the case of translation and localisation).

In a distributed context, social relationships take a different dimension. In most cases, people do not get the chance to meet in person. The relationships are shaped by mediated interactions (participating in the same teleconferences, email exchanges, chats and phone calls), by peers' references and professional reputation. Communication skills and a reliable communication infrastructure are extremely important, as the situations described earlier illustrate. In the communication with remote colleagues, simple features like good will and common sense prove to be vital for collaboration in an environment where time constraints and tensions are present every day.

6. Conclusion

The purpose of our paper was to put into light a number of actual knowledge work practices through some particular examples of collaborative work over distance.

Much research is dedicated to building new collaborative tools, envisaging better processes, creating new frameworks, methods and models, and a much smaller amount is actually looking at the human actors struggling with the implementation of all these in practice. It is too often forgotten that software development happens in the real world and "suffers from all the variation and unpredictability associated with people, who have their individual strengths and weaknesses, insights and blind spots."(Dawson 2003).

We focused here principally on the role of human actors, of their values and social connections in dealing with the challenges of distributed work and getting the work done. The practices described in this paper were developed by the team as they carried out their normal workaday activities, but we can assume other teams might create their own practices around the same tasks.

By adopting an ethnographically-informed approach, we attempted to put on "new lenses through which to see the world" (Dourish 2006) and shed a different light on the relationship between technology and practice. In a distributed environment, it is difficult to separate the technical practices of developers in one location from those of the other people across the organisation with whom they interact, from whom they learn, and with whom they exchange people, information and artefacts. Some practices remain specific to one team, while others spread and are adopted across the global organisation.

Our final conclusion is that the decisive factor for the success of projects with distributed team members is the human one, supported (or inhibited) by the organizational culture. Tools, although important, play only a secondary role. The utility of knowledge repositories increases appreciably in the presence of peers who can guide people to access the most appropriate and up-to-date resources and can be simply nullified when people are reluctant, or refuse, to adopt them. For people working in a distributed environment, communication and social skills need to be taken into account together with the technical ones.

Software development is just one of the knowledge intensive industries expanding so quickly nowadays. Knowledge work has its own particularities, and the additional challenges brought in by global distribution require attention. Despite our current focus on global software development activities, we expect our findings to be of interest for people involved in distributed work arrangements in other domains as well.

Acknowledgements

This work is part of the socGSD project at the University of Limerick. socGSD is one of the LERO (the Irish Software Engineering Research Institute) cluster projects funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI). We wish to thank our informants in the observed company for allowing us to interview and live with them on site for many days.

References

Aspray W., Mayadas F., Vardi M.Y. (ed.) (2006) "Globalization and Offshoring of Software - A Report of the ACM Job Migration Task Force", *Association for Computing Machinery*, [online], http://www.acm.org/globalizationreport/

Blackler, F. (1995) "Knowledge, Knowledge Work and Organizations: An Overview and Interpretation", *Organization Studies*, Vol. 16, No. 6, pp. 1021-1046

Carmel, E., Agarwal, R.(2001) "Tactical Approaches for Alleviating Distance in Global Software Development", *IEEE Software,* IEEE, March/April 2001, pp.22-29.

Cramton, C. D. (2001) "The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration", *Organization Science*, Volume 12 , Issue 3 (May 2001), pp: 346 - 371

Curtis B., Krasner H., Iscoe N.(1988) "A field study of the software design process for large systems", *Communications of the ACM*, vol.31 no.11, p.1268-1287

Davenport, T. H. and Prusak, L., *Working Knowledge: How Organizations Manage What They Know*, Harvard Business School Press, Boston, MA, 1998.

Dawson, R.; Bones, P.; Oates, B.J.; Brereton, P.; Azuma, M.; Jackson, M.L.;(2003) "Empirical methodologies in software engineering", *Software Technology and Engineering Practice, Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'03),* Vol.00, 19-21 Sept., pp.:52 – 58

De Souza, K., Awazu Y., Baloh P.(2006) "Managing Knowledge in Global Software Development Efforts:Issues and Practices", *IEEE Software* ,Volume 23 , Issue 5 (September 2006), Pp: 30 - 37

Dourish, P. (2006) "Implications for Design", *Proc. ACM Conf. Human Factors in Computing Systems CHI 2006* (Montreal, Canada), pp. 541-550.

Hansen M.T., Nohria N., Tierney T.(1999) "What's Your Strategy for Managing Knowledge?", *Harvard Business Review,*vol.77, no.2, pp.106-116.

Herbsleb J.D., Moitra, D.(2001) "Global Software Development", *IEEE Software*, IEEE, March/April, pp.16-20.

Kruchten, P., Lago, P., van Vliet, H., (2006) "Building up and Reasoning about Architectural Knowledge", 2nd International Conference on the Quality of Software Architectures (QoSA), Vaesteras, Sweden, *LNCS 4214*, June 2006.

Lanubile F., Damian D., Oppenheimer H.L.(2003) "Global Software Development: Technical, Organizational and Social Challenges", *ACM SIGSOFT Software Engineering Notes*, Vol.28, No.6, November.

Lethbridge, T., Singer, J., Forward, A., (2003) "How Software Engineers Use Documentation: The State of the Practice", *IEEE Software,* Vol. 20 Issue 6, pp. 35-39

Prikladnicki, R., Audy, J.L.N., Evaristo, R. (2003) "Global Software Development in Practice: Lessons Learned", *Software Process Improvement and Practice*, John Wiley and Sons,Ltd., Vol. 8, Issue 4, pp.267-281.

Rus, I., Lindvall, M., Suman, S.S. (2001) "Knowledge Management in Software Engineering", *DACS State-of-the-Art Report*, Fraunhofer Center for Experimental Software Engineering and University of Maryland

Sarker S., Sahay S. (2003) "Understanding Virtual Team Development: An Interpretive Study". *Journal of the Association for Information Systems*, Vol. 4, pp. 1-38

Seely Brown, J., Duguid, P.(2000) "Capture Knowledge without killing it", *Harvard Business Review*, May-June

Wegner, D. M., (1986)."Transactive memory: A contemporary analysis of the group mind". In B. Mullen & G. R. Goethals (Eds.), *Theories of group behavior* (pp. 185-208). New York: Springer-Verlag.