Project Kick-off with Distributed Pair Programming

Edna Rosen, Stephan Salinger, and Christopher Oezbek

Institut für Informatik, Freie Universität Berlin
edna.rosen, stephan.salinger, christopher.oezbek@fu-berlin.de

Abstract *Background*: More and more software development companies decide to share their workload between teams which are geographically distributed. One of the biggest challenges is to start up work when new team members are introduced at a distant site of a global cooperation. Usually existing development processes do not cover integrating distributed collaboration, hence there is a need to adjust them to make project starts comfortable, easy and fast. A field study was conducted to introduce distributed pair programming (DPP), a derivative of pair programming (PP) in a distributed context, as a new development method to support communication and enhance knowledge transfer right from the beginning of the project. *Objective*: The objective of the study was to uncover relevant procedures and problems of establishing DPP and to collect supporting procedure steps for future project starts in distributed collaborations. *Methods*: A variation of canonical action research (CAR) was used to both establish DPP, gather insights and allow feedback from the developers involved. *Results*: This paper describes the establishment of DPP in a corporate project kick-off. It also reveals some benefits and major problems about distributed collaboration like conflicts in role fulfillment, ambiguity about session goals and missing awareness. *Limitations*: The validity of this study is threatened by the small number of participants and their particular cultural backgrounds.

Keywords: POP-I.A. distributed collaboration, POP-I.B. transfer of competence, POP-II.A. novice/ expert, POP-II.B. coding, POP-V.B. field study

1 Introduction

Software development in the twenty-first century cannot avoid the effects of globalization on production. One of the biggest challenges for distributed software development is to make knowledge available at all necessary locations quickly and efficiently (Braithwaite & Joyce, 2005; Herbsleb & Mockus, 2003). This becomes even more important if distributed collaboration separates the domain experts from newly assigned developers.

To enhance communication and knowledge transfer between stakeholders, a development practice like pair programming (PP) may be introduced for project kick-offs. Usually PP is part of other agile software development practices which are combined to a whole development process called extreme programming (XP) (Beck, 1999). Nevertheless it is also possible to introduce PP as a single new development practice without changing existing development processes (Aveling, 2004). In PP, two programmers work jointly while only using one computer, mouse and keyboard. The developers regularly change between two roles (Williams et al., 2000): One developer is taking the role of the 'driver', controlling the equipment, while the other developer, the 'observer', follows what the former is doing. Although engaging two developers with one task seems to be a lot of additional effort (e.g. Hannay et al., 2009; Nosek, 1998), PP has shown to increase communication and knowledge transfer between team members and to produce code of higher quality (e.g. Bipp et al., 2008; Hannay et al., 2009).

Due to newest technologies it is possible to perform PP in a distributed context, then called distributed pair programming (DPP) (Baheti et al., 2002; Stotts et al., 2003). DPP is similar to PP in that developers are joined (albeit virtually) to collaborate on a given task, but different in that co-developers each have their own computer, keyboard and mouse, which allows them to also work independently. With this advantage though, new challenges arise: non-verbal communication is limited and most actions of the co-developers are not instantly visible (Gutwin & Greenberg, 1999; Hanks, 2008). To bridge this, developer's actions must be made noticeable by awareness functionalities, e.g. code highlighting (Salinger et al., 2010).

In cooperation with the German IT companies Teles AG and her holding SSBG a field study was conducted to establish DPP as an additional development practice for a project kick-off between developers with little to no experience in DPP or PP (an expert in Vienna and two developers at a new office in Bangalore). At each of their local offices the developers still worked integrated in their local teams using a waterfall-based development process.

This paper delineates the establishment of DPP to support a corporate project kick-off. Section 2 highlights what is important about distributed software development and the establishment of a new development practice. Section 3 discusses the research setting including research background, research method and offers a short description of the technical infrastructure. Section 4 presents the results, lessons learned, and an overview of the most significant problems which occurred. Finally, Section 5 contains a conclusion of the establishment process.

2 Related Work

Since the rise of the Internet the software development industry has shown interest in distributed collaboration (Olson & Olson, 2000). Several scientific studies and industrial experience reports have dealt with the desire to optimize global cooperation (e.g. Damian & Lanubile, 2004), offering essential reasons which outline the necessity of distributed collaboration. Poole (2004) and Bass et al. (2007) refer to outsourcing, a desire to employ best available developers from any location, growing global open source communities as well as economic necessity such as cost competitiveness or product strategy, i.e. addressing specific market requirements. Yap (2005) additionally states sharing results and knowledge between locations as a reason for distributed collaboration.

Most of the studies and experience reports agree that there are three primary aspects for successful distributed collaboration (e.g. Poole, 2004):

First, the best applicable practices according to the needs of the development process have to be chosen. Distributed development practices like DPP or loosely coupled development methods such as distributed party programming (Salinger et al., 2010) allow the establishment of single practices in addition to existing development processes, while the establishment of distributed extreme programming (DXP) changes the entire development process to XP in a distributed context. Choosing the best development practices also depends on who will be involved in the distributed collaboration. Some companies may want to change the development practices to create a whole new distributed team from different locations (Yap, 2005), whereas others only bring together experts and newbies temporarily when necessary (Bass et al., 2007; Schümmer & Lukosch, 2008).

Second, the distributed process has to be adapted to integrate into an existing organization (Cohn & Ford, 2003). To this end different perspectives have to be assumed, for instance from the developers, management or other departments involved. Also cultural, psychological and social aspects need to be considered (e.g. Bass et al., 2007; Canfora et al., 2003).

Third, a technical infrastructure must be established. Such infrastructure includes the developing environment, collaboration tools, audio or video connection (Stotts et al., 2003). Individual tool preferences, platform restrictions as well as resource constraints, e.g. available bandwidth, should be considered (Schümmer & Lukosch, 2008).

In the field study conducted, DPP was temporarily established as an additional development practice to transfer expertise from one company site to a new team at a different site. Using DPP as a temporary practice enabled to focus on the developer's needs and establish and improve the technical infrastructure. Cultural, psychological and social aspects were only considered in case they could be attributed to experiences from existing studies.

3 Research Setting

3.1 Project Background

The project emerged as a cooperation between the Institute of Computer Science at Freie Universität Berlin and the German IT companies Teles AG and her holding SSBG. The IT company wanted to kick

off a project between an office in Vienna and a new office in Bangalore and looked for a cost-effective and fast alternative to bring together their domain and programming expert from Vienna with the two newbies (experienced developers new to the company) from Bangalore. They wanted to transfer expertise from Vienna to Bangalore without having to change local development processes. To support the IT company in their distributed collaboration, the researchers provided a complete technical infrastructure including voice communication and a tool for DPP (see Section 3.3). This way the developers could easily work jointly and concurrently on their project and otherwise remain integrated in their local teams with individual tasks for each developer and a mostly sequential work flow. To support the developers in adopting DPP as a distributed development process and support the project kick-off, one researcher assumed the role of a process coach.

The main goal of the project kick-off was to develop a prototype based on a provisional specification of a voice recording solution for a VoIP application. To this end, the expert was expected to transfer his knowledge about the domain to his team colleagues at Bangalore and ensure a high level of understanding of the produced software artifacts and its interaction with existing parts for all participants. After the project kick-off, the new developers from Bangalore were supposed to be able to implement and maintain the project on their own.

To achieve these goals, the domain expert in cooperation with the researchers devised the following development process: First, tasks would be assigned by the expert mainly using the existing prescription from the waterfall-based model the company used, i.e. the new developers would receive tasks to implement independent components based on the provisional specification. Additionally critical parts of the project were developed by the expert to show and explain existing coding regulations. Second, it was decided to conduct regular DPP sessions of roughly two hours in length depending on the needs of the development process (which the expert decided to be once a week). Two different session types were envisioned by the expert: (1) The new developers would be asked to perform a code walkthrough (Freedman & Weinberg, 2000) of the code they had written so the expert could assess their progress and knowledge gains. Each developer was expected to present the code and mention critical and questionable points, giving the expert opportunity to comment on his interpretation of the specification in case of deviations. (2) The expert wanted to pair-develop (Williams & Kessler, 2000; Williams et al., 2000) the software, i.e. jointly create or edit artifacts of the software. The goal of this second type was to explain critical parts of the software to the new developers and provide opportunities for asking questions.

Starting in April 2009, sixteen weekly sessions using DPP were conducted over a period of four months. The expert participated in all these sessions, one of the new developers participated in three, the other in fifteen and both of them together in two sessions. The DPP setting was exploratively extended to three participants to test further benefits of DPP compared to co-located PP (as described by Salinger et al. (2010), the technical infrastructure deployed also allowed more than two participants). This was dropped by the developers not seeing any further advantage compared to the additional effort of a third developer involved. As one of the newbies was mainly involved in other tasks outside DPP sessions, he only participated a few times and later on information about the common project was transferred to him by the other newbie.

3.2 Research Method

The field study was conducted following principles from canonical action research (CAR) as described by Davison et al. (2004). Most important about this approach is that the researcher (in this case the process coach) and the participants are cooperating tightly and the direction of the collaboration can be influenced by either of these parties. This rather explorative research process is structured through several principles. It is based on an iterative process model (see Figure 1). With each iteration new insights are collected and interventions are planned to optimize the ongoing process. Process steps and iteration length can vary depending on individual demands of a particular research process.

One of the main goals of the researcher was to establish DPP according to the needs of the developers and overall goal of the project. This afforded to look at it from a researcher's perspective, i.e. the practicability of DPP in general as well as from the developer's point of view, i.e. the success of the

planned development project. For initial background information about the developers involved, these completed a preliminary questionnaire about their experience with and expectations of DPP. Then the process started with a researcher-developer agreement, which continuously provided a general direction for the research process. In it the developers involved agreed upon project goals to be achieved, e.g. to create a prototype, and on the overall structure of the research process, e.g. when to meet for DPP sessions.

This initiated a cyclic process which contained three main steps (illustrated by the process model in Figure 1). The first step in each cycle was to determine the actual state of DPP usage with a focus on occurring problems, the project status regarding the overall project goals and the fulfillment of specific session goals. Hence before each upcoming session the developers completed a questionnaire stating their planned tasks, e.g. scope and duration, and what benefit or difficulties they expected. After each session they completed another questionnaire to evaluate the success of the session, i.e. whether their session goals were achieved and if expected benefits or difficulties emerged. Additionally, the process coach participated in each session and gathered information through observation and by analyzing log-files and videos recorded during the session post-hoc. Last, "reflection meetings" were held after each session to let developers discuss their experiences with DPP and to support the analysis with first-hand impressions.

In the second phase of each cycle (ideal state analysis) the previously identified problems or other phenomena such as last minute changes to planned tasks were analyzed to improve the use of DPP. This was done by the researchers using scientific literature and regular discussion. Afterwards the results were discussed with the developers in one of the following reflection meetings or sometimes in individual interviews, e.g. if only one developer was affected. Finally, new or adjusted process goals were evaluated and set based on the insights of the actual state to approach a more ideal state.

The third and last phase of each cycle consisted of planning and performing interventions according to the results of the ideal state analysis, e.g. changing audio settings for better quality.

At the end of the project kick-off the developers were asked to state their final impression of DPP, their experience with it, the overall project success, as well as anything else about DPP they found remarkable in a last individual interview.

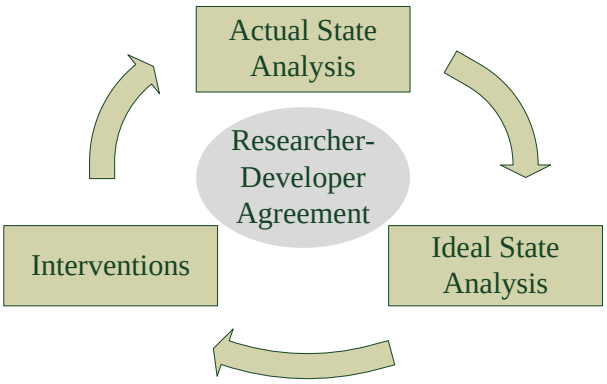

Figure 1. Cyclic research process model with three main phases based on a researcher-developer agreement

In summary, the following data sources were used to collect information about the DPP process:

- Observation: One of the researchers participated as the process coach in every DPP session.
- Preliminary questionnaire: One questionnaire was administered to the developers before the cooperation started to gather background information such as their level of experience with PP.
- Questionnaires before and after each DPP session: Each session was accompanied by a questionnaire about the session.

– Reflection meetings after each session or at least once a week: The developers and the process coach reflected together about the most recent session and discussed possible interventions for future sessions.
– Individual interviews: In total three individual interviews with the domain expert were conducted over the course of the collaboration.
– Final individual interviews: At the end of the project an interview with open questions about the overall success of establishing DPP was conducted.

3.3 Technical Infrastructure and Collaboration Tool

The different software packages necessary for conducting distributed pair programming were provided by the researchers and installed on a company server and on the computers of all participants. The packages consisted of the collaboration tool Saros[1] (Salinger et al., 2010), which integrates in the development environment Eclipse[2], the VoIP application Mumble and its server component Murmur[3], the instant messaging server OpenFire[4] and virtual private network client and server via OpenVPN[5].

The most central component in this setup is the collaboration tool Saros, which lets multiple developers work collaboratively in the development environment. Saros has been developed at Freie Universität Berlin by a team of students since 2006 (Salinger et al., 2010). To bring developers together for DPP, the software defines the concept of a session to which one participant in the role of the host can invite any number of participants as clients. The software then allows to share a software development project between all participants or synchronize existing copies to match the version provided by the host. During programming Saros closely models the roles of driver and observer known from PP by granting write access only to the driver and allowing the observer to follow the movement in files and package of a driver.

To increase awareness about the activities of the remote peers during a session, Saros also highlights the cursor, text selection, written text, visible viewport in a file, and the opened files in the project explorer. An annotated screenshot can be seen in Figure 2 presenting these options.

During the study all sessions were recorded on video both for scientific analysis and to improve Saros in combination with input from the participants and log-files generated by Saros.

4 Results and Lessons Learned

4.1 Session Overview

The first two of 16 sessions were used by the coach to explain the research process, introduce basic terms and processes of DPP and to show and explain the technical infrastructure to the developers. The following sessions mainly involved code walkthrough (sessions 3,7 and 9) and pair-developing including ad-hoc testing (sessions 4-6, 8 and 10-16).

In the first few research process cycles primarily a lot of adjustments in the technical infrastructure were needed, such as adjustments to audio settings and equipment to improve audio quality. Since Saros had never been deployed in an industrial scenario between continents before, it was necessary to find workarounds and software updates had to overcome several problems such as improving the synchronization of large projects.

After the fifth session a fixed starting time and duration of 90 minutes per session was set. Before, the expert had chosen a starting time and duration according to his planned task, which did not take into consideration any delays caused by developers not being on time, the synchronization of their large project, or answering general questions. Figure 3 gives an overview of the time spent in each session

[1] Available for download at `http://dpp.sourceforge.net/`.
[2] `http://www.eclipse.org`
[3] `http://mumble.sourceforge.net/`
[4] `http://www.igniterealtime.org/projects/openfire/`
[5] `http://openvpn.net/`

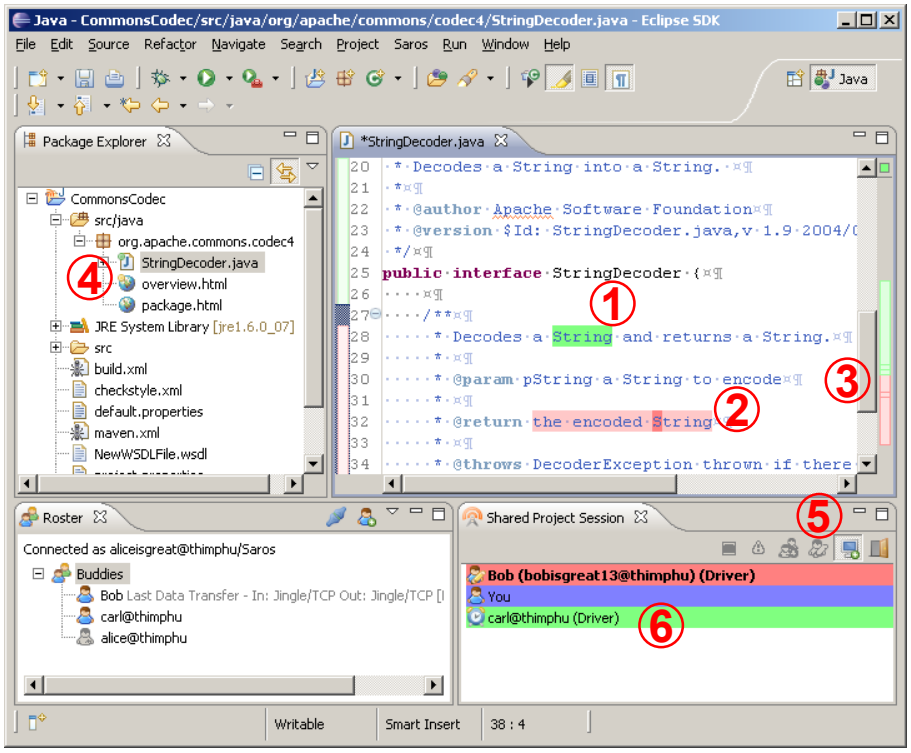

Figure 2. Various awareness features used in Saros such as (1) selection, (2) text edits, and (3) viewports highlighted in each users' color, (4) opened and active files by current drivers, (5) button for following the viewport of a driver, and (6) information about Eclipse being the foreground window.

and shows that preparation time declined as the developers became more experienced (with the notable exception of session 14 in which technical problems prevented any development to be started, but still allowed the developers to discuss changed requirements and to plan ongoing work).

During later research process cycles the focus then could be shifted to improving the use of DPP (described in the following sections) and on optimizing the research method, e.g. finding an ideal time for reflection meetings, and improving communication between developers. Planning reflection meetings was demanding because it was necessary to get all participants together and not let too much time go by after the sessions to be reflected. Sometimes only a few days after a session the developers would not remember what had happened in their last session. Finally, performing one reflection meeting after the last session of the week showed to be most effective.

4.2 Benefits of DPP

In twelve of the thirteen post-session questionnaires the developers declared the session a success and stated that most of the times DPP had been helpful to achieve their session goals (the expert agreed for 85% of sessions, the newbies for 89%). In the final interview they confirmed that their project goals were achieved.

Over the course of the study the following three benefits appeared to most prominently support the use of DPP:

– First of all, communication between developers was enhanced noticeably throughout the collaboration. Before the DPP sessions started, communication was limited to chat or e-mail. Due to DPP sessions and reflection meetings the developers talked to each other at least once a week for more than one hour. Moreover, communication was enhanced due to newly introduced walkthroughs and pair-developing in DPP sessions which were supplementary to their local, usually rather loosely coupled, development process. The developers used the session time to ask questions or discuss open

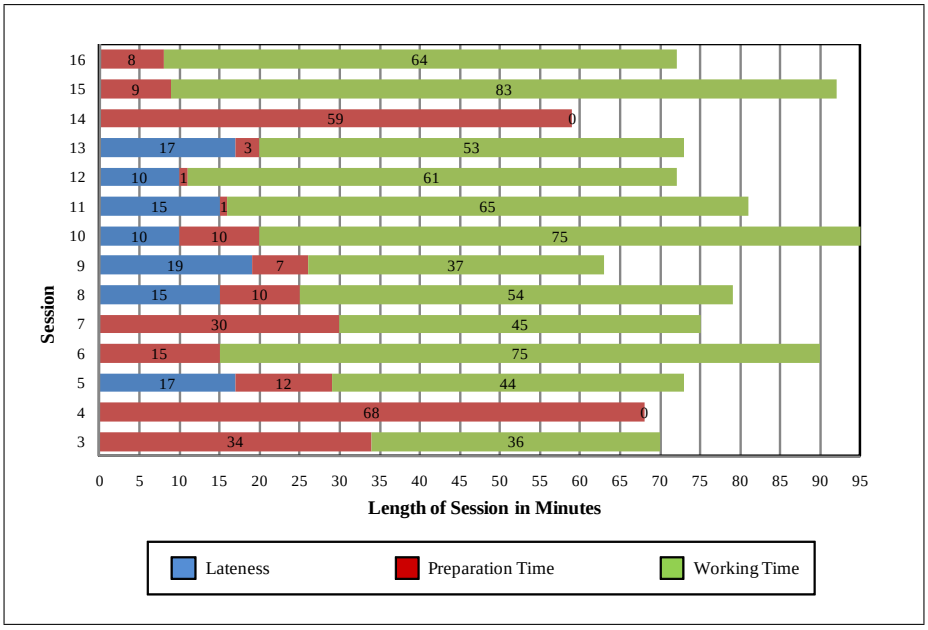

Figure 3. Length of DPP sessions as a breakdown of time lost due to developers coming late and having to prepare for the session, and time spent on productive work.

issues with the respective artifacts in plain view. In reflection meetings the developers also talked about the establishment of DPP in general, i.e. their experience and expectations, or discussed procedural strategies. In earlier sessions the driver, who was most of the times the expert, talked more than the observer. Nevertheless, between tasks or during project synchronization the developers used the pauses to reorder assignments or agree on general procedures. Later on in the collaboration the new developers integrated themselves more actively in sessions by making suggestions or discussing development issues.

- Second, the expert stated in eight of thirteen questionnaires after sessions that he transferred all important information to his co-developer. This was either in sessions where he was pair-developing or in walkthroughs of code previously created by one of the newbies outside the session giving feedback on requirements of the provisional specification. In most sessions the expert continued to write code simultaneously to explaining and teaching his partners, thereby showing a second benefit of DPP as combining teaching with productive work.
- Third, one of the newbies who presented his code in one of the walkthrough sessions stated in the questionnaire after the session that the task was accomplished faster than expected because there was no delay in the feedback. In the reflection meeting that followed he stated that feedback without DPP (through e-mail and chat) would have been much more complicated in comparison. Another benefit of DPP was observed in the sessions when errors of the driver could be avoided. In at least five of the pair-developing sessions the observer made the driver aware of errors and thereby avoid them, e.g. when the driver was about to write code in the wrong artifact or when the declaration of a variable he was about to add already existed.

4.3 Problems with Establishing DPP

Three of the problems that occurred in the establishment process could not be resolved even after interventions. These were (1) conflicts with role fulfillment, (2) ambiguity about session goals, and (3) missing awareness, which will be discussed in turn. The fact that development happened distributedly possibly influenced all of these problems. Since the developers did not have common office hours or did not have the possiblity to work co-locatedly might have had intensified these problems. Especially since the expert stated that he usually waits until common lunch breaks to discuss issues with colleagues, which did not happen in this project due to the distance.

Conflicts with Role Fulfillment A first problem arose in the context of assuming the roles of driver and observer, which in DPP—as in PP—stipulate the responsibilities of each developer during a joint session. For instance, XP recommends the driver to attend to details during programming and the observer to keep the top-level concerns in mind (Beck, 1999). Nevertheless role assignment in DPP as well as PP is not self-explanatory and hence can be challenging (Bryant et al., 2008).

The first indication of a problem with role assignment and fulfillment appeared in one of the first pair-developing sessions when the expert stated in the questionnaire after the session that he had the driver role for too long. As it is not unusual for an expert to assume the driver role and keep it over longer times than usually recommended by XP (Schümmer & Lukosch, 2008), this point was brought up in an interview with the expert to explore the reasons for his statement. The expert explained that he considered more regular role switching important to achieve the benefits of DPP. In particular he noted that he felt very exhausted from being driver most of the time and had had little opportunity to assess the knowledge gains of the new developers.

In the next pair-developing session the expert tried to hand over the driver role to one of the newbies. Yet, the newbies did not react to his request to assume the driver role and the expert continued being driver. The developers made no attempt to talk about this in the next reflection meeting. Hence the coach confronted the new developers with the issue. The newbies stated that from their point of view they did not have enough knowledge about the artifacts and did not feel comfortable being observed when taking over the driver role. Although the expert had expressed his wish for more role changes during sessions, the developers did not want to set up more formal rules for role fulfillment. Instead they agreed on finding spontaneous solutions on occasions in the session when necessary. In later pair-developing sessions it was observable that the newbies could integrate themselves more actively in the sessions by asking questions or making suggestions for code changes, although most of the times they still refused to take the driver role. Additionally, the final interview with one of the newbies revealed that he still had not understood how a second developer could integrate himself actively in a DPP session and that for him the observer role was mainly boring.

In a qualitative analysis based on the data collected through observations in the sessions, questionnaires, personal interviews, reflection meetings, and a discussion with the developers in the final interview it was attempted to build a conceptual model around the role behaviors following the paradigm model of Strauss & Corbin (1990) to distinguish causal conditions, intervening conditions and consequences.

Three primary causal conditions for the problems with role fulfillment could be identified from the data (see Figure 4): (1) The developers had *little experience with DPP and PP* and thus did not know what the distinct responsibilities of the roles were, when role changes would be best suitable and what benefits would arise from following the prescriptions of a process model such as XP on role segmentation or responsibilities. (2) The developers showed *little role consciousness* during the sessions, but often only mentioned problems when explicitly queried after the session. Thus instead of resolving conflicts during the sessions, the reflection meetings were necessary to raise the awareness of the developers on these issues. (3) *Diverging expectations* between the expert and the new developers caused them to require or reject different aspects of the roles. While the expert wanted to transfer the driver role to the new developers so they would practice writing code under his support, the new developers rather felt under observation and thus feared being caught making a mistake.

These causes are certainly dependent on each other—increased DPP experience in particular should both align expectations and raise consciousness about role mismatches—but sufficiently different in the way they could be addressed to be given separately here. Several other minor causes could be identified, but none of them had sufficient explanatory value.

As an intervening condition, pair-pressure (Williams, 2000) was identified. Usually such pressure is welcomed by the developers in a PP session because it increases concentration and helps developers push each other to complete their task (Williams, 2000). Unfortunately the developers from Bangalore could not benefit from this pressure, but rather their expectations diverged further and their consciousness

about the development process gave way to feeling uncomfortable to be observed while coding in a domain in which they were not experienced.

The consequence of suboptimal role usage is primarily to be seen (1) in *less productive sessions*, in which less code was produced and less knowledge transferred, (2) *boredom* and (3) *exhaustion* on the side of the new developers and the expert respectively.

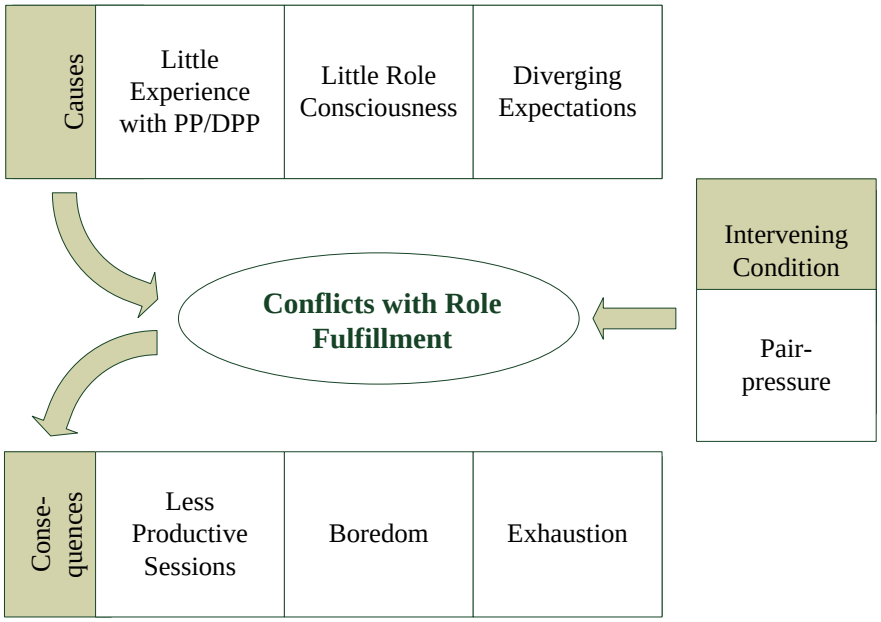

Figure 4. Possible causes, consequences and intervening conditions of conflicts in role fulfillment.

Lesson learned: Although problems in role fulfillment were identified in an early stage of the establishment of DPP and discussed in several reflection meetings, they could not be resolved. It seems as if it was not enough to know and talk about problems with role conflicts in reflection meetings, if the developers involved are not conscious enough about their conduct during the sessions. Evidently it happens that participants do not discover or experience the benefits of a new practice. Therefore it is suggested to try to improve knowledge about PP and DPP topics in particular before and during the process of establishing DPP. Moreover, it should not be ignored that the success of role fulfillment in DPP also depends on the fears and attitudes of the participating developers.

Ambiguity about Session Goals A second problematic phenomenon associated with DPP could be traced to the goals associated with individual sessions and is best explained with one particular session early on in the project: For this session the expert had scheduled a “code review” with one developer and his newly written code. In the questionnaire before the session the expert clarified his goal for the session as “code review presented by [name of newbie] followed by a discussion”, while the new developer stated as the goal “get the code reviewed by [name of expert] and discuss the open issues if any”. Thus by using the unqualified term “code review” when scheduling the meeting, the expert had inadvertently introduced ambiguity into the session goals, which led the new developer to believe that the expert would be primarily responsible for conducting the review. When the expert at the beginning of the session then asked the new developer to start presenting his code, the new developer silently concurred with the request, but obviously was not prepared for this task: First, he stated that he found it hard to find a good position in the code to start with. Then, a lot of times during the presentation he had to jump back and forth between different artifacts, his explanations were sometimes halting and several times he stated in the middle of an explanation that he forgot to mention some precondition.

After the session the expert stated in his questionnaire that the task was completed slower than expected. He attributed this to the large amount of code to be reviewed, ignoring the halting pace of the presentation, maybe not even aware that the newbie had been surprised by the expert's interpretation of the session goal and unprepared for a code presentation.

The next session designated as a code review then revealed how much time had been wasted by the ambiguous statement of performing a code review in the first session. Here the new developer had adjusted to the expert's interpretation of the goal and was excellently prepared to present the code.

Analyzing all DPP sessions which occurred during the project revealed that problems associated with ambiguous goals were more prevalent than this single example. Conceptualizing these incidents resulted in the following three causes to be identified for ambiguous session goals (see Figure 5):

The first cause of ambiguity in session goals, exemplified by the above example, was the *lack of precise communication*. In the above case it would have been sufficient to either describe in a single short sentence what a code review would entail or use the term walkthrough, which usually connotates the author to present the code (Freedman & Weinberg, 2000).

The second cause identified was the existence of *diverging project goals*. While the expert wanted to transfer much of his applicable domain expertise over the duration of the project, a goal to receive such domain knowledge was never mentioned by the newbies. The newbies' primary goal as stated both in the initial questionnaire and the final interview was rather to create code that was executable. This caused for instance a newbie to state after session 14, in which technical problems kept the developers from coding, that the session was not successful, although the expert stated it was a partial success because important questions were discussed and knowledge transfer had taken place.

The third reason why session goals often were ambiguous was session planning was conducted on *short notice*. In most cases the expert contacted the newbies only about one hour before the session to announce what the planned task would be. Since these announcements came in just before the newbies' lunch break (due to different time zones), this shortened their possible preparation time to zero. In an individual interview the expert stated that announcements were made on short notice after considering the latest status of the development process. He stated that he would not change this, ignoring the hint that this might allow the newbies more time for preparation. The newbies thus remained in a position in which the session goals were unknown to them until the very last moment.

Two intervening conditions could be identified to affect ambiguity in session goals: (1) The use of *instant messaging chat* aggravated the misunderstandings arising from the *lack of precise communication*, in particular because chat messages are more terse than voice or e-mail communication, are not persistently stored, and lack sufficient detail. (2) The emphasize of *flexibility over structure* within a session further amplified the ambiguity of session goals. For instance, the developers once performed lengthy ad-hoc testing of newly written code in a pair-coding session and thereby introducing a newly emergent goal of increasing quality to the existing goal of producing code for a certain feature.

The consequences of ambiguous goals ultimately were *wasted time and resources*—as the difference between the unprepared and slow walkthrough and the improved second session shows— and *inadequate session results*. Yet, more practically the lack of clarity about session goals ahead of time caused (1) *incorrect and insufficient preparation* for sessions by developers, and (2) led to an *invalidation of contributions* (the expert's lengthy explanations for instance seem a waste of time under the developers' assumption that the goal is primarily to produce code as fast as possible).

Lessons learned: Misunderstandings leading to ambiguity in session goals cannot be completely avoided. Nevertheless a short preparation sufficiently ahead of time to align the understanding of session goals, in particular who is in charge of what, can economize session time. Better aligned session goals are also more likely to increase satisfaction with a session as the contributions of all participants will integrate better and can be more easily valued by all parties.

Missing Awareness It is well known that shared awareness is a crucial element of distributed collaboration (Gutwin & Greenberg, 1999; Olson & Olson, 2000), where awareness describes the consciousness about one's own and the other participants' actions in the context of the collaborative environment

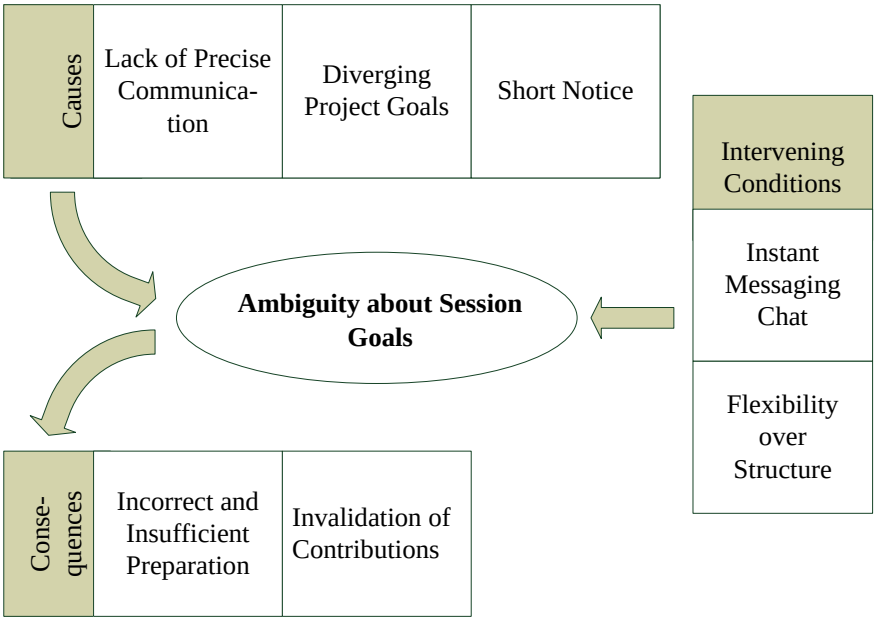

Figure 5. Possible causes, consequences and intervening conditions of ambiguity in session goals

(Dourish & Bellotti, 1992). Awareness can be challenging to attain at the beginning of a project when co-developers have never worked together and there is no common understanding of development strategies yet. Enhancing awareness may have a positive effect on distributed collaboration (Gutwin et al., 1996).

In DPP sessions awareness can be provided by the technical infrastructure as well as by the co-developers. Gutwin et al. (1996) have made an attempt to divide awareness into different categories such as technical awareness, workspace awareness, or social awareness (Gutwin et al., 1996). Workspace awareness covers all types of awareness which help the developers to find the location and actions of other co-developers in the developing environment. Awareness of the categories social or group-structural awareness cover social interactions such as expectations and abilities, e.g. if developers are aware of what other developers expect them to do next. Identifying the latter can be difficult, because it depends on the motives of the developers involved which are not always known. In the analysis of problems with role fulfillment mentioned above the motives of the newbies to not take over the driver role were questionable in that matter. It was not verifiable whether the newbies did not know what was expected of them or if they just pretended not to know. In the first case it would be a matter of awareness, in the latter merely ignorance. Although other forms of awareness were an issue in the field study conducted, this subsection will focus on missing awareness in terms of workspace awareness.

Three causes for missing workspace awareness could be identified by analyzing the collected data (see Figure 6):

(1) *Weaknesses in the technical infrastructure*: The technical infrastructure deployed in the field study and in particular Saros had never been analyzed in commercial software development before and did not include a video connection showing the remote partner's face, following advice from other studies (e.g. Baheti et al., 2002) in which developers had stated that the video connection had not resulted in additional benefit.

Analyzing the events of the DPP sessions showed that a video connection would have been favorable. Missing non-verbal communication (in distributed collaboration made visible through a video connection) was one of the reasons why actions of the observer were not always detectable in sessions. In earlier pair-developing sessions during the establishment of DPP in which the expert was the driver throughout the whole session, long phases were observed in which the observer's actions were not detectable. Sometimes background noises in sessions suggested that the newbie was distracted, e.g. a mobile phone rang or laughter could be heard over his microphone. On rare occasions one of the newbies noticeably started talking to one of his co-located colleagues, even provoking the expert to comment on it.

The problem of possible distraction of the observer was discussed in the following reflection meeting. The newbie stated that he could not remember these episodes of the session. To get more information about the newbie's behavior during pair-developing sessions, without breaking the limits of bandwidth, it was planned to install an additional desktop sharing application between him and the coach. For different reasons the desktop sharing application was never installed. Motivated by the problem, the Saros developers then improved the collaboration tool to show whether the development environment was the foreground window (in contrast to a browser for instance). The expert did not comment on any behavior of the newbies concerning their distractions during sessions. However in later sessions he demanded more frequently the observer's attention. This was noticeable through the amount of questions he asked the observer, e.g. about the location of the observer in the workspace or "what would you do in this case?".

(2) The second cause identified for missing awareness was the plain *non-use of awareness functionalities* provided by the collaboration tool. This was noticeable in sessions when the observer asked the driver at what location the former was, instead of using one of the Saros options, e.g. activating the "automatic follow mode" or double-clicking on the remote partner's name in Saros. Even when the developers used the follow mode, they used line numbers for orientation in the code, e.g. when asking questions about the code: "here in line number 500 we have an event", instead of marking the code with their cursor and thus highlighting it for the co-developer. Neither regular updates about existing and new Saros functionalities nor pointing out their advantages, e.g. preventing delays in the session by using the automatic follow mode and hence not having to ask for positions of the co-developers, could convince the developers of using the awareness functionalities provided by Saros.

(3) A third reason for missing awareness was identified as *lack of talk-aloud*, which was identified in one of the rare occasion when one of the newbies became driver in a session: The code he was typing in this situation was not being displayed in the observer's view because of a technical problem. Additionally the driver had not verbalized his actions and hence made it impossible for the observer to be aware that the driver was about to start writing code at a particular position. Taken together, the technical problem and the *lack of talk-aloud* led to the consequence that it took five minutes until the technical problem was noticed. When the expert as the host of the session tried to remedy the technical problem, he inadvertently overwrote the text written by the new developer, unaware of the work the new developer had already invested.

One important intervening condition which increased the problem of missing awareness was the *lack of role changes* during the sessions. Since the new developers rarely received the driver role (only in half of the pair-developing sessions, but never for more than five minutes), they also had little experience with managing awareness from both perspectives. More troubling, they in general did not expect role changes, as one of them stated in the final interview, and therefore did pay less attention to the action of the driver in the workspace, since they felt sure that only a passive role would be required from them. The fact that there was some evidence of the observer being distracted from sessions and that the observer had to ask for the position of the driver several times during sessions also led to this conclusion.

Figure 6 shows the discovered conceptual relationships and several consequences of missing awareness in the DPP sessions.

Lessons learned: Any comment from driver as well as observer can enhance awareness in DPP sessions. As stated by Stotts et al. (2003), constant exchange of information about what developers can see and commenting on actions keeps attention higher and avoids missing awareness. As later sessions showed, this can be achieved by the driver demanding frequent feedback through questions from the observer and should be enhanced by frequent role changes. Additionally, desktop sharing and/or a video connection should be integrated in the technical infrastructure to counteract missing awareness and to achieve more detailed and faster feedback between co-developers (Schümmer & Lukosch, 2008). Motivated by the requirements of additional awareness functionalities in the collaboration tool, the implementation of a desktop sharing functionality for Saros was started shortly after the project kick-off (Salinger et al., 2010).

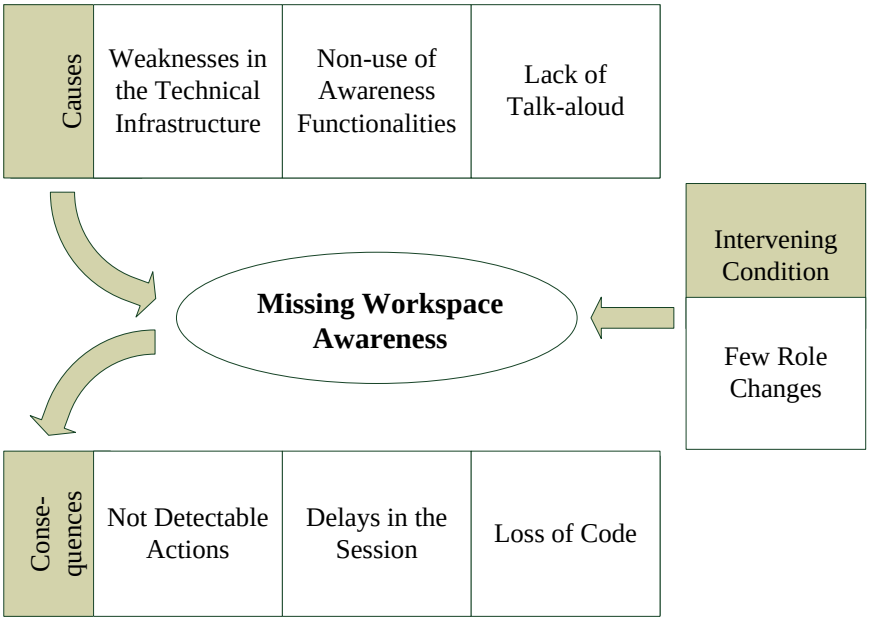

Figure 6. Possible causes, consequences and intervening conditions of missing workspace awareness

5 Conclusion

After sixteen DPP sessions the goal of the kick-off to develop a prototype was successfully accomplished within the given deadline and the development responsibilities were given exclusively to the Indian development team.

The field study conducted showed that DPP can be established and integrated into an existing development processes to support distributed collaboration in a project kick-off. The research method made it possible to constantly improve distributed collaboration and at the same time incorporate the developers' requirements. Frequent questionnaires and reflection meetings in combination with observations were essential sources of data collection and analysis. Constant improvement of the technical infrastructure according to the requirements of the development process, e.g. adjusting audio quality or introducing new awareness features of the collaboration tool, had a positive effect on the collaboration.

The analysis of the data collected confirmed already known insights, but also uncovered new problems about the establishment of DPP. Benefits such as a high level of communication, combining knowledge transfer and productive work as well as reduced delay in feedback supported the distributed collaboration and hence the project kick-off. Some of the lessons learned were that solving problems such as conflicts in role fulfillment, ambiguity in session goals and missing awareness can be challenging, if solving them is against the developers priorities.

For future work we are looking to replicate the study in other companies and distributed development settings to follow up first indications and other aspects such as inter-cultural influences about distributed collaborations.

Bibliography

Aveling, B. (2004). XP Lite considered harmful? In *Proceedings of the International Conference on Extreme programming and Agile Processes in Software Engineering (XP 2004), Garmisch-Partenkirchen*, volume 3092/2004 of *Lecture Notes in Computer Science*, (pp. 94–103)., Berlin / Heidelberg. Springer.

Baheti, P., Gehringer, E., & Stotts, D. (2002). Exploring the efficacy of Distributed Pair Programming. In *Extreme Programming and Agile Methods — XP/Agile Universe 2002*, volume 2418/2002 of *Lecture Notes in Computer Science* (pp. 387–410). Berlin / Heidelberg: Springer.

Baheti, P., Williams, D. L., Gehringer, E., & Stotts, D. (2002). Exploring pair programming in distributed object-oriented team projects. In *OOPSLA Educator's Symposium, Seattle, WA*.

Bass, M., Herbsleb, J. D., & Lescher, C. (2007). Collaboration in global software projects at siemens: An experience report. In *Proceedings of the International Conference on Global Software Engineering (ICGSE 2007)*, (pp. 33–39)., Washington, DC, USA. IEEE Computer Society.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Bipp, T., Lepper, A., & Schmedding, D. (2008). Pair programming in software development teams - an empirical study of its benefits. *Information and Software Technology*, *50*(3), 231–240.

Braithwaite, K. & Joyce, T. (2005). Xp expanded: Distributed extreme programming. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, (pp. 180–188)., Berlin / Heidelberg. Springer.

Bryant, S., Romero, P., & du Boulay, B. (2008). Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, *66*(7), 519–529.

Canfora, G., Cimitile, A., & Visaggio, C. A. (2003). Lessons learned about distributed pair programming: what are the knowledge needs to address? In *Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (pp. 314–319)., Los Alamitos, CA, USA. IEEE Computer Society.

Cohn, M. & Ford, D. (2003). Introducing an agile process to an organization. *Computer*, *36*(6), 74–78.

Damian, D. & Lanubile, F. (2004). The 3rd international workshop on global software development. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, (pp. 756–757)., Washington, DC, USA. IEEE Computer Society.

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, *14*(1), 65–86.

Dourish, P. & Bellotti, V. (1992). Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, (pp. 107–114)., New York, NY, USA. ACM.

Freedman, D. P. & Weinberg, G. M. (2000). *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*. New York, NY, USA: Dorset House Publishing Co., Inc.

Gutwin, C. & Greenberg, S. (1999). The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, *6*(3), 243–281.

Gutwin, C., Greenberg, S., & Roseman, M. (1996). Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. In *People and Computers XI*, (pp. 281–298)., Berlin / Heidelberg. Springer-Verlag.

Hanks, B. (2008). Empirical evaluation of distributed pair programming. *International Journal of Human-Computer Studies*, *66*(7), 530–544.

Hannay, J., Dybå, T., Arisholm, E., & Sjøberg, D. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, *51*(7), 1110–1122.

Herbsleb, J. D. & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, *29*(6), 481–494.

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM*, *41*(3), 105–108.

Olson, G. M. & Olson, J. S. (2000). Distance matters. *Human-Computer Interaction*, *15*(2), 139–178.

Poole, C. (2004). Distributed product development using extreme programming. In *Extreme Programming and Agile Processes in Software Engineering*, (pp. 60–67).

Salinger, S., Oezbek, C., Beecher, K., & Schenk, J. (2010). Saros: An Eclipse plug-in for distributed party programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. ACM. To appear.

Schümmer, T. & Lukosch, S. (2008). Supporting the social practices of Distributed Pair Programming. In *Groupware: Design, Implementation, and Use*, volume 5411/2008 of *Lecture Notes in Computer Science* (pp. 83–98). Berlin / Heidelberg: Springer.

Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., & Jackson, A. (2003). Virtual teaming: Experiments and experiences with Distributed Pair Programming. In *Extreme Programming and Agile Methods — XP/Agile Universe 2003*, volume 2753/2003 of *Lecture Notes in Computer Science* (pp. 129–141). Berlin / Heidelberg: Springer.

Strauss, A. L. & Corbin, J. M. (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, *17*(4), 19–25.

Williams, L. A. (2000). *The collaborative software process*^{SM}. PhD thesis, The University of Utah. Adviser-Kessler, Robert R.

Williams, L. A. & Kessler, R. R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*, *43*(5), 108–114.

Yap, M. (2005). Follow the sun: Distributed extreme programming development. *Agile Development Conference/Australasian Database Conference*, *0*, 218–224.