Industrially Usable Distributed Pair Programming

Dissertation
zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

Julia Schenk
Institut für Mathematik und Informatik
Freie Universität Berlin

Berlin
2018

Gutachter:

Professor Dr. Lutz Prechelt, Freie Universität Berlin, Deutschland
Professor Dr. Daniela Damian, University of Victoria, Kanada

Datum der Disputation: 11.07.2018

Für meine Mutter, meinen Vater und Till

–

For my Mum, my Dad, and Till

Contents

List of Figures

List of Tables

Acknowledgements / Danksagungen

When drinking water, remember its source.

Chinese proverb

My biggest thanks go to my family, in particular to my parents: Great to have you and thank you for your absolute support at all times.

Lutz Prechelt I want to thank for the challenging and promoting advice throughout this research project as well as for his confidence and granted freedom in all aspects of my work. I always felt that he fully supports me and believes in me.

A big thank goes to my colleagues Stephan Salinger, Franz Zieris, Björn Kahlert, Ulrich Stärk, and Edna Rosen. I look back and appreciate a great many discussions with their critical-constructive feedback. With pleasure I remember the good time we had.

Daniela Damian, Angela Rook, Eirini Kalliamvakou und Jordan Ell I want to thank for their warm welcome in their research lab during my stay at the University of Victoria in Canada. Their suggestions and perspectives provided important insights and ideas for this work.

All students I was allowed to support with their theses, I would like to thank for working with me towards the vision of a stable and user-friendly tool for distributed pair programming. Their ideas and efforts, as well as those of all other *Saros*[1] team members, made *Saros* a mature tool that is used in professional environments. With that they established the basis for this work.

Subsequently, very special thanks go to the two participants of this study. Their willingness to be recorded during their work for several days and, associated therewith, to accept the technologic and time-wise efforts led to the second mainstay of this work, the data.

Magdalena Priske, Martin W. Kirst, Anne Schulz, and Gesine Milde I would like to thank for giving me a part of their lifetime with proofreading this dissertation. Their different expertise provided me feedback from various perspectives.

My coach Katrin Kausch I want to sincerely thank for the inspiring meetings. Her support with calibrating my personal compass helped me to stay on target and to get this work done.

[1] see `http://www.saros-project.org` (accessed November 11, 2017) and Section 2.3 "Saros"

German version of acknowledgements:

Gedenke der Quelle, wenn Du trinkst.

Chinesisches Sprichwort

Mein größter Dank geht an meine Familie, insbesondere meine Eltern: Danke, dass ihr da seid und mich immer bedingungslos unterstützt.

Lutz Prechelt danke ich für die fördernd-fordernde Betreuung bei diesem Forschungsvorhaben sowie dem entgegengebrachten Vertrauen und Freiraum bei meiner Arbeit. Ich hatte stets das Gefühl, dass er hinter mir steht und an mich glaubt.

Meinen Kollegen Stephan Salinger, Franz Zieris, Björn Kahlert, Ulrich Stärk und Edna Rosen danke ich für die zahlreichen Gespräche und das kritisch-konstruktive Feedback auf dem langen Weg zu dieser Arbeit. Gerne schaue ich auf die überaus schöne gemeinsame Zeit zurück.

Daniela Damian, Angela Rook, Eirini Kalliamvakou und Jordan Ell danke ich für die herzliche Aufnahme in ihrer Arbeitsgruppe während meines Forschungsaufenthalts an der University of Victoria in Kanada. Ihre Anregungen und Perspektiven haben mir wertvolle Einsichten und Ideen für diese Arbeit geboten.

Allen Studierenden, die ich bei ihren Abschlussarbeiten begleiten durfte, danke ich dafür, dass sie gemeinsam mit mir an der Vision eines stabilen und benutzerfreundlichen Tools für die verteilte Paarprogrammierung gearbeitet haben. Ihre Ideen und Mühe, sowie die aller anderen *Saros*[2]-Teammitglieder, haben *Saros* zu einem reifen Tool mit einer großen Nutzergruppe gemacht. Damit haben sie die Basis für dieses Forschungsprojekt geschaffen.

Ebenso gilt der Dank den beiden Teilnehmern dieser Studie. Durch ihre Bereitschaft, sich über mehrere Tage hinweg bei ihrer Arbeit aufzeichnen zu lassen und den damit einhergehenden technologischen und zeitlichen Aufwand in Kauf zu nehmen, ist das zweite Standbein dieser Arbeit, die Daten, entstanden.

Magdalena Priske, Martin W. Kirst, Anne Schulz, und Gesine Milde danke ich dafür, dass sie mir einen Teil ihrer Lebenszeit geschenkt haben, indem sie diese Arbeit Korrektur gelesen haben. Durch ihre unterschiedliche Expertise bekam ich Rückmeldungen aus verschiedenen Perspektiven.

Katrin Kausch, meiner Coachin, danke ich ganz herzlich für die inspirierenden Gespräche. Ihre Unterstützung beim Kalibrieren meines persönlichen Kompasses hat mir geholfen, meinen Kurs zu halten und letztlich diese Arbeit fertig zu stellen.

[2]siehe `http://www.saros-project.org` (aufgerufen November 11, 2017) sowie Abschnitt 2.3 “Saros”

Abstract / Zusammenfassung

Context: Pair programming is a common agile software development practice. It means that two developers sit in front of one computer, working on the same task. They share their thoughts and jointly work out the solution. Another present trend in the software industry is **global software development**, where teams work on a project but from different locations around the world. Regarding the physical closeness of the individuals, **both approaches are diametral**.

Motivation: Although the close collaboration of pair programming and the distance of distributed teams appear to be incompatible, the biggest challenges of virtual teams are strongly encouraged by pair programming: communication, trust, and knowledge transfer. Thus, a crucial question is whether and how distributed teams can profit from pair programming by doing distributed pair programming. The little research in that field has been conducted in university contexts, and mostly focused on comparing the efficiency of pair programming and distributed pair programming. However, a consistent picture could not yet be drawn in that regard. Moreover, they have mostly utilized technical means that limited the participants' interaction possibilities.

Objective: This study is groundwork for establishing an understanding of how distributed pair programming of professional developers looks like when using a dedicated tool. Such a tool facilitates equal interaction possibilities for both participants, enabling them to separately browse their workspaces. This in turn bears the risk that they visually and mentally decouple, thereby thwarting the principles of pair programming. Accordingly, the overarching research goal is to sensitize participants to beneficial contextual conditions and behavioral patterns that avoid decoupling and other obstacles during distributed pair programming. This work is a first step towards that goal.

Method: This dissertation is a single-case study using elements of Grounded Theory methodology to conceptualize observations of a pair of professional software developers that used a dedicated tool for distributed pair programming called *Saros*.

Results: The surprisingly positive findings show that, **contrary to popular opinion, distributed pair programming can work well**. The flexible interaction possibilities facilitated by a dedicated tool led to new, subtle interactions of the pair that had positive effects on the collaboration process and that usually are not possible in pair programming that unobtrusively. First of all, these new interaction phenomena are conceptualized and described. Furthermore, personality traits, behaviors, and circumstances that appeared to be relevant for distributed pair programming are discussed. Finally, the results raised new research questions like whether **distributed pair programming is a better form of pair programming** or whether **the expressiveness of source code can be considered a 'magic' basis** for remote interaction. Apart from the above, this work tries to encourage tool evaluation for distributed collaboration from a different perspective. Often, the yardstick for evaluating tools of remote collaboration is their degree of imitating the classical, non-distributed situation. This involves the risk of not adequately addressing the complexity of the new situation in the distributed setting with its own and new interaction channels and dynamics and thus not to discover their individual potential. In light of this, the results of my research show that distributed pair programming is a different work mode to just doing pair programming in a distributed fashion.

German version of abstract:

Kontext: Paarprogrammierung (engl. pair programming (PP)) ist eine verbreitete Praktik der agilen Softwareentwicklung, bei der zwei Programmierer gemeinsam an einem Computer an einer Aufgabe arbeiten – sie teilen sich ihre Gedanken mit und erarbeiten gemeinsam die Problemlösung. Ein weiterer gegenwärtiger Trend in der Softwareindustrie ist **global verteilte Softwareentwicklung** (engl. global software development (GSD)), bei der Teams von unterschiedlichen Standorten aus zusammen an einem Projekt arbeiten. Unter dem Aspekt der Nähe der Beteiligten Personen sind **beide Ansätze diametral**.

Motivation: Obwohl die enge Zusammenarbeit bei der Paarprogrammierung und die Entfernung der verteilten Teams konträr erscheinen, adressiert die Paarprogrammierung doch genau die größten Probleme von globaler Softwareentwicklung: Kommunikation, Vertrauen und Wissenstransfer. Eine wesentliche Frage ist daher, ob und wie verteilte Teams von den Vorteilen der Paarprogrammierung profitieren können, indem sie verteilte Paarprogrammierung machen (engl. distributed pair programming (DPP)). Die wenige Forschung in diesem Bereich wurde im universitären Umfeld durchgeführt und konzentrierte sich hauptsächlich auf die Frage der Effizienz von verteilter Paarprogrammierung im Vergleich zu klassischer Paarprogrammierung. Bislang konnten diesbezüglich jedoch keine konsistenten Aussagen getroffen werden. Auch wurden für die Durchführung der Studien weitgehend technische Hilfsmittel verwendet, die die Interaktionsmöglichkeiten der Teilnehmer stark beschränkten.

Zielsetzung: Diese Arbeit ist Grundlagenforschung, um ein Verständnis dafür zu entwickeln, wie die verteilte Paarprogrammierung bei professionellen Softwareentwicklern funktioniert, wenn sie ein dediziertes Tool dafür nutzen. Mit einem Tool dieser Art sind sie in ihrer Interaktion gleichberechtigt und können sich unabhängig voneinander in ihren Arbeitsbereichen bewegen. Dies wiederum birgt das Risiko, dass sich die beiden auch visuell und mental entkoppeln können. Dies würde jedoch die Prinzipien der Paarprogrammierung konterkarieren und damit deren Vorteile, die sich aus der Gedankensynthese bei der Problemlösung ergeben. Das übergeordnete Forschungsziel dieser Arbeit ist daher, virtuelle Teams für Verhaltensweisen und Kontextbedingungen zu sensibilisieren, die Entkopplung und andere Prozessprobleme bei der verteilten Paarprogrammierung vermeiden helfen. Diese Arbeit ist ein erster Schritt im Hinblick auf dieses Ziel.

Methode: Die vorliegende Studie ist eine Einzellfall-Analyse. Sie basiert auf Beobachtungen eines Paares professioneller Softwareentwickler, die mittels eines dedizierten Tools für die verteilte Softwareentwicklung, *Saros*, zusammen gearbeitet haben. Unter Zuhilfenahme von Elementen der Grounded Theory methodology (GTM) wurden aus den konkreten Beobachtungen Konzepte gebildet.

Ergebnisse: Die überraschend positiven Ergebnisse zeigen, dass, entgegen der verbreiteten Ansicht, **verteilte Paarprogrammierung sehr gut funktionieren kann**. Die flexiblen Interaktionsmöglichkeiten, die ein dediziertes Tool bietet, führten zu neuen, subtilen Interaktionen des Paares. Diese wirkten sich positiv auf den Prozess aus und sind in dieser Dezenz bei der klassischen Paarprogrammierung oft nicht möglich. Diese neuen Interaktionsweisen werden in dieser Arbeit zunächst konzeptualisiert und beschrieben. Darüber hinaus werden Persönlichkeitseigenschaften, Verhaltensweisen und Kontextfaktoren beschrieben, die für eine gute verteilte Paargrogrammierung als relevant erkannt wurden. Die vorliegenden Ergebnisse führten wiederum zu neuen Forschungsfragen, wie beispielsweise ob **verteilte Paarprogrammierung die bessere Paarprogrammierung** ist oder ob **die Ausdruckskraft von Quellcode eine 'magische' Basis** für die verteilte Kollaboration ist. Auch möchte diese Arbeit den Impuls geben, Werkzeuge für die verteilte Zusammenarbeit aus einem anderen Blickwinkel als bisher zu evaluieren. Bislang ist der Maßstab für die Bewertung eines Tools für die verteilte Zusammenarbeit dessen Grad an Nachahmung der klassischen, nicht verteilten Situation. Dies birgt jedoch die Gefahr, nicht der Komplexität der neuen, verteilten Situation mit neuen Interaktionskanälen und Dynamiken gerecht zu werden und deren Potential zu entdecken. Unter diesem Aspekt zeigen die Ergebnisse dieser Arbeit, dass die verteilte Paarprogrammierung etwas anderes ist als nur Paarprogrammierung mit verteilt sitzenden Teilnehmern.

Selbständigkeitserklärung

Ich habe die vorliegende Dissertation selbständig, ohne unerlaubte fremde Hilfe angefertigt. Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die vorliegende Arbeit ist in dieser oder anderer Form zuvor nicht als Prüfungsarbeit zur Begutachtung vorgelegt worden.

Teile der Arbeit wurden in Schenk et al. (2014) publiziert.

Notation

Citation:

- References as part of a sentence designate the source of what is said in the sentence.
- References after a paragraph, not being part of a sentence, refer to the whole paragraph.

Markup:

- Direct quotations are set in *italics* and enclosed in double quotation marks.
- Proper names of products or alike are written in *italics*.
- Key points of a section are written in **boldface**.
- Defined names of concepts or patterns are CAPITALIZED.
- When referenced in the text, the title of chapters and sections are enclosed in double quotation marks.

This dissertation uses the masculine form in a generic sense which includes both masculine and feminine.

Chapter 1

Introduction

Chapter Structure – *What to expect in this chapter and why*

This chapter spans the contextual frame of this inquiry. Context of this work is the agile software development practice pair programming (PP) in connection with global software development (GSD). Accordingly, this chapter starts by introducing these basic ideas and their relevance in the software industry. Then, as a consequence of the benefits of PP and the trend of GSD, distributed pair programming (DPP) is introduced. Since DPP is particularly vulnerable to awareness issues, this is followed by a sensitization for awareness and its relevance for collaboration and in particular for remote real-time collaboration.

Subsequently, Section 1.2 "Approaches to Distributed Pair Programming (DPP)" outlines three different technical approaches of how to realize DPP, and discusses their respective pros and cons. One of these approaches, extended distributed pair programming (eDPP), is subject of this study. Thus, thereafter, the related overall research question is presented.

To integrate the research at hand into the existing state of knowledge, Section 1.4 "Related Work" gives an overview of research areas directly related to DPP, such as PP, GSD, and technical means for distributed collaboration along with their specific challenges. The chapter closes with an overview of what little research there is on DPP. Section 1.4 "Related Work" focuses on closely related research with the purpose to:

- provide a context for understanding the topic of this work and its relevance
- give background information to understand and assess the results of this study
- give an overview of related tools and the eDPP tool used in this study to evaluate its fitness for industrial use
- understand the added value of the current work in its research field

PP is a software development practice where program code is developed in pairs. The pair members sit next to each other, share one computer, and jointly implement the solution for a given problem. Together, they focus and work on the same aspect of the problem solution. Ideally, their knowledge and thoughts complement and inspire each other. This is an intense work mode, requiring fine-grained coordination and intense communication.

If applied properly, PP has proven to be worth its invest in double staffing because it positively impacts aspects like product quality, efficiency, knowledge sharing and communication in the team, as well as

the team member's job satisfaction (see the following Section 1.1.1 "Pair Programming – a Common Agile Practice" and Section 1.4.1 "Pair Programming (PP)").

At times of globalization and internet bandwidth for low cost, however, teams often are spread around the globe. The dispersed team members cannot sit next to each other and share one computer. And if they still want to do PP?

Distributed team members can do DPP. This means they use dedicated tools which allow them to communicate and to jointly work on program code. Compared to the side-by-side situation, such a technical setting is still cumbersome and far behind the natural mechanisms regarding unobtrusiveness, ease of interaction, and richness of transmitted information like spectrum of voice, facial expressions, and paralinguistic nuances. These aspects, however, are essential for efficient collaboration. Accordingly, **virtual teams face the question if DPP is a reasonable endeavor at all**.

Research so far could not draw a consistent picture in that regard, but it could neither be clearly shown that teams doing DPP performed significantly better nor worse than PP with non-distributed participants. Nearly all inquiries have been conducted in university contexts, aiming at measuring the product quality in lines of code or passed test cases. Or they determined the efficiency of DPP by the students' grades. Furthermore, the inquiries mostly used workarounds, alienated tools which did not grant freedom of interaction.

However, dedicated DPP tools explicitly address the demands of PP and efficient collaboration. They enable both participants to easily edit a shared code base while keeping it in sync. From their individual machines, participants can follow the other one's position and editing activities in real-time. In particular in combination with a channel for verbal communication, there is a good chance that the dedicated tools empower distributed teams to successfully do DPP.

Since distributed teams do not have a choice whether to do PP or eDPP, this work does not strive for a quantitative comparison of PP and DPP. Distributed teams face the question whether to do eDPP or rather not undertake such tight cross-site collaboration. To answer this question it is necessary to understand whether DPP can be a reasonable endeavor and, if so, how to successfully do it. Therefore, aspects and behavior need to be identified which may lead to successful DPP collaboration, or rather the opposite.

This dissertation is groundwork to develop an initial understanding of DPP phenomena and their impact on the collaboration process. It is is a single-case study based on direct, non-participatory observation (recordings) of one week of work of a pair of professional software developers successfully carrying out DPP.

These two developers used the dedicated DPP tool *Saros*[3]. With *Saros*, each participant has a local copy of the shared artifacts and uses his own keyboard and mouse. Each of them can independently scroll or switch files in the shared workspace or move to an unshared workspace area. Both can freely edit every artifact while *Saros* keeps their copies in sync in real-time by applying the local changes to the remote partner's files. Due to this interaction and browsing freedom for both participants, this approach to realizing DPP is referred to as eDPP (see Section 1.2.3 "extended Distributed Pair Programming (eDPP)").

The observed pair was quite experienced with eDPP. They were doing their regular work in their familiar environment. The case study indicates that eDPP can work quite well – the collaboration of the pair was smooth and did not exhibit notable signs of clumsiness or productivity losses. Actually, they used their interaction freedom to help the process in a way that would not have been possible in PP.

Thus, this inquiry is also a proof of existence for something that has not been examined before and that in the area of computer-supported cooperative work (CSCW) research is assumed not to be feasible (da Silva Estácio and Prikladnicki, 2015; Olson and Olson, 2000): the eDPP process of experienced professionals. More precisely, it is about the **understanding of industrial distributed pair programming that appears to be not inferior to local pair programming**.

[3] `http://www.saros-project.org` (accessed November 11, 2017)

Considering the relevance of dissemination of GSD and PP (see the next sections), **this work focuses on a very specific aspect of software development which has, however, a wide relevance in the area of software development**.

1.1 Agile and Global Software Development

Agile methods are well established in software development – more than 80% of professional developers state to work in an agile manner (see Figure 1.1).

The underlying principles of agile software development emphasize the importance of lightweight, fluent processes and a most intense communication of all parties involved[4]. These principles in combination with an iterative approach take account of the repeated experience that especially in software development a 'plan ahead' and then 'do it right the first time' attitude does not work (Williams, 2012). Actually, such projects often flop (Williams, 2012). An agile mindset attaches importance to an ongoing user-feedback based on concrete intermediate product versions. This attitude is considered as necessary base for developing software of high value for the customer or the user. To efficiently handle frequent feedback, a flexible software process is needed. Such a process must be fluent and be able to quickly react to changed or new requirements. This demands to reduce aspects making the process viscous, like long-term planning horizons, tedious documentation, slow and cumbersome communication channels, or poorly motivated staff just taking orders. Instead, lightweight processes with competent team members who feel responsible for their project are needed. They should be able to efficiently communicate, have comprehensive knowledge of the product as well as the project status, and a good judgement concerning effects of requirement changes.

Agile software development involves agile process models like *Scrum*[5] as well as concrete implementation practices like PP. Agile process models describe the overall process in terms of general activities and roles throughout the software development process and refer to the overall process on team level. Concrete agile practices define how certain development activities have to be carried out, such as how testing has to be performed, how requirements have to be captured or how source code has to be developed. PP is a practice for the latter.

As can be seen in Figure 1.1, *Scrum* as a software development process is widely used in professional software development (70% usage). Nevertheless, agile processes like *Scrum* are not in the focus of this thesis because they are not about close collaboration of individual programmers and therefore of low relevance with respect to the research question. The next most commonly used item shown in Figure 1.1 is PP. Since it belongs to the main focus of this thesis, its relevance is outlined in the following section. Further aspects of PP will be discussed in detail in Section 1.4.1 "Pair Programming (PP)".

[4] `http://agilemanifesto.org/` (accessed November 11, 2017)
[5] `https://www.scrum.org` (accessed November 11, 2017)

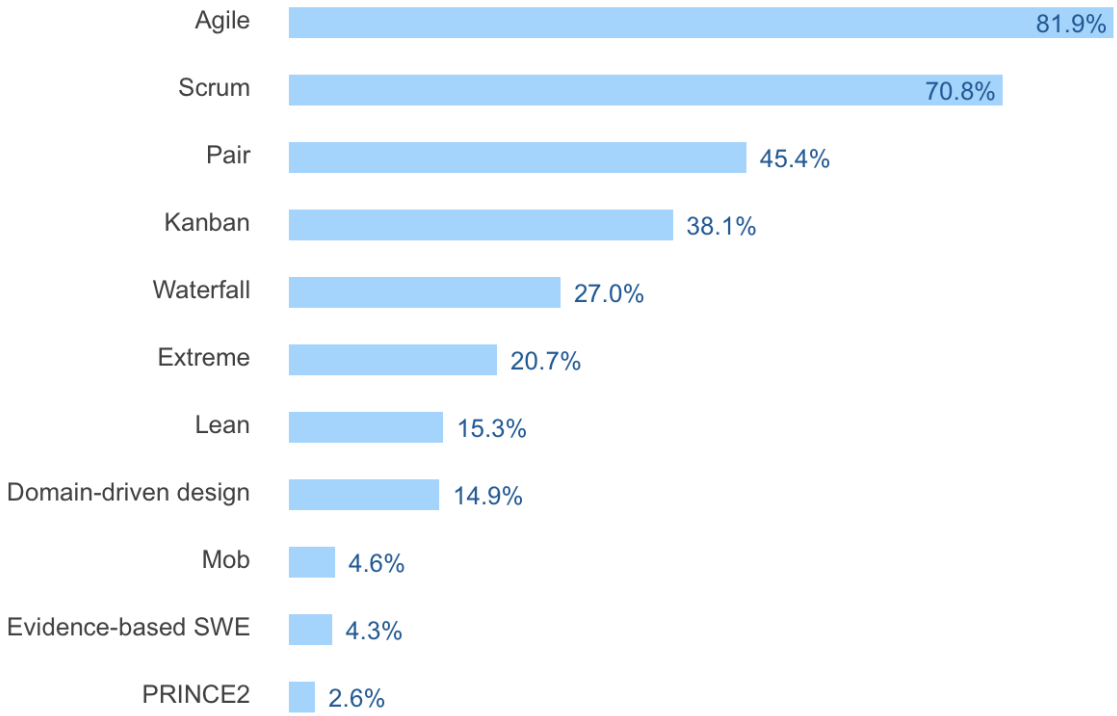

Figure 1.1: Dissemination of agile software development and pair programming in professional software development.

Stack overflow survey from 2017, 20.519 responses for this question (`https://insights.stackoverflow.com/survey/2017#development-practices/` (accessed November 11, 2017)).

1.1.1 Pair Programming – a Common Agile Practice

45% of professional software developers use PP (see Figure 1.1) *"to improve their code quality and skill set"* [6].

Pair Programming (PP) *(Explanation of terms)*

> Pair programming is characterized as follows: Two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test (Williams and Kessler, 2000). *"The two programmers are like a coherent, intelligent organism working with one mind, responsible for every aspect of this artifact. [...] Both are equal, active participants in the process at all times and wholly share the ownership of the work [...]"* (Williams et al., 2000).
>
> One of the participants is called the driver, the other is called navigator (Hulkko and Abrahamsson, 2005; Plonka et al., 2012a) or observer (Salinger et al., 2013). The driver has control over the mouse and keyboard and is responsible for editing the code. The observer is following the mental trail of the driver, being available as *"an ever-ready brainstorming partner to the driver"* (Hulkko and Abrahamsson, 2005). The participants can and should switch roles periodically at any desired point during their collaboration (Plonka et al., 2012a,b; Williams et al., 2000).

The roles are defined according to the ownership of the input devices. Originally, this definition coupled the ownership of the physical devices with the type of mental engagement in the collaboration: it suggested that the observer thinks on a strategic level and reviews the code while the driver is writing the code (Bryant et al., 2008).

This strict coupling of input device ownership and abstraction level of thinking has proven not to be true. PP studies found that this driver/observer distinction is misleading and inappropriate. For example, when examining the abstraction levels of the driver's and observer's statements, Bryant et al. (2008) found that *"the pattern of abstraction levels of navigator's utterances are very similar to those of the driver"*. Both, the driver as well as the navigator, mainly think on an abstract level instead of just a semantic level (Bryant et al., 2008). Chong and Hurlbutt (2007) also contradict the widely assumed characterization of the roles, pointing out that both participants, independent of their actual role, discussed about implementation details as well as strategic issues – **the pair moved to different levels of abstraction as a whole**. Similarly, Höfer (2008) found that the pairs frequently switched roles. These switches were effortlessly and without further explanations or introductions for the prospective driver. Based on this, the authors conclude that the participants have to be mentally on an equal level, thinking about the same things because otherwise they would have to 'lift' each other to an equal mental level when performing a role switch.

Interestingly enough, this independence of input device ownership and level of thinking is in line with the findings of this work. In eDPP, also the observer can use his keyboard and mouse at any time. And, as will be shown in Chapter 4 "Results", he used them to contribute to the program code or look things up at very different levels of abstraction.

Working as 'one brain' at the same computer in PP requires intense communication and a fine-grained coordination of mental and physical activities and makes PP a multifarious research area. And although findings differ, in general they agree that PP positively affects efficiency, product quality, communication and knowledge transfer within the team, code responsibility, learning, and job satisfaction. For the software industry, these are vital aspects because bad or defective program code is costly – and this applies to companies with collocated as well as with distributed teams.

PP will be discussed more comprehensively in Section 1.4.1 "Pair Programming (PP)".

[6] `https://insights.stackoverflow.com/survey/2017#development-practices` (accessed November 11, 2017)

1.1.2 The Trend of Global Software Development (GSD)

Another *"prevalent trend"* (Jain and Suman, 2015) in the software industry is global software development (GSD). The rapid and significant improvement of internet, computer, and software technology and their availability at low cost let the world virtually grow together, even across big geographical distances (Ågerfalk et al., 2008). *"Research from the Institute of Corporate Productivity stated in 2008 that 67 percent of the companies surveyed felt their reliance on virtual teams would grow in the next few years. For larger companies, this figure rose to 80 percent"* (Olson and Olson, 2014).

Global Software Development (GSD), Distributed Software Development (DSD), and Virtual Team *(Explanation of terms)*

GSD refers to software development with team members, or parts of them, spatially separated but collaboratively working on the same software project. The spatial distribution can reach from residing in different buildings at one company site up to being spread either within the same country or across countries or continents. (Layman et al., 2006)

The team members may belong to the same or to another organization (Misra et al., 2013). When the team is nationally distributed, their collaboration is referred to as distributed software development (DSD), or, from the organizational perspective, also referred to as **onshoring**.

When the team members are spread across national boundaries, this is called global software development (GSD) (Jain and Suman, 2015), also referred to as **offshoring**.

Offshoring done with nearby countries (same time zone, similar culture, short travel time) is specified as **nearshoring**. To some degree, this leverages some of the big challenges of GSD because it avoids communication and coordination problems due to different time zones and (corporate and personal) cultural differences.

Offshoring done with far off countries is referred to as **farshoring**.

Since in GSD the team members, or parts of them, are not physically collocated but still work against a shared goal, they are a **virtual team** (Baheti et al., 2002a).

Regardless of the location dimension, **the collaboration in GSD can either be synchronous or asynchronous**. Asynchronous collaboration means that tasks are not handled simultaneously on the different sites but are finished on-site and then integrated with the work of the other site(s) at different times. Asynchronous collaboration is supported for example via e-mail, task management software, version control systems, or other tools for combining information or work results. Synchronous collaboration means that whole teams or individuals in real-time commonly work on the same task or goal – having meetings or creatively working on artifacts. That way of collaboration may be supported by software for virtual audio or video meetings, or tools for real-time collaborative writing like Google Docs[7] (Olson and Olson, 2014), or similar tools for collaborative editing of source code, mind-maps, or other artifacts.

Among other advantages, the potential benefits of GSD are reduced staff costs due to wage differences among countries, proximity to local markets, increased staff pool, and round-the-clock development when involving different time zones. In Western Europe, for example, nearshoring is done due to the shortage of skilled professionals and the wage differences between Western and Eastern Europe.

Despite the potential benefits, the distribution and limited communication in GSD lead to some not as obvious but essential challenges. Apart from the evident time-zone differences and cultural clashes, the tool-dependency strongly hampers natural, need-based ad-hoc communication and leads to further downstream problems. As there is nobody looking over the other's shoulder nor does anyone automatically perceive what is going on, the awareness of the other's work is greatly reduced (Olson

[7] `http://www.google.com/intx/en/enterprise/apps/business/products/docs/` (accessed November 11, 2017)

and Olson, 2014). Information and status of work need, on the one site, explicitly be made visible and, on the other side, explicitly be monitored. Olson and Olson (2014) describe this as the *"blind and invisible"* phenomenon: *"People on distributed teams are invisible to people not at their location. Symmetrically, these same people are blind to the activities of the remote people. This means that necessary information must be communicated explicitly, through the heavy use of email, blogs and/or wikis, and audio or video conferencing"*. Other consequences of the missing personal contact and communication are the lack of trust and the absence of common ground, a shared vocabulary and work style (Olson and Olson, 2014). These difficult conditions often lead to a low motivation to undertake explicit (additional) efforts to collaborate or communicate cross-site, or to make remote teams aware of the current work status. These briefly touched aspects of GSD are discussed in more detail in Section 1.4.2 "Global Software Development (GSD)".

Notwithstanding, GSD is present and an important aspect for developers: *"When we asked respondents what they valued most when considering a new job, 53.3% said remote options were a top priority. 63.9% of developers reported working remotely at least one day a month, and 11.1% say they're full-time remote or almost all the time"*[8]. Figure 1.2 reveals the different variants of remote working hours in the software industry in 2017.

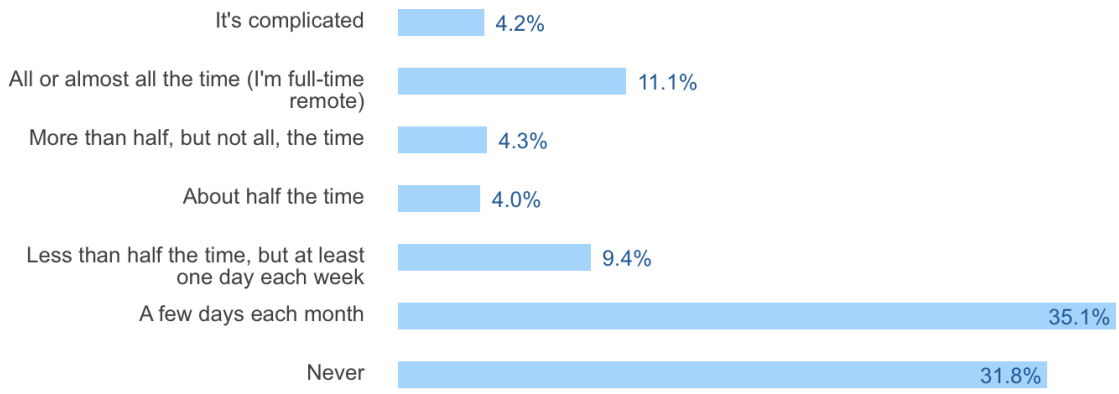

Figure 1.2: Variants of remote working hours in software development

Stack overflow survey from 2017 [9], 44.000 responses in total for this question.

In conclusion, GSD is common practice for software companies nowadays. A plausible consequence of the dissemination of remote work and the benefits of PP is, that distributed teams also actually want to undertake PP. However, as discussed above, intense communication and collaboration is especially difficult in distributed settings and, accordingly, doubts arise concerning the compatibility of both:

"As a consequence of these major trends, software development organizations have been striving to blend agile software development methods like Extreme Programming and distributed development to reap the benefits of both. However, agile and distributed development approaches differ significantly in their key tenets. For example, while agile methods mainly rely on informal processes to facilitate coordination, distributed software development typically relies on formal mechanisms" (Ramesh et al., 2006).

Looking at it the other way around, agile software development practices and in particular PP address exactly the biggest challenges of virtual teams (Bandukda and Nasir, 2010; Hanks, 2003) – PP fosters personal contact, trust building, intense and informal communication, collaboration and knowledge exchange.

[8] A survey from stack overflow in 2017: `https://stackoverflow.com/insights/survey/2017#remote-work` (accessed November 11, 2017)

To allow distributed teams to benefit from the effects of PP, a possible solution is to let them do PP mediated by technical means that bridge their physical distance. This setting is referred to as DPP.

1.1.3 Distributed Pair Programming

Distributed Pair Programming (DPP) *(Explanation of terms)*

> DPP, also referred to as virtual pair programming, refers to the synchronous (real-time) collaboration of two developers as in PP but from different physical locations (Stotts et al., 2003). They have to use a technological infrastructure that emulates the local PP setting (collaborative programming on one machine).

DPP is an intense form of distributed, synchronous collaboration of two individuals and, as in PP, requires intense communication and a sophisticated coordination of mental and physical activities.

Shared Workspace and Virtual Shared Workspace *(Explanation of terms)*

> A shared workspace is the physical frame in which a group accomplishes a task. It comprises the work artifacts (i.e. a sheet of paper), the tools (i.e. scissors) as well as the relevant physical infrastructure (i.e. tables and walls).
>
> In case of program code, the artifacts are of immaterial nature and they are manipulated by non-physical instruments, i.e. by software tools. In that case, due to its immaterial nature, the workspace is a **virtual workspace**. In PP, the pair works in a virtual workspace – the programing environment – but is situated in a shared physical workspace (table, walls, input devices, etc.).
>
> When a group jointly works in a virtual workspace, they have a **virtual shared workspace**. In DPP, each participant has its own physical surrounding but the collaborative work happens in the shared virtual workspace. Since this virtual shared workspace is the relevant part where the actual DPP activities are happening, it is usually only referred to as shared workspace.

Compared to local PP, there are significant differences in the distributed situation: When sitting next to each other on one machine, both pair members see each other and are unrestricted in their verbal and nonverbal communication. The observer can simply perceive what artifact the driver is working on and can see his detailed actions. He usually knows what overall goal these actions relate to. Equally, the interpretation of the observer's facial expressions and gestures indicate his mood and focus of attention. Finally, both participants automatically perceive situational factors and gain a detailed mutual understanding of what the other one is doing in the general physical environment as well as in the *shared workspace*. The non-verbal signals and most information about the joint working situation are not available in a virtual shared workspace. Nevertheless, such so-called awareness information are of great importance for a smooth and efficient collaboration.

1.1.4 Awareness

Roughly spoken, awareness is the general knowledge about what is going on in a (collaborative) situation as well as about several other contextual factors of the situation and the parties involved. It helps to master everyday situations by integrating perceived events into existing knowledge and experience, and accordingly to align one's own actions. *"This conception of awareness involves states of knowledge as well as dynamic processes of perception and action"* (Gutwin and Greenberg, 2002).

Awareness *(Explanation of terms)*

Awareness is the result of an ongoing process of perception and interaction with our environment. It is usually gained intuitively. In a situation, awareness comprises all aspects of that situation like context and reason of that situation, activities, the place itself, as well as people and their moods, roles and relationships. For each individual this provides a kind of **knowledge space in which their own perceptions and interpretations are integrated**. On the one hand, this helps to understand the situation and others' behavior, and, on the other hand, to align one's own behavior and actions with others' activities. Hence, awareness is key for an efficient collaboration. (Gutwin and Greenberg, 2002)

Gutwin and Greenberg (2002) summarize the following four basic characteristics of awareness:

- Awareness is knowledge about the state of an environment bound in time and space.
- Environments change over time, so awareness is knowledge that must be maintained and kept up to date.
- People interact with and explore the environment, and the maintenance of awareness is accomplished through this interaction.
- Awareness is an auxiliary goal —– that is, the overall goal is not to maintain awareness but to complete some task in the environment.

As indicated above, awareness involves all aspects of a situation, and several classifications of these situation-relevant information were made – varying in their detail and focus. With the focus on face-to-face collaboration, Gutwin and Greenberg (1995) divided awareness information into four non-exclusive, interrelated categories:

Informal Awareness: General knowledge about people's position, doings, and availability within the physical surrounding. It is gathered by being in the same physical location and facilitates casual interaction and finding collaboration opportunities (Gutwin et al., 1996). A typical example of using informal awareness is to see someone on his way to the coffee machine and decide to also get a coffee and to start a conversation.

Group-Structure Awareness: Being in the loop about people's positions and areas of authority within a group. It is gained by knowing the group and the people and helps for example to find appropriate contact persons for concerns. In the example above group awareness would help to decide whether the person going to the coffee machine is the right person to address for a specific issue.

Social Awareness: Knowledge about others' mood, interest and subject of focus. Hearing others' verbal utterances and perceiving their body language allows adequate social interaction. For example, before the colleague went for a coffee, the other one went by the office, saw him frowningly staring at his monitor and decided better not to disturb yet.

Workspace Awareness: The notion about what others are doing where in the shared workspace. Seeing and hearing others work, or the effects of their actions allows to align and integrate one's own actions into those activities. For example, while collaboratively crafting a poster and seeing that someone works on a certain position influences the personal decision which section of the poster to work on to avoid physical collisions.

When referring to the managing of complex, dynamic, or risky situations, like for example when piloting a warplane, the general term awareness is sometimes specified as situation awareness (Gutwin and Greenberg, 2002).

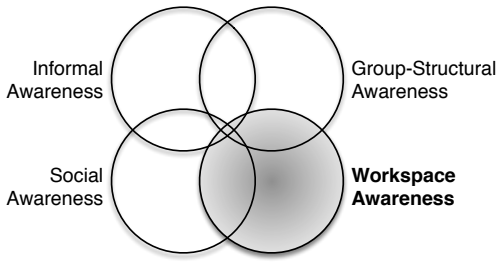

Figure 1.3: Types of awareness in group work according to Gutwin and Greenberg (1998)

Situation Awareness *(Explanation of terms)*

Situation awareness is defined as *"the up-to-the minute cognizance required to operate or maintain a system"* (Gutwin and Greenberg, 2002). As a precondition for conscious acting within a situation it comprises the *"perception of relevant elements of the environment"*, the *"comprehension of those elements"* and the *"prediction of the states of those elements in the near future"* (Gutwin and Greenberg, 2002).

The collaboration of people in a shared workspace is a specific situation, involving complex, dynamic interactions in that workspace. Accordingly, workspace awareness is *"a specialization of situation awareness, one that is tied to the specific setting of the shared workspace"* (Gutwin and Greenberg, 2002). It involves the collaboration-related elements of informal awareness, social awareness, and group structure awareness, abstractly shown in Figure 1.3.

Workspace awareness is particularly evident when working on the same item. When knowing what others do in a shared workspace, people can coordinate their actions in a natural way. Explicit rules or technical restrictions are not required. (Gutwin and Greenberg, 2002)

This emphasizes the predominant relevance of workspace awareness in particular for every kind of real-time collaboration, which will be focused on in the following Section 1.1.5 "Workspace Awareness".

1.1.5 Workspace Awareness

Workspace awareness provides information about what is happening in the workspace. It comprises what others are doing in the workspace as well as the state of the artifacts, which serves as a representative for the state of the work performed on them. (Gutwin and Greenberg, 2002)

Workspace Awareness *(Explanation of terms)*

Workspace awareness is the knowledge about the current situation and the actions of others in the shared workspace. This refers to what others are currently doing where in the workspace, and to anticipate their intentions. (Greenberg et al., 1996)

In particular for the latter aspect, also informal, social, and group structure awareness are very helpful – knowing the other one, his expertise, role, mood, and alike help to assess and anticipate his actions.

This knowledge is the fundamental context for aligning and integrating one's own activities in a collaboration (Dourish and Bellotti, 1992).

Workspace awareness information are based on visual and auditive information extracted from the physical environment and **enable collaborative work to be effortless and less error-prone** (Gutwin and Greenberg, 2002). Watching and hearing other people doing their work as well as seeing artifacts change provides the subtle details that are needed for an effective and smooth joint work: *"The benefits of workspace awareness are subtle, but over the course of a collaborative interaction, they can markedly improve a group's effectiveness"* (Gutwin and Greenberg, 1996).

However, when a group is physically distributed, they do not have a shared physical environment, and workspace awareness cannot be obtained in the natural, easy way: *"Unfortunately, many of the things that supported workspace awareness in face-to-face situations, such as peripheral vision, rapid glances, three-dimensional sound, and the ability to see the entire workspace, disappear in the transition to a groupware setting"* (Gutwin and Greenberg, 1995).

Groupware *(Explanation of terms)*

Groupware is group-aware software — it is aware of the fact and fulfills that requirement that it is used by a group of people. Moreover, it helps groups of people to do their collaborative work by providing access to a shared working environment (Ellis et al., 1991). *"The goal of groupware is to assist groups in communicating, in collaborating, and in coordinating their activities"* (Ellis et al., 1991).

DPP is also realized via groupware, with a group size of two. Accordingly, **workspace awareness is a central issue in DPP and eDPP**. In both PP and eDPP, the participants continuously collaborate on the same object and within that they usually focus on the same code segment. The following Section 1.1.6 "Workspace Awareness in DPP" presents the workspace awareness issue for the specific case of real-time collaborative work like DPP.

1.1.6 Workspace Awareness in DPP

In contrast to PP, in DPP each developer works on his own computer, and collaboration is realized via a groupware tool. Accordingly, any interaction is restricted by the groupware, and the perceivable workspace environment shrinks drastically. The pair members are not automatically aware of what the other one can see, what object he is working on and what he is doing. Neither is social awareness information, which is particularly derived via non-verbal communication channels, available.

Settings vary, but usually during DPP the participants are audio-connected for their verbal communication. Dedicated groupware for real-time collaboration tries to overcome awareness issues by providing explicit workspace awareness information about the presence and location of others in the workspace. Nevertheless, these awareness information are an artificially generated subset of the rich information available in the collocated setting: *"Real-time distributed groupware often provides virtual workspaces. However, interactions within workspaces are impoverished when compared with their physical counterparts"* (Gutwin and Greenberg, 1996).

PP thrives on personal communication and smooth coordination which in turn involve body language and workspace awareness. Groupware provides only a very limited subset of both, if at all in the case of non-verbal communication cues. This raises the question whether DPP (PP done via a groupware) is a reasonable effort or whether applying an agile practice like PP to a distributed setting *"is a paradox predetermined to failure"* (Schümmer and Lukosch, 2009).

1.2 Approaches to Distributed Pair Programming (DPP)

"The most common causes of pair-programming burnout are technical issues. If you spend an hour before a pairing session just trying to get connected or if your connection drops midsession, then you're going to grow weary. Likewise, if you're using a textual editor that gives one half of the pairing team too much control, you might grow tired of feeling that you are not contributing to the work product" (Kutner, 2013).

DPP can be realized via various technical tools or collections of tools. They differ in their tailoring for DPP, their supported interaction possibilities, and provided awareness information.

In his book *Remote Pairing*, the programmer and practitioner of DPP, *Joe Kutner*[10], outlines several (suites of) tools which are actually used by software practitioners to do DPP. The book is not a scientific consideration, but *Kutner* pragmatically discusses the different configurations from the viewpoints convenience of set-up and equality of interaction. This pragmatic perspective may serve as a valuable grounding for the subsequent discussion of general DPP approaches and their practical suitability.

- Screen sharing: using *Skype*[11] or *VNC*[12]
- Remote desktop: using *Google Hangouts*[13] or *Screenhero*[14]
- Shared editor: using a terminal-based text editor like *Vim*[15] or *Emacs*[16] in combination with the terminal multiplexer *tmux*[17] and *ssh*[18] connection
- Distributed editor: using the *Eclipse*[19] integrated development environment (IDE) extended with the shared editor plugin *Saros*

Roughly spoken, the tools shape a continuum of DPP-supporting functionalities. This ranges from screen sharing, which *"does not provide any specific tool support for distributed pairing"* (Hanks, 2008) but which *"permits the pair to use any single-user application that they wish"* (Hanks, 2008), to dedicated DPP tools that prevent *"potential users from using their preferred development tools, thus limiting the DPP environment's audience"* (Hanks, 2008). The latter also involve the problem that their provided functionality does not adequately support the DPP activities (Hanks, 2008).

The different tools and features shape different technical setups for realizing DPP, which can be grouped into **three basic approaches: screen sharing, remote pair programming (RPP), and extended distributed pair programming (eDPP).**

These three approaches are presented in the following sections. Figure 1.4 shows who may see and do what with the respective approaches. To facilitate matters, it draws on the concrete scenario that two programmers named Alice and Bob work on different sites and want to do PP.

[10] `http://jkutner.github.io/` (accessed November 11, 2017)
[11] `http://www.skype.com/` (accessed November 11, 2017)
[12] `https://www.realvnc.com/` (accessed November 11, 2017)
[13] `https://plus.google.com/hangouts` (accessed November 11, 2017)
[14] `https://screenhero.com/` (accessed November 11, 2017)
[15] `http://www.vim.org/` (accessed November 11, 2017)
[16] `http://www.gnu.org/software/emacs/` (accessed November 11, 2017)
[17] `http://tmux.sourceforge.net/` (accessed November 11, 2017)
[18] `http://www.openssh.com/` (accessed November 11, 2017)
[19] `https://www.eclipse.org/` (accessed November 11, 2017)

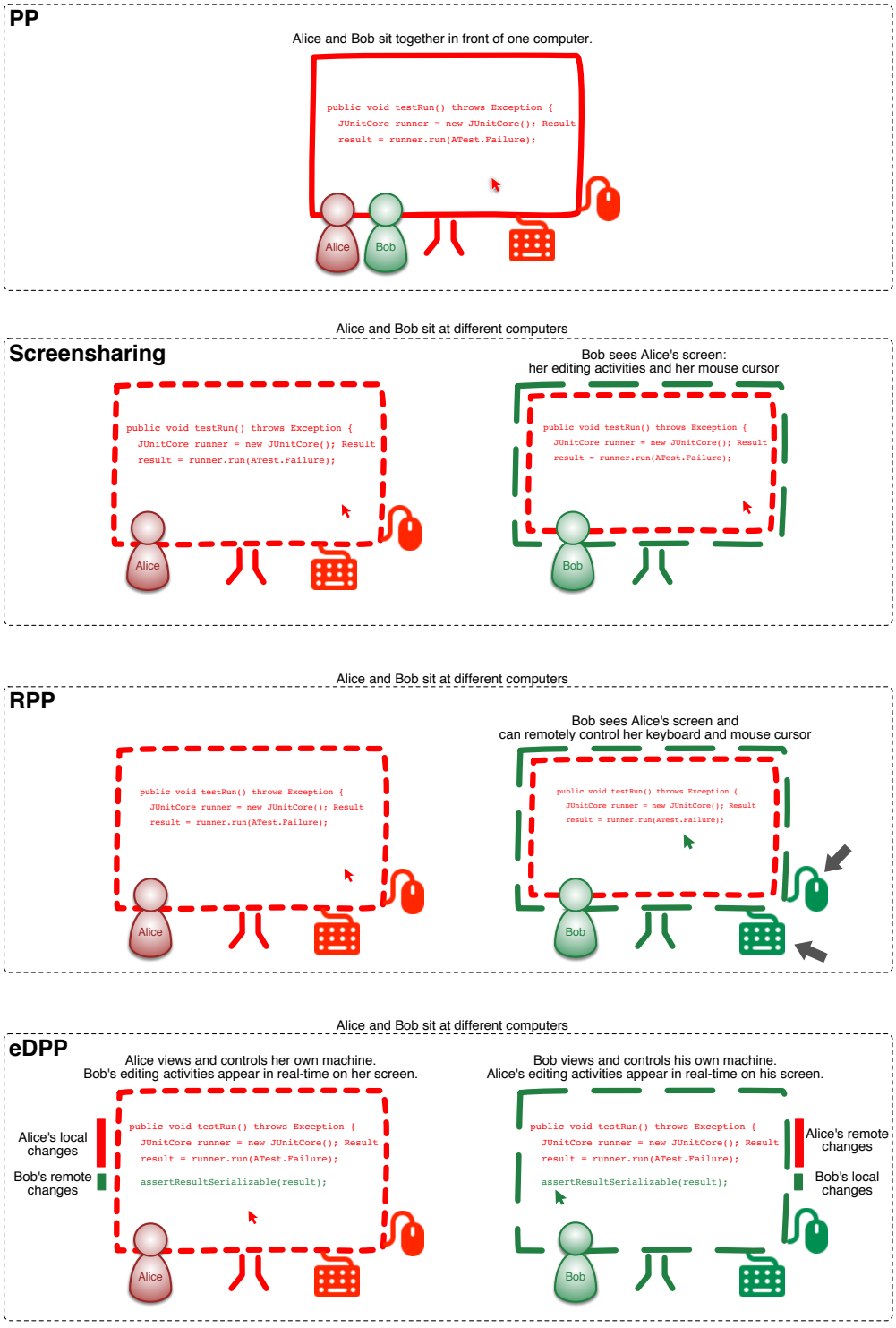

Figure 1.4: Technical approaches to DPP

From PP over Screen Sharing to remote pair programming (RPP) and eDPP – who can see and do what.

1.2.1 Screen Sharing

One way to imitate DPP, albeit without the possibility of role-switches, is using screen sharing where **Alice works on her local machine and Bob can view her screen** using a screen sharing tool like *VNC*, *Adobe Connect*[20], *Skype*, or alike.

Screen sharing usually allows to share the whole screen or individual windows. As in local PP, Alice and Bob can see the same screen contents, as long as they have about the same screen sizes. Stotts et al. (2003) even suggest that Bob *"should have a screen size slightly larger than the host machine; this allows the window showing the host PC to fit entirely on the remote, requiring no scroll bars"* (Stotts et al., 2003).

With a short delay, Bob can follow the actions happening on Alice's screen and see the changes she makes in the artifacts. Therefore he knows what Alice is doing in the workspace and, in that regard, the **workspace awareness is good**.

However, there are several drawbacks when sharing screens: Changes in the workspace caused by keyboard shortcuts of Alice suddenly appear in the workspace, and Bob is not able to understand the reason behind them. This problem is common to all DPP approaches and might be be addressed by explicit awareness information about the usage of keyboard shortcuts on the remote site. Apart from this, the awareness of the physical actions and reactions of Bob is far lower than in local PP. Even with a video connection, subtle cues would be hard to detect (see page 75 in Section 1.4.5.1 "Efficiency of DPP".

Moreover, Alice and Bob are tied to a unilateral workspace and since screen sharing does not provide two-way control (Kutner, 2013), **Bob is only a passive visitor, unable to interact with Alice's machine**. In Figure 1.4, this is illustrated by the missing input devices on Bob's machine. He is just able to follow Alice's mouse and text cursor. He is not even able to copy and paste something from his machine in the shared artifact, which would be *"a convenient feature"* (Stotts et al., 2003).

Role switching – when Bob would like to become the driver – is quite awkward too, and therefore very unlikely to happen: It would require a re-start of the screen sharing session, with Alice being the visitor, and a transfer of the joint artifacts to Bob's machine. Since role-switching should be possible easily and happen frequently in PP, **screen sharing is no more than a poor approximation to DPP**.

In particular with regard to the PP principle of equality of sharing, which says that *"tools should not give one party a control advantage over the other"* (Kutner, 2013), this approach is quite suboptimal (see also Section 1.4.1.4 "Principles for Good Pair Programming"). Screen sharing does not equip both participants with equal possibilities but limits the possibility of collaborative pair-work and role-switching: *"these sessions were sometimes hampered by technology as the screen sharing software did not allow for concurrent editing and this practice was perceived as frustrating for the developers"* (Modi et al., 2013). Screen sharing may thus be applied for some simple purposes, but is not considered as a reasonable mode for DPP (Kutner, 2013).

1.2.2 Remote Pair Programming (RPP)

A notable improvement can be reached with **remote desktop applications that, additionally to screen sharing, enable two-way control of the input devices** like for example in *Google Hangouts*. In this setting, both participants are still tied to Alice's workspace while Bob may also control her input devices for pointing, editing, and so on. Nevertheless, Alice's mouse and keyboard still support only one active focus point, so Alice and Bob compete for the input devices — if Bob grabs for the mouse, he takes it away from Alice. In Figure 1.4, this is illustrated by equipping Bob with input devices while still remotely looking at Alice's screen. The **physical and workspace awareness is the same as for screen sharing**, but with the difference that Bob now can engage in the collaboration to some degree.

[20] `http://www.adobe.com/products/adobeconnect.html` (accessed November 11, 2017)

A solution to this 'mouse-grabbing' issue is offered by the tool *Screenhero*, which simultaneously provides two mouse pointers so that each participant controls his own mouse but also sees the other one's mouse pointer. With this, Bob's situation is slightly better, but he is nonetheless 'the remote' visiting Alice's machine.

From the interaction-possibilities' perspective, this setting can be considered as an approximation to local PP and thus is referred to as **remote pair programming (RPP)**.

Nevertheless, the pair is far away from working as peers – the setting involves an asymmetric situation: whether Alice and Bob compete for the mouse or not, bandwidth requirements are yet a big issue for screen sharing and remote desktop applications (Kutner, 2013). Due to network latency, Bob is disadvantaged for viewing and operating Alice's computer and he is not working in his familiar environment. However, Bob *"is second class. [...] Finally, while both programmers can type and click at the same time, there is only a single mouse and keyboard focus. Characters become intermixed when typed at the same time and windows can fight for focus. The pair must actively cooperate and the driver must take their hands off the keyboard when the observer is using it."*[21]

Although screen or application sharing may have its advantages for 'exotic' (other than source code) artifacts (Djemili, 2006), RPP is not a satisfactory solution for DPP which requires equal interaction and easy role switching: *"[...] we must function as peers. Both programmers must be able to control the environment equally and contribute to the code base"* (Kutner, 2013).

A way to eliminate the bandwidth issue for RPP is avoid transferring whole graphical interfaces but to share a text-based terminal session. This special case of RPP still involves the asymmetry of 'host' and 'remote' and is quickly explained in the next Section 1.2.2.1 "Shared Text Terminal". This is followed by the presentation of the third approach in Section 1.2.3 "extended Distributed Pair Programming (eDPP)", a way for overcoming the asymmetry of RPP by using a distributed editor, stand-alone or embedded in an IDE.

1.2.2.1 Shared Text Terminal

A lightweight, bandwidth-saving but also rather primitive variant for doing RPP is to share a terminal session with a terminal-based editor like *Vim* or *Emacs* (Kutner, 2013).

Terminal sharing is realized by using a terminal multiplexer such as *tmux*, GNU screen[22] and the like. Alice may start a terminal session and, using *tmux* and a *ssh* connection, Bob could connect to this session. Then both are connected to the same terminal and share the terminal viewport and the terminal's cursor. Thus Alice and Bob can write at the cursor's position and both see exactly the same contents.

However, terminal-based editors are optimized for mouse-less interaction and the 'mouse-grabbing' issue would turn into a 'cursor-grabbing' problem. Moreover, both participants need to be quite skilled concerning the text-only working style of the actual terminal-based editor and its native commands (Kutner, 2013). Editor-specific keybindings should not be underestimated as a hampering factor for equal contribution of the programmers and role switching: *"[...] so sometimes when I sit with Greg, you know, I can't type – he's an emacs user, I'm a vi user, I just can't put my fingers to the keyboard, I wouldn't know what to do"* (Chong and Hurlbutt, 2007). Such constellations resulted in sessions where one participant maintained the keyboard authority from start to finish (Chong and Hurlbutt, 2007).

[21] `https://www.eclipse.org/community/eclipse_newsletter/2016/december/article2.php` (accessed October 15, 2017)

[22] `http://www.gnu.org/software/screen/`(accessed November 11, 2017)

1.2.3 extended Distributed Pair Programming (eDPP)

"In reality, pair programming is a much more fluid collaboration, with both parties diving into the code as needed; remote pair programming should be the same. Any hinderance to the back-and-forth flow of coding should be eliminated. [...] 'It takes two to tango'. The tango is a fluid dance of two skilled people, moving together. The tango is not one person dancing, while their partner watches them dance on television, shouting suggestions"[23].

A way for eliminating the 'host'/'visitor' as well as the bandwidth issue of RPP is to use a dedicated distributed editor. A distributed editor is a groupware explicitly designed for the requirements of distributed real-time programming. Accordingly, it copes with workspace awareness issues and may be embedded in an IDE or be a stand-alone application. An example of such an IDE-integrated distributed editor is *Saros*. *Saros "enables a collaborative editing environment that works well with high-latency networks"* (Kutner, 2013).

With *Saros* or other dedicated DPP tools, Alice as well as Bob work on their local machine, each one having a local copy of the shared artifacts. Each of them controls his/her own keyboard and mouse and freely determines his/her viewport. Both can view and modify each file at will and at the same time, while each modification is readily transmitted to the partner and executed there as well: If Alice is typing, her changes appear in Bob's workspace and vice versa, when Bob is editing, this is transferred to Alice. Since the editing activities are performed locally, they are instantly visible in the respective local workspace: *"That's an important aspect of* Saros *because it means that both programmers will experience real-time write speed. There is no lag between pressing a key and seeing the change on our screen"* (Kutner, 2013).

The bottom line is that **both work in their 'home' environment, see each other's changes while having a consistent code base**. Figure 1.4 illustrates this by showing that Bob no longer remotely looks at Alice's machine but works on his local machine with his changes appearing in Alice's workspace and vice versa. Both always have a consistent copy of the jointly edited artifacts.

Due to the independent workspaces, this setup is different from doing PP in a distributed way. Compared to PP, **the distributed pair has extended interaction possibilities**. This is why this approach is referred to as **extended distributed pair programming (eDPP)**.

eDPP requires both participants to use the same or compatible tools and is less similar to local PP, but it **removes the asymmetry and imbalance of RPP**. The physical awareness is just as bad as for RPP and the workspace awareness is even worse: because Alice and Bob can independently move and edit in their workspaces, their views are not necessarily the same. This in turn means that some of the potential benefits of PP such as working like a coherent mind, having continuous reviews, and knowledge transfer, are lost to some degree. eDPP tools explicitly address the distributed situation and try to improve the workspace awareness situation, by providing relevant awareness information like the presence of others in the shared workspace, what files they are working on, or their viewport within a file. The relevance of video usage as a potential awareness source in DPP is briefly discussed in Section 1.4.5.1 "Efficiency of DPP" on page 75.

However, on the other hand, due to the independent workspaces, **Alice and Bob are less coupled and therefore more flexible** during their collaboration: Whenever necessary, they can move separately and join again in their shared workspace. This potentially enables new ways of collaboration and may relieve the pair from the PP-imposed work mode for inappropriate tasks (see Section 1.4.1.2 "Appropriateness of Pair Programming").

As discussed above and summarized in Figure 1.5, eDPP is the most flexible and fair approach for DPP. Neither Alice's or Bob's interaction possibilities are hampered. However, **the price for this flexibility is the reduced workspace awareness**. It is limited to the awareness information explicitly provided by the tool. Since they can independently switch files or move to different areas in their workspace, they

[23] Comment from Kutner `http://remotepairprogramming.com/post/66106200389/hi-joe-i-was-wondering-if-youve-worked-with-vs` (accessed November 11, 2017)

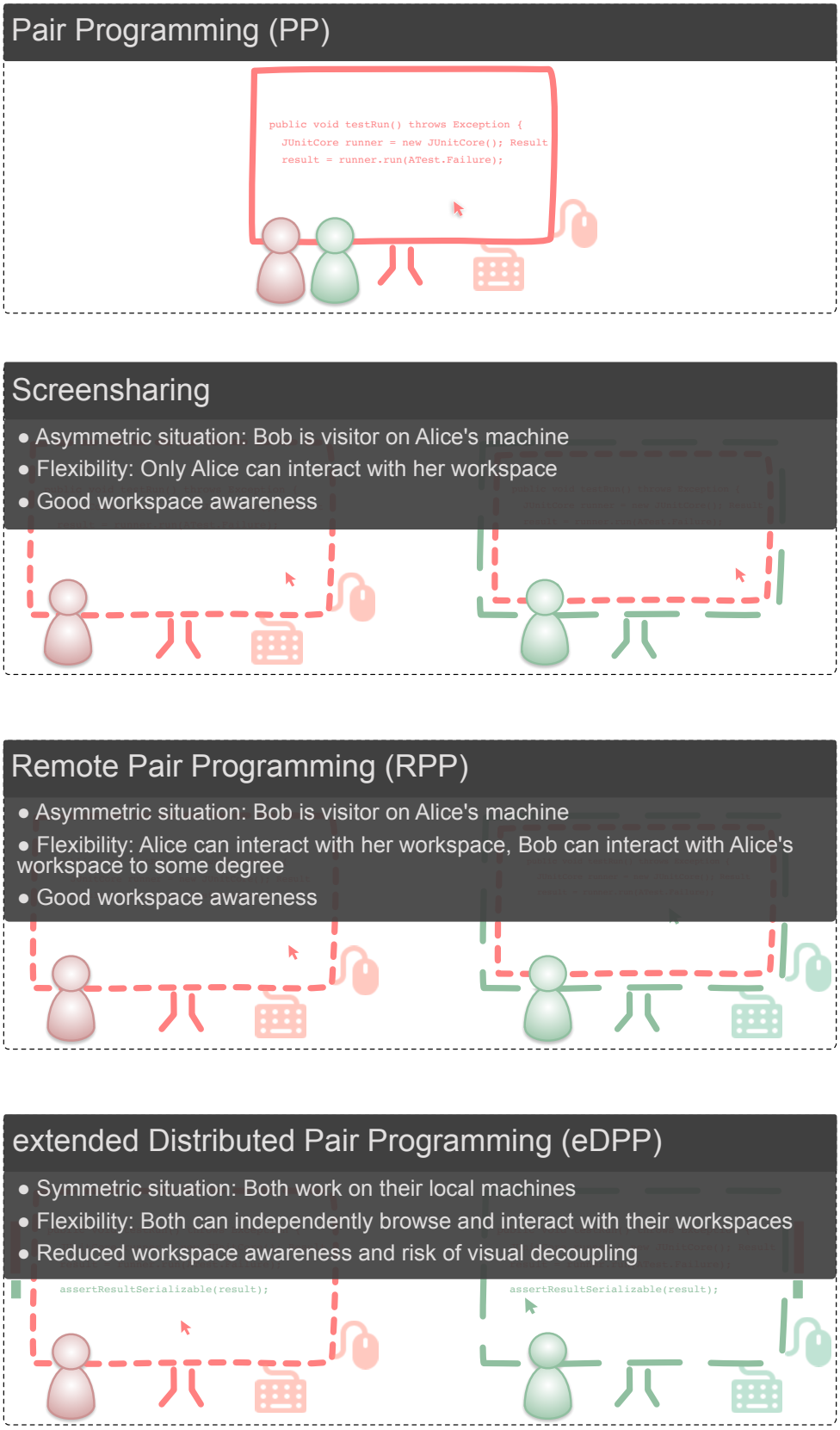

Figure 1.5: Approaches to DPP: workspace awareness and flexibility

face the risk of visual decoupling. So far it is not understood how **the reduced workspace awareness on the one hand and the increased flexibility on the other hand** affect the eDPP process. This takes us back to the fundamental question whether practicing DPP is a reasonable idea for virtual teams. This work examines precisely this issue by analyzing the eDPP process with regard to the reduced awareness and the increased flexibility. The respective research question is detailed in the following Section 1.3 "Research Question".

1.3 Research Question

Nowadays, environments for *"virtual co-location"* (Herbsleb, 2007) have several amazing features. However, their usefulness in particular with respect to awareness and communication challenges is hardly understood:

"Current research on collaborative IDEs is adding impressive capabilities, and tackling some important integration issues. Yet we do not know, at this point, how close these environments are to creating the natural, effortless kind of awareness and communication that happens in collocated settings. We should focus on creating an environment that puts people in virtual proximity so that those who do in fact need to coordinate and communicate can do so as naturally through the environment as they could if they had offices in the same hallway" (Herbsleb, 2007).

In addition to the fact that distributed teams do not have the choice between PP and DPP but rather to do DPP or nothing at all, also from this perspective, a quantitative comparison of PP and eDPP is not in focus of this thesis. With such an approach, research so far could not consistently report significant findings. In contrast, the **goal of this work is to better understand eDPP processes**.

Such an understanding in turn contributes to the overall practical research goal of helping virtual teams to successfully implement eDPP, which means to be able to give profound advice about 'how to make sensible use of eDPP'. This involves to sensitize participants to beneficial process behavior by providing a catalogue of behavioral patterns and anti-patterns. In addition to behavioral patterns, context factors for successful eDPP processes must be taken into account, such as type of task, working environment, or team and social setting. Finally, eDPP tools and environments must be appropriately adapted.

Although these aspects are touched upon, it is not the aim of this thesis to tackle this multi-layered overall goal. It rather addresses the beginning of the path towards that goal: understanding the current eDPP process and its fitness regarding awareness and interaction.

Awareness and the new interaction possibilities are quite evidently inherent specifics of eDPP. This prior knowledge or theoretical sensitivity (in terms of Grounded Theory methodology (Corbin and Strauss, 1996)) helped to find plausible starting foci for the data analysis, resulting in the following overall research question:

Which phenomena arise from

- reduced workspace awareness and
- independent editing capabilities

in eDPP processes?

1.3.1 Overall Research Goal and My Contributions

The research question of this work is of empirical-analytical nature: **understand eDPP processes with regard to reduced awareness and increased flexibility**. Figure 1.6 illustrates how my research helps to achieve the overall goal of supporting virtual teams to successfully do eDPP.

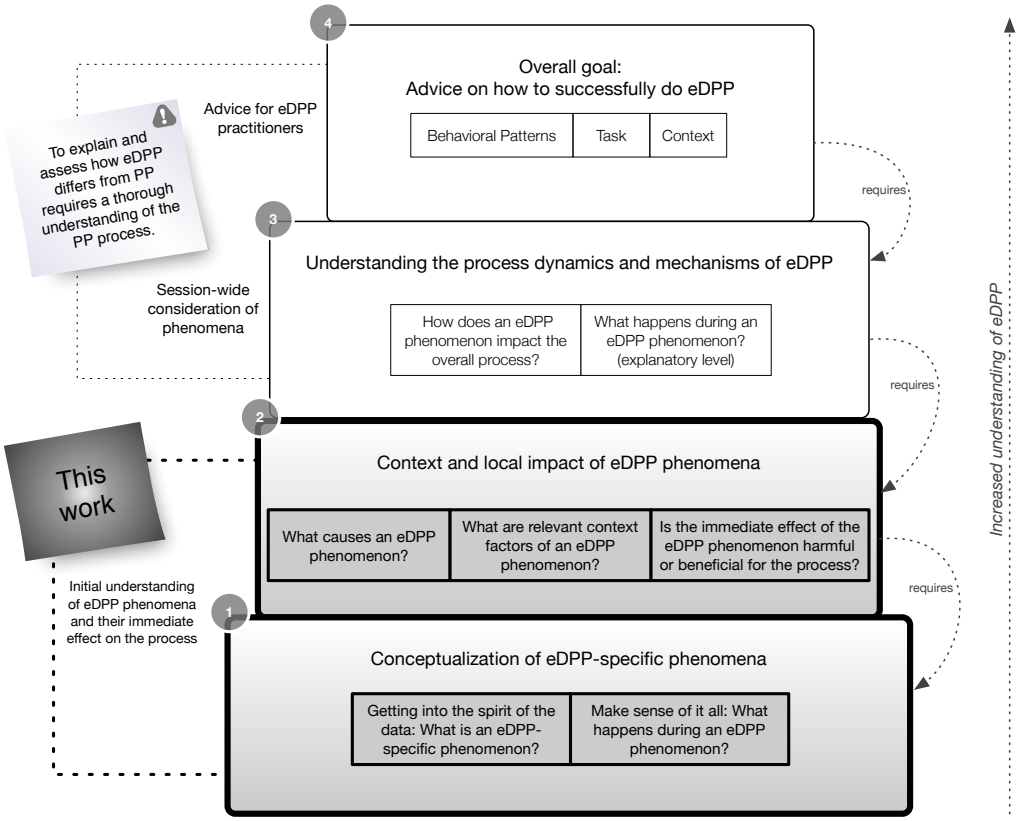

Figure 1.6: This work as a base for the overall goal

My research and further vital prerequisites for the goal to give profound advice about how to make sensible use of eDPP.

The pyramidal structure is used to highlight that each upper level substantially builds upon the underlying level of knowledge:

Level 1 (part of my work): This level is about developing a feeling for eDPP-relevant phenomena. The data is sighted and interesting events are identified. They are described with regard to 'what is happening there'. The phenomena are compared with each other to identify common or different characteristics so that concepts evolve. **Outcome of this level are types of eDPP phenomena alongside with their descriptive explanation**.

Level 2 (part of my work): On this level, apparently relevant context information of phenomena are conceptualized. Beyond that, a phenomenon's direct cause and immediate effect on the process or product is considered. Long-term or session-wide effects on the process are not taken into account. **Outcome of this level are concepts for relevant context information of eDPP phenomena as well as for their local cause and effect on the process**.

Level 3: On this level, the description of phenomena goes beyond an isolated treatment and is of explanatory nature. Phenomena are analyzed and their session-wide impact is considered, taking into account the session's goal, type of task, involved personalities, and other relevant factors. This requires to continuously analyze the pairs' interactions for the whole session, and not only when specific events occur. Therefore, it covers not only events that are specific to the distributed setting but all occurring phenomena. Finally, to understand how eDPP phenomena affect or change PP collaboration, i.e. which process dynamics are due to the eDPP setting, the dynamics of PP processes must be understood first, which is not yet resolved satisfactorily. **Outcome of this level is a holistic understanding of the dynamics in eDPP processes and the long-term effect of eDPP phenomena**.

Level 4: This level represents the overall goal this thesis is aligned to. At this level, **relevant aspects of how to successfully use eDPP are discussed**. This involves behavioral patterns, task types, as well as other contextual factors that appear to be relevant. As with the above, this requires a thorough understanding of PP processes and context factors.

Outlook: A level not shown in Figure 1.6 but that would also be above level 3 (parallel to the overall goal) is the outlook on some prospective ideas for further research in the area of eDPP. For example, the comparison of eDPP with other approaches like PP, extended pair programming (ePP), side-by-side (SbS) programming, or RPP. Depending on the desired character of the results, this level is divided into understanding the process differences or measure (and compare) effects of usage. However, looking at the effects does not provide explanations which in turn are necessary to be able to get traceable, repeatable results. Some initial ideas for further research projects are presented in Section 5 "Conclusions", formulated as conjectures.

extended Pair Programming (ePP) *(Explanation of terms)*

In the event of ePP, a pair does PP, sitting locally together, but collaborates by using an eDPP tool: they sit next to each other, each one views and operates his own computer but they have a joint virtual (and physical) workspace.

Side-by-Side (SbS) Programming *(Explanation of terms)*

"In side-by-side programming, two people sit close enough to see each others' screens easily, but work on their own assignments. [...] Then, they can work on their own respective assignments, but each can ask the other at any moment to look at a piece of code, run a test, or similar" (Cockburn, 2004).

In contrast to PP, the developers do not work on the same task but have each other within reach to ask for help.

As indicated in Figure 1.6, my research is located on the base level and with that it constitutes some ground work for the overall goal on level 3. **My research has to start at this base level because little is known about the mechanisms of PP** (see Section 1.4.1.3 "Pair Interaction in Pair Programming") and still **less is known about the eDPP process** and its process phenomena (see Section 1.4.5 "Distributed Pair Programming (DPP)"). As a consequence, the initial step for understanding what happens during eDPP is to grasp, characterize, and understand eDPP phenomena – a challenge which is tackled in this thesis.

The analysis of the recorded data yielded **central eDPP phenomena** which reflect local events attributed to awareness issues or independent workspace capabilities, **alongside with a context model** which captures attributes considered locally relevant. The respective attributes of the context model and their concrete manifestationas are described in detail, starting from Section 4.4 "Results Specifying Attributes of eDPP Phenomena".

For each phenomenon type I give a comprehensive, holistic picture: Based on the context model it is described, its immediate impact on the process is explained, and it is narratively presented and underpinned with examples (starting from Section 4.5 "Results Relating to an Interleaved Work Mode").

Clearly, overall this just scratches the surface of understanding what happens during eDPP. The set of identified phenomena also does not claim to be complete. **It represents an initial, data-grounded set of actual eDPP-induced behavior**.

As a further step in the direction of the overall research goal to provide advice on how to successfully do eDPP, the **behavioral patterns** of the observed pair that plausibly contributed to their successful collaboration are outlined in Section 5.1 "eDPP – Potential and Hazards" on page 203.

This inquiry is a **single-case study** and its limitations are discussed in Section 3.3 "Limitations and Threats to Validity". It **describes the smooth usage of eDPP** by one quite eDPP-experienced pair and with that it reveals the potential of eDPP. Hopefully, the unexpected positive results arouse interest and inspiration for further research projects. In the long run, these should provide helpful insights about how and when to practice eDPP and further improve eDPP tools.

To be able to conduct my research project, an industry-ready eDPP tool was needed. As a consequence, an additional **contribution of my research was the further development of such a tool**. The scope of necessary preliminary work is outlined in the next Section 1.3.2 "Tool-Related Contributions". The concrete tool and its history is presented in Section 2.3 "Saros".

1.3.2 Tool-Related Contributions

In 2010, when my research started, no established, stable, and industrially usable eDPP tool was available. The most comprehensive[24] eDPP tool at this time with regard to awareness information, and supporting a non-restricted interaction for the participants was *Saros*. However, **Saros was neither in a stable, industrially usable state nor were there any noticeable or even professional users**.

The claim of my research project was to examine industrially usable eDPP of professional developers. Before I could get going, the following basic conditions needed to be accomplished:

1. An industrially usable eDPP tool is needed. Industrially usable means the tool has to be
   (a) useful, reliable and
   (b) usable.
2. The tool should not be an unsteady testbed. As every application, it may shape the collaboration by its provided technical capabilities, but apart from that the tool itself should not come to the fore.

[24] see tool discussion in Section 2 "eDPP Requirements and Tools"

3. To not consider the study a tool study but an eDPP study, the tool should be state-of-the-art for eDPP.
4. The tool has to be used by professional developers. They must be active users who frequently use it and are familiar with it.
5. Users willing to participate in the study are needed. This means that not only the developers but also their company has to agree to record their eDPP sessions (their audio, video and screen). In particular the latter is a sensitive issue with regard to business secrets.

These requirements demonstrate the **big gaps of the actual and the desired situation that needed to be closed before I was able to start with the actual research**. Since *Saros* is developed at the same working group I was part of, I had the chance to work on these requirements.

Saros is a plugin for the *Eclipse* IDE that enables real-time distributed software development with up to five participants. A detailed description of *Saros*, its evolution and differentiation to other eDPP tools is given in Section 2.3 "Saros". When I started my work, the development of *Saros* was characterized as follows:

- *Saros* is an open source project. It is mainly realized by members of the Software Engineering research group at Freie Universität Berlin[25] and student theses[26].
- *Saros* grew evolutionarily. By and by, features and improvements were added without adequate automated testing or user testing.
- There is a high fluctuation of student team members. Students writing their theses had limited time to become familiar with *Saros'* architecture and source code and left when their thesis was finished. Accordingly, the development of *Saros* involved many newcomers and was far from straightforward.
- *Saros* had to struggle with recurring problems concerning internal quality like reliability, efficiency, and maintainability.
- Usability and the user perspective were actually not considered during the first years of development.
- Moreover, the problems with the internal quality negatively influenced the usefulness of *Saros*. This in turn has led to a bad overall user experience, and consequently no real users appeared.

Facing these issues, significant improvements of *Saros*' stability and usability were necessary to improve it to such an extend it could serve as a testbed for this inquiry. The first aspect, stability, comprised the revision and integration of features and the overall architecture and was the responsibility of the technical project leader. **I was the person in response for *Saros'* usability and user experience**. I supported all team members from the usablity / user experience perspective, initiated and worked on usablity / user experience topics and supervised 16 student theses. All activities followed an iterative, user-centered approach by selecting appropriate methods like heuristic evaluations, user-interviews and usability tests.

In summary, I worked on the following overlapping and inter-related areas of *Saros* which will be described in more detail in the following paragraphs:

- Improve the out-of-box experience and general usability
- Improve the interface- and interaction design
- Revise and extend the awareness information provided by *Saros*
- Improve user friendliness in terms of understandability of *Saros'* usage concepts
- Provide usability advice for developers

[25] `http://www.inf.fu-berlin.de/w/SE/WebHome`(accessed November 11, 2017)

[26] For more information of *Saros'* development history see `http://www.saros-project.org/history` (accessed November 11, 2017)

- Be an interim project manager for the *Saros* project
- Relaunch of the *Saros* website
- Work on *Saros'* outreach

In summary, after about 3 years of intense work on stability and usability issues, the first fruits of these efforts showed up. *Saros* became stable and reached industrial strength. It could be considered as a state-of-the-art eDPP tool and ultimately several users appeared. They contacted the project for problems, questions, suggestions, or compliments. One of these contacts agreed to be recorded while using *Saros*. Finally, **after 3.5 years of focusing on tool-related work, the data for my research was going to be available**.

1.4 Related Work

The research area of DPP, i.e. the real-time collaboration of distributed pair programmers, encompasses the scope of the thesis at hand. All in all, research of the following fields is of direct relevance to my own work:

- **DPP**
- **PP** $\rightarrow$ pair programmers
- **CSCW** $\rightarrow$ (computer mediated) real-time collaboration
- **GSD** $\rightarrow$ distributed

Agile software development, GSD and CSCW alone involve a huge spectrum of related topics that are in general of interest to this work. However, most of them are not directly relevant for the specific issue of this inquiry or do not apply at this stage of research. Consequently, in fact little research is directly connected to the immediate, synchronous collaboration process of distributed programmers.

1.4.1 Pair Programming (PP)

Before focussing on PP in a distributed setting, this section provides an introduction to PP and an overview of the most prevalent research areas.

1.4.1.1 Issues and Benefits of Pair Programming

Problematic aspects of pair programming When doing PP, developers move from working individually – self-determined concerning working style and speed – to a quite tight collaboration with a partner. This requires communication, reconcilement of opinions as well as coordination of work. Such a change in working style can lead to skepticism and reluctance in the first place, since *"most programmers are long conditioned to working alone"* (Williams and Kessler, 2000). This form of tight collaboration is quite an intense work mode requiring heavy concentration. Therefore it is exhausting for the participants. (Williams and Kessler, 2000)

Apart from the personal sensitives of the developers, it appears that the customer or the project manager has to pay two developers for doing the same thing which could be done by a single person just as well (Cockburn and Williams, 2001). Although this way of looking at PP is rather shortsighted, it reveals the intuitive points of criticism to PP: *"pair programming has also received criticism over increasing effort expenditure and overall personnel costs, and bringing out conflicts and personality clashes among developers"* (Hulkko and Abrahamsson, 2005).

Consequently, the **main starting points research has to address are cost-effectiveness of PP and the pair interaction**. And although there has been a significant amount of empirical research concerning the efficiency of PP, i.e. Cockburn and Williams (2001); Dyba et al. (2007); Hulkko and Abrahamsson (2005); Nawrocki and Wojciechowski (2001); Nosek (1998); Sillitti et al. (2012); Williams and Kessler (2000); Williams et al. (2000), **there is no satisfactory trade-off validation of PP**:

"However, at present, we do not have a clear, objective, and quantitative understanding of the claimed benefits of such development approach. All the available studies focus on the analysis of the effects of Pair Programming (e.g., code quality, development speed, etc.) with different findings and limited replicability of the experiments" (Sillitti et al., 2012).

Nevertheless, **in general, research results draw a positive picture** concerning several beneficial effects of PP (Hulkko and Abrahamsson, 2005).

Product perspective: better product quality In PP research, various metrics were used as indicators for product quality. Accordingly, often the construct validity is problematic and the results are hardly comparable (Hulkko and Abrahamsson, 2005): *"Quality was typically reported as the number of test cases passed or number of correct solutions of programming tasks, but student grades, delivered functionality, and metrics for code complexity were also used as measures of quality"* (Hannay et al., 2009). Apart from that, studies were mostly conducted in university settings, few were experiments with professionals, and all varied in duration and kind of task. Notwithstanding, research shows that PP has a rather positive effect on product quality (Hannay et al., 2009).

In an experiment with professional developers, Nosek (1998) compared the readability of solutions done in pairs vs. that of solutions done by individual developers. All code implemented by pairs got a significantly better grading than the code done by individuals. Williams et al. (2000) did a university experiment with thirteen individuals and fourteen pairs. The students had to do four assignments over a period of six weeks. The programs of the pairs passed about 15% more test cases than the ones developed by the individuals. A meta-analysis of fourteen studies by Hannay et al. (2009) concludes that, except for one case, all studies show a **small to medium positive effect of PP on product quality**.

As explanations for the positive effect of PP on product quality, the following reasons are mentioned:

- **Simpler code**: Code produced by pair programmers is shorter and simpler and therefore easier to understand and to maintain. Beyond that, code which is simple and easy to understand is an indicator for a better underlying design. (Cockburn and Williams, 2001; Hulkko and Abrahamsson, 2005)
- **Fewer defects**: The observer continuously reviews the code while the driver is typing (Cockburn and Williams, 2001), therefore *"defects are prevented or removed almost as soon as they hit the paper"* (Williams et al., 2000). Besides, post-reviews are often not esteemed or neglected due to time-pressure (Cockburn and Williams, 2001). However, even if post-reviews are done, real-time reviews during PP still outperform downstream reviews (Cockburn and Williams, 2001; Hulkko and Abrahamsson, 2005).
- **Better design**: The communication when working in pairs encourages a powerful cross-fertilization of ideas. As a synergy effect, more than two approaches for a solution emerge and are evaluated (Williams and Kessler, 2000). The decision making is more flexible and open-minded (Bryant et al., 2008) and the *"'design tunnel vision', which occurs when one makes a design decision and sticks with it no matter what"* (Williams et al., 2000), is less likely. At the same time, pairs quickly take a decision for one solution (Williams and Kessler, 2000).
- **Increased compliance with standards**: The inherent pair pressure of PP results in higher discipline concerning the adherence to coding conventions and process standards since *"working under the gaze of a critical colleague may result in positive changes in behavior"* (Bryant et al., 2008).

Programmers' perspective: more joy and satisfaction In university as well as in professional settings, a significant majority of developers stated that they actually like to do PP (Williams et al., 2000) and that they prefer to work in pairs, even if before they were used to working alone (Cockburn and Williams, 2001). The following three aspects were named:

- **Joy during PP**: Developers enjoy working with a programming mate more than programming alone (Cockburn and Williams, 2001; Nosek, 1998). *"In a survey of professional pair programmers, 96% stated that they enjoyed their job more than when they programmed alone"* (Williams et al., 2000).
- **Confidence in solution**: Developers are more confident in code that has been collaboratively developed than in code they have done without a partner (Cockburn and Williams, 2001; Nosek, 1998; Williams et al., 2000). Problems which would be very difficult or even impossible to solve alone may be solved with greater ease in pairs (Dyba et al., 2007).
- **Job satisfaction**: PP enhances the subjectively perceived job satisfaction of the developers (Williams et al., 2000).

Team perspective: learning, communication and trust The intense interaction and communication during PP fosters knowledge transfer between the participants: *"Learning happens in a very tight apprenticeship mode. The partners take turns being the teacher and the taught, from moment to moment. Even unspoken skills and habits cross partners."* (Cockburn and Williams, 2001)

The bidirectional flow of information spreads knowledge on different levels (Cockburn and Williams, 2001):

- **Code base**: knowledge about the code and its relationships: *"Professionals don't create knowledge silos. Rather, they learn the different parts of the system and business by pairing with each other. They recognize that although all team members have a position to play, all team members should also be able play another position in a pinch"* (Martin, 2011).
- **Working environment**: Tips and tricks concerning the technical development environment.

- **General programming and design skills**: Knowledge concerning things like good design or *clean code*[27] as well as concrete programming language skills.
- **Programming rules and conventions**: general or team-specific coding standards or conventions.
- **Task-specific knowledge**: Domain knowledge or knowledge about specific requirements.

Learning can be the main goal of a PP session. However, during sessions that aim at producing code, the pair members also exchange thoughts and knowledge and thus learn from each other (Zieris and Prechelt, 2016). Consequently, when pairs vary in the team, the knowledge scatters in the team as a side effect. This results in a team with more homogeneous competence and code familiarity. From the management perspective, this is positive with respect to the *winning-the-lottery* factor[28] in a project.

Beyond that, through the intense collaboration, people learn to know and to trust each other (Cockburn and Williams, 2001; Zieris and Prechelt, 2014), and this also in times when the developers do not work in pairs. That generally increases communication willingness and is an important factor for example to lower the inhibition level for making issues visible, which involves reporting and talking about problems or asking others for help.

Economic perspective: efficiency of pair programming PP must pay off – so effectiveness is a critical success factor that decidedly influences the use of PP in industry. Unfortunately, research findings differ on this issue.

Studies concerning the effectiveness of PP in relation to solo programming either compare the time spent on a task (or a set of tasks) or the effort in person-hours (summed up for pairs) on individual or on team level.

Here, also the aforementioned problem of the different study environments applies. The diversity with regard to participants, tasks, settings, and *"other moderating factors, such as amount of training in PP, motivation, team climate, etc., are likely to be relevant as well"* (Hannay et al., 2009) and presumably are part of the problem of the mixed to inconsistent research results.

For example, in a university experiment of Williams et al. (2000), students working in pairs needed less time for their task as a pair, but when summing the time of the two pair members, they needed 60% more person-hours for their first assignment. After this phase of adjustment, the overhead of person-hours diminished to an additional time effort of 15% in total. This number is congruent with the findings of Nosek (1998) who made an experiment with 15 full-time programmers (5 worked as individuals, 10 as pairs).

A meta-analysis of Dyba et al. (2007) reviewed 15 studies that compared the effort of PP to individual programming. The number of participants was between 12 and 295, with a median of 24. Their overall conclusion is that projects developed in PP-style are finished faster than projects done with solo programmers (medium-sized reduction of development time) but in total require more person-hours (medium-sized increased effort). The phenomenon that the productivity of the sum exceeds the sum of the individual's productivity is sometimes referred to as *"pair jelling"* (Williams et al., 2000) and is a determining factor for decreasing the overall duration of a project (not its effort in person-hours) (Hulkko and Abrahamsson, 2005).

In any case, when weighing benefits and costs of PP, it is too simplistic to compare only the time individuals or pairs spend on a task or to sum the effort in person-hours. PP avoids downstream costs due to lower code quality, like when removing defects at a later stage (Cockburn and Williams, 2001), having code that is hard to maintain or designs that hardly scale. **When adding the time solo-programmers would need to raise their code to the quality of that done by pair programmers, *"the individuals would take even more time than the pairs"*** (Williams et al., 2000).

[27] `http://cleancoders.com/` (accessed November 11, 2017)

[28] The smallest number of team members who may win the lottery and leave your company before the project would be in serious trouble.

As the mixed evidence concerning the efficiency of PP indicates, little can be concluded from the quantitative examination of PP detached from setting, task nature, or participants' context. The next section addresses this dilemma by discussing the usefulness of PP in specific contexts.

1.4.1.2 Appropriateness of Pair Programming

"In addition to evaluating, if PP is beneficial, it should be also considered, when it is most useful" (Dyba et al., 2007). In order to get **meaningful insights regarding the benefits and efficiency of PP, several aspects have to be taken into account**, like for example the appropriateness of PP for a situation or task. The following aspects regarding the usefulness of PP and specific contexts have been addressed so far:

- **Kind of task**: The collaborative implementation of tasks is considered useful for complex, non-routine tasks, since two *"heads are better than one for achieving correctness on highly complex programming tasks"* (Dyba et al., 2007). For simple tasks developers often prefer to work alone. Pairs stated that they found it most useful to do PP for analysis and design tasks. In contrast, they prefer to be undisturbed to *"individually read and fully understand the problem they need to solve, think about complex logical problems, and do experimental prototyping"* (Williams et al., 2000). Sometimes pairs run test cases autonomously but within touching distance, and in case of detecting defects they pair up to jointly solve the problem. (Plonka et al., 2012b; Williams et al., 2000)

  However, limiting the usefulness of PP to non-trivial tasks is problematic: The difficulty lies in the transition between trivial and non-trivial tasks and the classification of a task concerning this matter, actually before completion and gaining full understanding of it. In this context, it is a matter of particular interest to which proportion software development involves trivial tasks, as here the reuse of existing solutions is common and reasonable. Moreover, in particular tasks considered to be 'no-brainers' run the risk of being underestimated and that the developer doing it is not careful enough or switches to 'auto-pilot'.

- **Phase in software development life cycle**: Pair programming was more often applied during a project's start phase to gain orientation and understanding of the system (Hulkko and Abrahamsson, 2005; Williams et al., 2000) than in later project phases. It is questionable, however, how far this actually reflects the usage of PP in professional contexts.

- **Goal of collaboration**: When learning is the main goal of a PP session, it should not take place under time pressure, otherwise the learners stated that they felt to thwart the expert. They stopped asking questions and therefore were rather passive in the collaboration. When developers paired-up to combine their complementary competence for solving a task, it was observed that they split up for sub-tasks according to their competence. While one of them solved the subtask, the other developer 'opted out' of the collaboration to some degree. In such sessions, the pair looses the benefit of peer-learning and real-time peer-reviewing but without considering this harmful. (Plonka et al., 2012b)

- **Personality of participants**: Personality traits like style of communication, interpersonal behavior, degree of dominance, and alike are plausible factors for shaping the collaboration and interaction of a pair. However, research examining the impact of personality for PP could not statistically significantly verify that personality type or conflict handling ability is a crucial factor for pair success. An experiment with 564 students of different graduations showed *"that students prefer to pair with someone they perceive to be of similar technical competence"* (Katira et al., 2004). In two controlled experiments with 84 undergraduate students, Sfetsos et al. (2006) found that pairs of developers with mixed Keirsey[29] temperaments significantly outperformed the pairs with similar temperaments with regard to more intense communication and conceived quality of the collaboration (developers' satisfaction, knowledge acquisition, and participation). These

[29] `http://keirsey.com/cgi-bin/keirsey/kcs.cgi` (accessed November 11, 2017)

findings indicate the relevance of the individuals' mindset, behavior and skills. For example, this is also apparent in the PP principle 'don't take things too seriously – being open-minded during collaboration' (see Section 1.4.1.4 "Principles for Good Pair Programming"), which addresses the inappropriate insistence on one's own opinion. The actual manifestation of such traits and their impact in PP can hardly be measured by general personality categorizations. Actually, their relevance and impact in PP is not yet understood. In that regard, more research like that of Salinger et al. (2013) or Salinger and Prechelt (2013) aiming at an understanding of the PP process and its specific interactions is necessary as a base to decrypt, among others, these aspects. The next section presents the little research that has been done so far in that direction. However, it mainly aims at a better understanding of the roles in PP.

1.4.1.3 Pair Interaction in Pair Programming

Looking at the effects of a pair's collaboration does not provide explanations of what makes a pair efficient or inefficient. To take a step forward in this matter, it is important to understand the collaboration process itself – the dynamics of the pair interaction: the casual relationships of the interactions of the pair members and their consequences for the process. In such a way, an understanding about good and problematic behavior in PP can be gained. The following paragraphs address the little research that examines PP from this perspective.

Concerning the distribution of roles, some developers mentioned that they prefer to be the driver, while others prefer to be the observer. The reason for preferring to be the driver is that they would become impatient otherwise. Developers who favor to be the navigator state that they feel uncomfortable when they are observed during editing. Interestingly, PP experience seems to be an important factor for role preferences because all developers who preferred to be the navigator had less than six month of PP experience. (Plonka et al., 2012a)

A further aspect for role allocation seems to be the **machine ownership**: *"Developers stated that the developer who 'owns' the computer is more likely to be the driver. Furthermore, developers reported that they feel more efficient and comfortable when driving their own computers and that an unfamiliar computer slows them down"* (Plonka et al., 2012a).

Beyond that, the distribution of roles has an impact on **decision making** – the **driver is more dominant** (Plonka et al., 2011).

As discussed in Section 1.1.1 "Pair Programming – a Common Agile Practice", the assumption that the observer thinks on a strategic/high level while the driver is concerned with the implementation details has proven to be false. Rather, the pair jointly glides through to the different abstraction levels of thinking. To examine the individual abstraction levels of thinking, the abstraction levels of the pair members' utterances have been analyzed. Salinger et al. (2013) went a step further by not characterizing the abstraction level of communication units between the participants but analyzing the content of the individual verbal contributions of the pair members. They determined several nuances of the observer role with far more complex properties than the original characterization.

Salinger and Prechelt (2013) provide a language of its own to describe interactions during the PP process. The so-called *base layer* aims at understanding the PP process and provides a vocabulary to break up the PP process interactions through named concepts *"describing directly observable communication events, activities pertaining to the computer, and activities pertaining to the rest of the work environment"* (Salinger and Prechelt, 2008). It may be considered as a corner stone for further research on the mechanisms at work during PP. This in turn can help to formulate sound advice in terms of behavioral patterns and anti-patterns for doing PP. Nevertheless, albeit on a general level, some constructive hints for good PP already exist. They are presented in the next section.

1.4.1.4 Principles for Good Pair Programming

As mentioned in the last section, little is known about PP skills – to be able to define a good pair programmer. Williams and Kessler (2000) as well as Kutner (2013) provide some practical-oriented principles for good PP. They are summarized in the following.

- **Share everything – equality of vision**: Both partners have to enjoy equal rights in the collaboration with respect to seeing what is happening in the shared workspace. None of the participants should do anything that the other cannot see. In PP, this requirement is realized by using the same computer, but it is becoming an issue when doing DPP.
- **Play fair – equality of control**: Both partners should actively contribute to the process and, independent of expertise and PP experience, driving should not be predominated by one partner. In DPP, this affects the technical equipment: the tools used should give both participants equal capabilities of control.
- **Be comfortable – cherishing the shared time**: Both partners should handle their own as well as the partner's time with respect and create a physical and technical environment where both feel comfortable and can easily collaborate and communicate. For example, a partner should not have to wait for the other one who might be making extensive private phone calls.
- **Self-doubts away – no negative self-perception**: Participants stated that they found it hard to collaborate with someone who has no confidence in his programming skills.
- **Clean up your mess – no ignorance of problems**: The four-eyes benefit concerning defect detection should be valued by effectively eliminating defects that have been identified.
- **Wash your hands before you start – being open-minded to pair work**: The partners should not start collaborating with a general reluctance or skepticism to PP. They should believe in the success of the pair and the joy and efficiency of working as a pair.
- **Do not take things too seriously – being open-minded during collaboration**: Both developers should be open-minded for suggestions, opinions and alternative working styles and should not waste time by endless discussions about them.
- **Flush – reviewing work done individually**: When integrating work that has been done independently in their collaborative work, the partners should review or even rewrite it.
- **Stop when you are tired – taking breaks from work**: The continuous concentration during PP is quite demanding for both participants. There should be time for breaks in which something is done which is not related to the task.
- **Pause pairing – taking breaks from doing PP**: There are types of tasks that programmers stated they prefer to do alone (see page 49). It is okay to work alone for 10%-50% of the day.
- **Live a balanced life – socializing**: Life is more than just going to work and doing PP. To be a valuable and fit partner, people should exchange ideas with other developers than their PP partner and they should pay attention to their work-life balance.
- **Be responsible as a pair – no blaming each other**: Both partners are equally responsible for every aspect of the work product, also for problems.
- **Be aware of wonder – effects of brain synthesis**: When undertaking PP, there is a synergy effect especially in problem solving that leads to more than just the summation of the two individuals' solutions.

This section gave a rough overview of the prevalent PP research areas and their findings. Unfortunately, most research focused on measuring the efficiency of PP and was conducted with students, not professional software developers. Plausibly, the mixed results stem from not taking into account the participants' *"PP skill"* (Zieris and Prechelt, 2014). The PP process is treated as a black box although

the human or interpersonal skills determine the participants' interactions and shape the course of the PP process. However, without such an understanding, effects can hardly be explained and therefore not controlled. For PP, the situation improves, in particular with the work of Salinger and Prechelt (2008) and Zieris and Prechelt (2016) who analyze and conceptualize the interaction episodes in the PP process. For DPP, things are worse – some research measured the efficacy of DPP but none investigated what happens during the DPP process in terms of pair interaction (see Section 1.4.5 "Distributed Pair Programming (DPP)").

Altogether, the preceding section helps to understand the strengths and weaknesses of PP as well as the weaknesses of PP research so far. This in turn facilitates understanding the research approach of this work and assess the results of this work.

There is a strong tendency to scale PP to distributed settings without, however, having a fundamental understanding of the local PP process.

Before turning to the technical challenges for distributed collaboration in Section 1.4.3 "Computer Supported Cooperative Work (CSCW)", the next Section 1.4.2 "Global Software Development (GSD)" discusses the people-related aspects and problems of distributed teams.

1.4.2 Global Software Development (GSD)

"Global Software Development stems from the Software Industry's desire for globalization of business to derive increased market-share and from its' desire to gain a competitive edge through outsourcing, subcontracting and forming strategic partnerships. Rapid advances in computer networks, telecommunication and internet technologies have made it possible for developers from different geographical locations and technical specialities to form virtual teams in a distributed setting. This allows teams to jointly develop the same artifact of software in a collaborative way. However, GSD represents a radical shift from the way software is engineered traditionally in a collocated team setting."(Reeves and Zhu, 2004)

GSD relies on technical means for realizing the collaboration of the physically separated team members (Baheti et al., 2002a), and, as such, it involves the aspects of technology for group collaboration and workspace awareness (that will be discussed in Section 1.4.3 "Computer Supported Cooperative Work (CSCW)" and Section 1.4.4 "Workspace Awareness in Real-Time Groupware"). Beyond that, GSD research looks at the team aspects and software-development-specific issues of remote team collaboration. The GSD-inherent distributed situation involves aspects like different nationalities, cultures, languages, countries or sites. These lead to potential benefits but also to a number of strong challenges for virtual teams and management. In this section both sides of the coin will be examined.

GSD in general encompasses several areas of research, each of them in turn involving far-reaching topics and sub-topics and thus a lot of interesting research areas in that relatively new domain. In order not to loose focus, this section sticks with an overview of the most prevalent aspects of GSD that are important to become aware of the integration and relevance of eDPP in GSD. It will thus not discuss every raised GSD topic nor its various solution approaches in detail. Before shedding light on the downsides of GSD, this section starts with the potential of GSD as well as with the motivating factors for companies to do GSD and for developers to work in distributed teams.

1.4.2.1 Motivation for doing GSD

Driving factors of GSD are that the globalization has significantly affected software development industries (Ray and Samuel, 2016) and that *"the current dynamic business environment requires organizations to develop and evolve software systems at Internet speed"* (Ramesh et al., 2006).

Apart from these 'market-imposed' factors, GSD involves some other economic benefits: Companies may *"benefit from favorable governmental policies and tax subsidies"* (Jain and Suman, 2015). Furthermore,

successful, innovative products and services require skilled staff with different expertise and good knowledge about the *"domain, context, and constraints of a product"* (Cooper et al., 2012) as well as the target user group's specifics. In that regard the location-flexibility with its possibility of closeness to local markets and customers is an important aspect for companies.

In times of skills shortage and price and time pressure, from the company's perspective, GSD can solve nearly all of their most evident issues. Wage differences between countries allow to reduce staff costs. The location-independence in GSD teams significantly increases the pool of skilled staff with diverse expertise. Due to different time zones, farshoring facilitates quicker development by round-the-clock development or shifted working hours. However, it is quite sensitive to time zone problems and cultural differences. (Herbsleb, 2007; Jain and Suman, 2015; Mockus and Herbsleb, 2001; Nguyen et al., 2008)

For developers, GSD not only provides the chance for their own location-flexibility, it may also render their work more exciting. They have the possibility to travel to other countries, speak other languages, and collaborate with different nationalities and cultures. These factors are assumed to have a positive impact on the developers' intrinsic motivation (Beecham and Noll, 2015). Since satisfied and highly *"motivated engineers working in distributed software development have a significant influence on project success"* (Beecham and Noll, 2015), this aspect is not to be neglected.

1.4.2.2 The Downsides of GSD

What appears like paradise on earth for employers and employees also involves big challenges and uncomfortable efforts, resulting in a doubtful effectiveness of GSD: The round-the-clock development turbo did not ignite. Moreover, *"research in the last decade has consistently found that distribution has a negative impact on collaboration in general, and communication and task completion time in particular"* (Nguyen et al., 2008). Apart from that, the following questions are still not finally answered: *"why does interval of the distributed projects appear to be longer than interval of a comparable co-located project? Why do the costs of a distributed project not go down according to lower expenses associated with some of the locations involved?"* (Mockus and Herbsleb, 2001)

In addition, GSD is a young and immature phenomenon which *"introduces a great deal of complexity to an already complex process"* (Treinen and Miller-Frost, 2006) and thus opens a broad field for investigation. Herbsleb (2007) demands for a better understanding of several aspects in GSD and the need of understanding their interplay and context to have general valid insights:

"We currently have a number of individual solutions, such as tools, practices, and methods, but we understand as yet very little about the tradeoffs among them, and the conditions of their applicability. If we work toward compatible processes across sites, can we reduce the amount of communication? If we carefully design architectures to isolate work at different sites, can we get away with incompatible processes? We currently have very little to go on for addressing these crucial questions. We have a pressing need for good theories that will provide a sound basis for reasoning about tradeoffs and predicting outcomes." (Herbsleb, 2007)

Finally, the entire 'GSD-ecosystem' with its context, culture, social interaction, personal limits, behavioral patters and human diversity has to be better understood. Based on that, tools, process, and practices can and should be adapted to best serve the needs and challenges of GSD (Nguyen et al., 2008). Hence, when the ecosystem's 'inhabitants' have to use improper tools and processes or have to live in inappropriate conditions, effects like inharmoniousness, uncomfortableness, inner refusal, and alike are likely to appear. That in turn can manifest in inefficient work and communication.

The next paragraphs discuss the **three dimensions of distance in GSD – geographical, socio-cultural, and temporal distance**. They reflect the current state of knowledge about the basic challenges of GSD and their roots. The distances are nonexistent in collocated teams (Holmstrom et al., 2006a) but hamper trust building within dispersed teams and have other problematic consequences. Even though socio-cultural distance can also happen to some degree in a collocated team, it is a lot easier to handle than in a distributed setting.

Due to **geographical distance**, dispersed team members, if at all, meet less often than collocated team members (Moe and Šmite, 2008). Informal or ad-hoc communication or spontaneous collaboration does rather not happen (Herbsleb, 2007). Interestingly enough, with regard to recession of communication and collaboration, there is no significant difference whether teams reside on different continents or whether their offices are about 30 meters or more apart (Herbsleb, 2007).

Temporal distance exists when the team members are in different time zones, thus reinforcing the effects of geographical distance. The hurdle for synchronous communication and collaboration, even mediated by technology, is higher since the team members' working hours do not fully overlap. To eliminate this issue and to reduce the geographical spread, nearshoring has become a GSD-variant that is increasingly used (Tjørnehøj et al., 2012).

Socio-cultural distance refers to the diversity of team members of different countries and cultures and it affects many facets of team collaboration: *"Culture influences the common understanding between teammates due to diversity in people's assumptions, behaviors, expectations about leadership practices, team norms, attitudes towards hierarchy, sense of time, and communication styles"* (Moe and Šmite, 2008).

These distances come into effect in nearly all aspects of software development like processes and practices, requirements and risk management, process measurement and control, and tools. After all, **the most affected factor is trust** or trust building, which has a significant impact on collaboration, coordination, and communication in the team and thus in turn affects the aforementioned points.

1.4.2.3 Lack of Trust in GSD and its Effects

Trust between team members is the believe in the other one's expertise and attitude. It refers to the willingness to take the risk of being dependent on that team member with regard to important activities or work results (Moe and Šmite, 2008). It **influences how others' activities are assessed, interpreted and how expectations are set** (Tjørnehøj et al., 2012). Reciprocally, this disposition **impacts personal willingness to invest in the social and professional interaction and collaboration with others**.

Trust *"is believed to be a fundamental factor for the success or failure of virtual teams"* (Moe and Šmite, 2008) because it is vital for a strong and effective communication and feedback culture in a team and thus for the performance of the team (Moe and Šmite, 2008). In a team, however, trust is not a matter of course, it has to develop. In particular at the beginning of a cooperation, the feelings, attitudes, and opinions regarding other team members are rather irrational, based on the impressions from the first few contacts or on hearsay from others – this is so-called **affective trust** (Tjørnehøj et al., 2012). It is important that these preconceived notions are replaced by more rational or substantiated perceptions so that **cognitive trust**, *"relying on rational assessments of the other party's integrity and ability"* (Tjørnehøj et al., 2012), can evolve and remain (Tjørnehøj et al., 2012).

Building and maintaining trust is more difficult in geographically dispersed teams than in collocated teams. In contrast to the need of intense communication and continuous assessment of the others' integrity, in GSD, team members, if at all, meet seldom and only for a limited time and for the rest, their communication happens via technical means (Tjørnehøj et al., 2012). Teams from other sites do not take part in the informal communication of the members that are co-located. They are not involved in the conversations before or after a technically mediated meeting and are not *"privy to other team members' reflections, conclusions or resolutions"* (Al-Ani et al., 2013). Regardless of that, technically mediated communication by far does not provide the same richness and atmosphere as face-to-face interaction and cannot take place spontaneously like for example when meeting in the aisle or at the coffee machine. Time differences intensify the issue of meetings for distributed teams.

Socio-cultural distance hampers communication and collaboration because of differences in communication culture, social norms, meanings of gesture and facial expressions, attitudes to hierarchies, work practices, punctuality, values in social and work life, and alike (Moe and Šmite, 2008). When

such aspects are interpreted offhandedly with only one's one cultural background and world view, this may lead to misunderstandings and misinterpretations in the interaction with others. Therefore, *"poor socialization and social cultural fit"* (Moe and Šmite, 2008), in particular in combination with the unawareness of that cultural diversity, represent a big hurdle for trust building (Moe and Šmite, 2008).

Interestingly, socio-cultural diversity seems to be a sensitive issue many people are aware of as may be concluded from their politically correct answers when interviewed:

"Our analysis of their trust levels towards remote others, of diverse cultural regional backgrounds, revealed that they typically had lower levels of trust. This finding led us to several conclusions. First, the discrepancy between the answers given by participants when asked directly and the reality of their sense of trust towards remote others can be traced to the study participants' sense of what is socially acceptable or what the politically correct answer to such a direct answer should be" (Al-Ani et al., 2013).

The **lack of language skills also hampers socializing and willingness to communicate** and thus is another barrier for trust building (Moe and Šmite, 2008).

Unfortunately, *"a lack of trust may put the overall collaboration in jeopardy"* (Moe and Šmite, 2008). When team members do not trust each other nor their leader, they do not behave quite loyally and do not primarily work towards the common team goal. They rather rival and concentrate on their own interest and goals. Constructive feedback and error culture is therefore not possible, instead the team members try to protect themselves and to be untouchable for teammates or their manager. A manager lacking confidence will try to strengthen control of his team. This in turn is counterproductive and reduces the *"level of trust even more"* (Moe and Šmite, 2008). For example, a growing number of meetings is an indicator for lack of trust on the part of the manager – he does not trust in the reliability and self-responsibility of the team members and they in turn do not build trust in their manager in that way. (Moe and Šmite, 2008)

In general, **developing a feeling of belonging is much more difficult in distributed teams** (Moe and Šmite, 2008). Additionally, parts of the team located in smaller sites may have a sense of inequality (Beecham and Noll, 2015). They may have the impression that their work and desires do not count the same as those from team members located in the headquarter or other/bigger sites that determine for example processes or meeting times.

By tendency, software developers are relatively well-paid employees so that for their daily work extrinsic factors might lose some of their relevance. Instead intrinsic factors like *"a need for challenging, creative work with impact for customers and users"* (Beecham and Noll, 2015) are important for their work satisfaction and motivation. Accordingly, when parts of the team are not involved in developing essential solutions but have stick to minor and less important tasks, the underlying need of creating something that matters is ignored and will lead to a drop-off of motivation. (Beecham and Noll, 2015)

For the individual team members, lack of trust, poor socialization, language barriers and different socio-cultural background may create a discomfortable working situation with demotivating factors (Beecham and Noll, 2015; Moe and Šmite, 2008). They cannot that easily assess who they are working with – what the other one's expertise and morale is. All this negatively affects the personal willingness and thus intensity of communication, collaboration, and knowledge sharing with other team members. A drop-off of communication among the team members or with their manager can lead to delay in responses, decreased productivity, delayed escalation of problems, not asking for help or communicating changes, and alike – all vital factors for team efficiency and success.

1.4.2.4 Tools, Processes and other Challenges in GSD

The need of making communication, coordination, and knowledge explicitly available for all sites **requires the usage of tools for documentation, coordination, meeting-setup**, and alike. This places a further burden on team members of virtual teams. To increase the willingness of team members

to use such systems for group collaboration, the groupware designers have to manage great non-trivial issues, as discussed in Appendix A.3 "Groupware Design: Pitfalls and Fallacies".

Despite people and tool-related challenges, process management and control develop new dimensions of complexity in distributed software projects (Moe and Šmite, 2008). A distributed team *"cannot be managed by walking around"* (Bhatnagar, 2014), but remote, culturally diverse people have to be asynchronously managed (Bhatnagar, 2014). Besides the communication and coordination issues, teams on the different sites may have different skill sets and a different velocity, which makes **effort estimation and planning a more complex endeavor** (Ray and Samuel, 2016).

Even for single-site projects, **requirements management** *"is the main challenge for the software development process"* (Misra et al., 2013). Communication and coordination are vital for successful requirements elicitation, analysis, negotiation, and their ongoing update and management (Damian and Zowghi, 2003). In addition to the already difficult situation regarding communication and coordination, in GSD projects, the requirements and the knowledge about them are distributed over different locations and they are handled and modified decentralized. The classical way of becoming aware of changes by personal contacts and informal communication does not work out in GSD projects. The formal, document-driven tracking of requirement changes is slower, ineffective, and by tendency rather treated shabbily. (Damian et al., 2003; Herbsleb et al., 2001; Mockus and Herbsleb, 2001)

As a result, for managers it is more difficult to obtain *"a snapshot of the current version of features being implemented and related information such as decisions about change request approvals"* (Damian et al., 2003). For those who want to work on a requirement, the communication and coordination in that regard is not trivial: *"contacting originators of requirements and related issues, for clarification or elaboration, and [. . .] determining the responsible stakeholders for a particular decision"* (Damian et al., 2003) is more difficult and cumbersome.

Another important team management area affected by the challenges of GSD is **conflict management**. It is critical for trust building and a good, non-poisoned atmosphere and communication in the team but it is almost non-existent in distributed teams (Moe and Šmite, 2008). Moods and team atmosphere cannot be subtly perceived like in a collocated situation. Conflicts may first be recognized at a stage when they manifest as sarcasm or explicit utterances in the communication channels. Accordingly, they are not recognized in their early stages, and the 'kill the monster while it is small'-approach of tackling issues before they become serious problems is hard to put into practice.

1.4.2.5 Approaches for Tackling the Challenges of GSD

The preceding sections focused on the challenges distributed teams and their managers face. They result from the three GSD-inherent types of distances – spatial, temporal, and socio-cultural. Depending on the team constellation one or the other issue is more or less prevalent. Nevertheless, most issues basically depend on human-related factors like trust, communication, and coordination of work. Several approaches to tackle these fundamental issues have been figured out, aiming at reducing one or several types of distance. These approaches are briefly listed below, starting with the three most important trust-engendering factors.

Depending on the team structure, not every virtual team requires every approach to the same intensity. Nevertheless, to have a high-performance virtual team, the following aspects need to be explicitly and sustainably addressed. See Thomas (2014) for interesting, practical insights and advice about the management of virtual teams based on several years of experience.

- Regularly **invest in face-to-face meetings and socializing activities**: Traveling to remote sites is an irreplaceable action for GSD teams. The team members get to know each other, have private and professional exchange, get insights in the others' workplaces, working infrastructure, as well as in the work and social circumstances at the other sites. These factual insights regarding the team mates' background, expertise, and personality increase cognitive trust and socio-cultural understanding. This in turn is the base for building rapport and common ground, both vital

aspects for an effective communication and collaboration of people. (Al-Ani et al., 2013; Canfora et al., 2003; Clark, 1996; Clark and Brennan, 1991; Kotlarsky and Oshri, 2005; Moe and Šmite, 2008; Seidl, 2015; Stotts et al., 2003)

- **Take your time**: A group of people needs time to grow together as a team. Getting to know each other and each other's level of knowledge is a process that must take its time. Without it, people will not be able to realistically assess and understand each other's work, words, and actions. This is necessary for each new member joining the team. (Al-Ani et al., 2013)
- **Experience working together**: For an efficient collaboration of distributed team mates, it is quite beneficial when they previously have worked together face-to-face. Then they know each other's way of working, character, and competences. They will profit from that face-to-face experience when collaborating via technical means because they can better assess and adjust to each other. (Al-Ani et al., 2013)
- **Deliver what is expected**: If team mates do not behave reliably and trustworthy in the collaboration, cognitive trust drops off. (Moe and Šmite, 2008)
- **Clear communication** of expectations: There should be defined communication rules for expectations, conflicts, and alike so that they are available and everyone knows how to handle them (Moe and Šmite, 2008). Moreover, rules for communication should be established together with the team, for example that emails are at least partially answered within 24 (or 48) hours. Or that when important information are transmitted, the receiver immediately confirms that he received the data so that the sender knows that everything went well – and on this occasion at all costs the receiver thanks for the transmission of the information (Thomas, 2014).
- **Regular, predictable feedback** among sites: Frequent communication and continuously keeping all sites in the loop avoids the 'long absent, soon forgotten' effect among sites. People are continuously aware of the remote team members and what they are doing. This strengthens trust in the remote team members and facilitates team communication and coordination and thus team performance. (Moe and Šmite, 2008)
- **Invest in technical means**: On the one hand, the technical infrastructure should be adequate for the group structure. It has to be appropriate for their types of meetings, their ways of collaboration, and their different cultural working styles (see Section A.3.2 "Usage Context and Group Heterogeneity"). Beyond that, it should be available and easy to set up and use. In the last two decades, technical means for remote collaboration have improved significantly. Today, at least in technical domains like the software industry, collaboration tools are available, and using them *"is routine, everyday practice for many"* (Bjørn et al., 2014). The *"technology readiness"* (Olson and Olson, 2000), referring *"to the difficulties faced in adapting, adopting, and bringing collaboration technologies into use"* is not the key challenge any more, as illustrated by the following observation of Bjørn et al. (2014):

  "[...] we followed a team of three people around for about an hour in their attempt to locate an available video conferencing room where the equipment worked. After several failed attempts spent entering video rooms without access to Internet, without an available remote control, or already booked by others, they had to postpone the meeting. These observations highlight that collaboration technology readiness is more an issue of organizational practices and unstable technology rather than the lack of knowledge about the technologies." (Bjørn et al., 2014)
- **Team structure**: More important than the actual team size or project type is the proportional distribution of team members among sites. At each site, the team should be big enough so that they feel equal to the other sites and that they sense that their work is important for the overall project:

  "Our findings suggest that an imbalance in team dispersion has a significant impact on the development of trust. Many participants emphasized that 2-3 members located in a remote site will lead to their 'isolation' or their being regarded as an 'outpost', as the contribution from that

site will be insignificant in comparison with the productivity of other larger sub-teams collocated in other sites. Participants implied that this isolation and minimal dependency (because of comparatively low productivity) on team members in such small numbers would subsequently lead to a lower level of trust the study participant had towards the remote team member." (Al-Ani et al., 2013)

- **Improve language and cultural fit**: Language is the base for communication. Lack of language skills hampers the communication with team mates. The same applies to lack of mutual cultural understanding, which in the worst case manifests as *"cultural egocentricity"* (Thomas, 2014). This refers to an ignorant attitude expressed in comments like 'These Indian developers yet again, when will they learn to be on time? We have to teach them what German punctuality means!'. These gaps should be explicitly tackled with trainings for all, not only for the non-natives. (Krishna et al., 2004)
- **Short asynchronous communication loops**: In agile development approaches, and in particular in eXtreme programming (XP), user stories are not specified in detail in advance. Only when a story starts, the developers communicate with the relevant stakeholders to clarify questions and detailed requirements. These clarifications should be immediately possible without introducing delays in the development. When stakeholders are distributed, however, personal or synchronous communication is often a more expensive process or not immediately possible so that asynchronous communication channels like e-mail are used. (Layman et al., 2006) It is important that such asynchronous requests are quickly answered because in *"a globally-distributed XP team, prompt responses to asynchronous queries positively impact development commitment and confidence and create a focused development environment"* (Layman et al., 2006).
- **Facilitate socializing**: Tools should not only focus on formal meetings and tasks but also encourage socializing and informal communication. For example, a team chat room can provide a platform for informal and social exchange. (Thomas, 2014)
- **Bridgeheads**: One or more people from a remote site reside at a local site. They are the on-site bridge to the remote site to facilitate contact with the local site. On-site, the bridgeheads socialize with the local team, undergo their culture and work practice, and participate in their daily work. The bridgeheads change over time so that in this way all team members take part in the remote teams. (Carmel and Agarwal, 2001; Layman et al., 2006)
- **Customer authority**: *"Define a person to play the role of the customer up front. This individual must be able to make conclusive decisions on project functionality and scope, must be readily accessible, and must have a vested interest in the project"* (Layman et al., 2006). This is in particular relevant in teams with distributed stakeholders where it *"becomes essential for effective requirements management in a distributed XP project to succeed"* (Layman et al., 2006).
- **Cultural liaisons**: *"Although some movement toward other cultures is possible, it is unrealistic to expect expatriates in any country to be able to think and act like locals"* (Krishna et al., 2004). Therefore it is important to 'install' cultural bridges between the locals and the non-locals (team members or customers), ideally people who are fluent in both languages and for several years familiar with both cultures. (Krishna et al., 2004)
- **Straddlers** (technical and managerial liaisons): As with cultural liaisons, the same applies for distances between development and management departments. Accordingly, a defined person should be determined who is responsible for the daily communication between them. This person needs a good command of all languages involved as well as an understanding of the concepts and terminology of both worlds. (Layman et al., 2006)
- **Explicit trust management**: Establishing and maintaining trust in a team is no matter of coincidence. An important GSD project management responsibility is not only to encourage trust building at the beginning of forming a team but also to ensure trust preservation within a team, i.e. that the team members behave trustworthily throughout the collaboration. (Tjørnehøj et al., 2012)

- **Leadership exemplary function for trust**: It is of vital importance for a team leader to behave in an exemplary and trustworthy form. In his function he plays a major role in establishing the trust and communication culture in a distributed team. His tasks involve to stand in the for team members' rights in the company's policies and to feel responsible to solve problems and conflicts. (Al-Ani et al., 2013)
- **Staff-care**: For GSD environments, proper staff needs to be found, integrated into and trained for the company-specific GSD environment. As discussed, GSD involves a lot of potential demotivating factors and thus caring for staff satisfaction is vital for GSD projects: *"People are the most important component in software development. However, especially in small and medium-sized software companies that employ GSD, an employee-care culture is almost nonexistent and, thus, temporary contracts, low personal development perspectives, low salaries and so on are very frequent [...] In such scenario, talented personnel will not stay long and the personnel continuity has been suggested as a factor that increases quality and effectiveness of software development"* (Misra et al., 2013).
- **Transparent, flexible process with shared understanding**: A GSD team should use a common, defined development process. All people involved should be trained how to live this process in their daily work. The team and relevant stakeholders should have easy access to process and product information, and project status and they should review this information on a daily basis. This transparency is helpful for project planning and early reaction to problems or misunderstandings. (Layman et al., 2006; Moe and Šmite, 2008)

In summary, for GSD *"most opportunities are found on the business level, whereas most challenges are introduced at the level of development practice"* (Holmstrom et al., 2006b). Agile practices in turn are considered to be helpful for overcoming the problems of GSD (Moe and Šmite, 2008; Tjørnehøj et al., 2012; Šmite et al., 2010). In particular PP exactly addresses the biggest issues of distributed teams: trust building, intense communication and knowledge transfer between team members. Consequently, it appears that DPP and other agile practices can help distributed teams to profit from PP and overcome some of the biggest miseries in GSD.

1.4.2.6 GSD and Agile Software Development

Without a doubt, a process depending on explicit documentation of information and process status as well as explicit technical set-up for communication and coordination is not lightweight and encouraging person-to-person communication (Baheti et al., 2002a). Different working hours and distribution of teams and tasks are not in line with the trend of open spaces aiming at a greater team productivity by encouraging communication, easy information sharing on walls, or spontaneous brainstorming at a whiteboard.

A comparison of agile GSD with collocated agile processes is, however, not of interest here. Of course, in such a comparison, GSD is quite likely to lose the competition of who is the most agile. **The focus should rather shift to the question of how agile practices can successfully be used or adapted to help GSD teams** with improving their communication, collaboration, and efficiency (Baheti et al., 2002b; Holmstrom et al., 2006b). Accordingly, examining the sensible feasibility of agile practices in GSD is a relevant task (Kircher et al., 2001).

There are agile elements, in particular XP practices, which are independent of the team structure and may be applied in distributed projects (Holmstrom et al., 2006b), like *small releases*, *continuous integration*, *sustainable pace*, *metaphor*, *simple design*, *test-driven development*, *customer tests*, *design improvement* (refactoring), *collective ownership*, and *coding standards*. Nevertheless, they still rely on self-responsiveness, discipline, and intense communication of the members and stakeholders. Other XP practices like *whole team*, *planning game*, or *pair programming* need to be adapted in a distributed team situation or need to be supported by technical means (Holmstrom et al., 2006b). eDPP tools facilitate pair programming. eDPP can also support developers to practice *whole team* to some degree:

whole team means that every stakeholder (not only developers) who contributes to the success of a project is part of the team. Every team member should have a sense of allegiance to the team and all work closely together.

Only with a deep understanding of the values and goals of the agile practices these can be reasonably adapted to successfully unfold their potential in distributed settings. For example, Kircher et al. (2001) discuss ideas of how different practices and face-to-face events can be substituted by or accomplished via technical means. In addition to such theoretical considerations, a thorough understanding of how effectively each individual practice actually is implemented in the daily practice of GSD teams is important. Then, and with the agile principles and values in mind, it can be assessed how successfully a distributed team adapted an agile practice. It also allows insights about what needs to be changed/improved and about the conditions that make a practice successful in a distributed setting.

This dissertation offers a first approach by examining the usage of the concrete XP practice PP for one distributed pair. PP is the most famous and collaboration-intense XP practice and at the same time very efficient for cross-site communication and knowledge transfer (Bandukda and Nasir, 2010; Holmstrom et al., 2006b; Moe and Šmite, 2008). **Although this dissertation thus focuses on a very specific aspect of distributed software development, it has a rather wide area of application and considerable practical relevance**.

1.4.3 Computer Supported Cooperative Work (CSCW)

Nowadays systems are needed which allow for multiple users and may support groups with their tasks:

"Regardless of the size of the task, computers have traditionally focused on increasing an individual's productivity and have somewhat ignored the fact that many projects are worked on by groups" (Fouss and Chang, 2000). *"Even systems designed for multi user applications, such as office information systems, provide minimal support for user-to-user interaction. This type of support is clearly needed, since a significant portion of a person's activities occur in a group, rather than an individual, context."* (Ellis et al., 1991)

A *"relatively new area of research"* (Fouss and Chang, 2000) focusing on this issue is CSCW. It *"looks at how groups work and seeks to discover how technology (especially computers) can help them work"* (Ellis et al., 1991). **From a CSCW perspective, distributed software development is known to be difficult in general** (Olson and Olson, 2000) and recent research is still concerned with fundamental questions regarding tools and practices for synchronous and asynchronous collaboration on team or individual level (Olson and Olson, 2014).

The above indicates that CSCW is a multifaceted research field, comprising technological as well as social aspects:

CS: the technological perspective: computers supporting the group's work and processes

CW: the social perspective: collaborative work, a *"social phenomenon that characterizes group work"* (Cruz et al., 2012).

To further set out the complex aspects of the social and collaborative interactions involved in the social perspective of CSCW, a more sophisticated description of group and group members by Cruz et al. (2012) is given:

Group *(Explanation of terms)*

> *"Groups can be defined as 'social aggregations of individuals' with awareness of its presence, conducted by its own norms, and supported by task interdependencies towards a common goal in a shared purpose or work context [46]. In this sense, a group is constituted by particular characteristics, such as: size (3 to 7, >7), composition, location, proximity, structure (leadership and hierarchy), formation, group awareness (low or high, and cohesiveness), behavior (cooperative or competitive), autonomy, subject, and trust. The group members have a personal background (work experience, training, and educational), skills, motivation, attitude towards technology, previous experience, satisfaction, knowledge, and personality"* (Cruz et al., 2012).

The definition of CSCW shows that the subject matter of CSCW are **socio-technical systems that have to consider far-reaching aspects of different technical and non-technical disciplines**. CSCW provides a broad field of activity for, among others, computer scientists, psychologists, sociologists, anthropologists, and management experts. Ideally, these disciplines are involved in the design of group-supporting technology, which is referred to as groupware. (Fouss and Chang, 2000; Rein and Ellis, 1989)

Per definition, a group has at least 3 members. This definition is helpful to distinguish from a pair. In the context of CSCW, however, the differentiation is not that strict – there is no derivative like 'pairware'. Nevertheless, some challenges of groupware are plausibly bigger when handling a group of several people than when handling a pair. This aspect is touched upon in Section 2.1 "Requirements for eDPP Tools".

Groupware *(Explanation of terms)*

> The term groupware denotes computer applications or hardware that are explicitly designed to support the cooperative work of groups or teams – regardless of whether the members are collocated or remote and independent of the type of tasks to accomplish.

Due to the quite general characterization of groupware, a lot of application types fall in this category (Fouss and Chang, 2000). A more differentiated definition and taxonomy of groupware systems would be helpful to better *"contrast and compare various groupware tools"* (Fouss and Chang, 2000). Such an overview of groupware definitions is given in Appendix A.1 "Definitions of Groupware". Appendix A.2 "Groupware Taxonomies" presents the most common classification attempts for groupware.

Nevertheless, the comparison of different types of groupware is not part of this dissertation. In this work only one type of groupware (for real-time collaboration of two developers) is considered and it will be compared with similar tools with respect to the scope of functions. This is why the next two sections are relevant – they create awareness about general requirements for groupware development (page 62 ff.) and adapt them for the special case of eDPP collaboration (page 66 ff.).

The following observation regarding the development of groupware also might be of interest:

"Many systems, applications, and features that support cooperative work share two characteristics: A significant investment has been made in their development, and their successes have consistently fallen far short of expectations. [...] In most instances of failure, a substantial and timely return on investment was certainly anticipated; the decision-makers were not in business to throw away money" (Grudin, 1988).

In light of the complexity of groupware, this insight is no surprise. It reveals two important aspects: even a significant effort during development is no guarantee for user acceptance, since it still might be the wrong effort. This is true for single-user applications as well as for groupware. The difference or advantage is that in single-user application development managers and developers are trained and have

practice. They can rely on their knowledge, experience and intuition to make sound decisions. (Grudin, 1988)

For groupware development, this currently does not apply because *"a typical CSCW application will be used by a range of user types – people with different backgrounds and job descriptions, all of whom may have to participate in one way or another for the application to succeed. The decision-maker's intuition will fail when an appreciation of the intricate dynamics of such a situation is missing"* (Grudin, 1988). Appendix A.3 "Groupware Design: Pitfalls and Fallacies" discusses these aspects, the general challenges for groupware design, in more detail.

1.4.3.1 Groupware Design: General Principles

Cooperative work encompasses *"information processing and communication activities"* (Ellis et al., 1991) among the individuals. Accordingly, apart from the awareness and the functional requirements also individual skills und peculiarities, interpersonal interactions, and specific workflows have to be facilitated to make a groupware system truly effective (Lukosch and Schümmer, 2007). The difficulty here is that the different factors – the group processes, the work context, and the interactions with the technology – influence and change each other (Fouss and Chang, 2000; Rein and Ellis, 1989). **This unpredictable dynamic leads to highly complex requirements for groupware design**.

This section outlines relevant aspects of groupware design. The respective aspects remain fairly general, as discussing concrete mechanisms or technical details would be extremely far reaching and falls outside the scope of this dissertation. Instead, merely the crucial points are raised and important approaches described.

System requirements due to (distributed) group setting A groupware system usually manages or integrates the work of multiple users who are *"distributed in time and/or space"* (Ellis et al., 1991). This implies several technical system requirements concerning user communication, data protection, and consistency.

- **Communication**: The group members must be able to exchange information. The adequacy of the provided communication channels depends on the time/space setting and the required intensity of information exchange. Often the debate concerning communication support of groupware is on how to make the technical channels for information exchange as effective as face-to-face communication. Ellis et al. (1991) offer an alternative point of view by suggesting to to understand the new distributed situation and its challenges and provide (new) means or channels for communication or collaboration that best support it (see also the last paragraph of Section 5.2 "Conjecture: ePP – the Better PP" on page 205).

- **Access Control/ Privacy**: When multiple group members access shared data, it is often necessary to define who is allowed to do what on which part of the shared data. Access permissions may change over time and thus the permissions mechanism should be easily understandable and customizable. (Ellis et al., 1991)

  Beyond that, a *"system can simplify the process of obtaining appropriate access rights by supporting negotiation between parties"* (Ellis et al., 1991).

- **Floor control**: *"Synchronous interaction can lead to parallel and conflicting actions that confuse the interacting users and makes interaction difficult"* (Schümmer and Lukosch, 2007). Technical mechanisms for avoiding conflicts are locking or floor control. Locking ensures that only one user at a time can edit a certain part of the shared data. It requires an adequate locking granularity, considering *"the overhead of requesting and obtaining the lock"* as well as the *"timing of lock requests"* (Ellis et al., 1991). *"These problems point to a basic philosophical difference between database and groupware systems. The former strive to give each user the illusion of being the*

system's only user, while groupware systems strive to make each user's actions visible to others" (Ellis et al., 1991).

Floor control mechanisms like turn-taking protocols avoid conflicts by allowing only one user at a time to have the stage for editing the shared data (Schümmer and Lukosch, 2007). This puts all other participants in the passive role of viewers, requiring explicit effort to become active. In particular in synchronous collaboration on textual or visual objects, this is unnatural and inhibits *"the free and natural flow of information"* (Ellis et al., 1991). **Social floor control**, when editing rights are not technically regulated but socially agreed upon by the participants, is more flexible and appropriate for interleaved interactions. Nevertheless, this may *"be unfair, distracting, or inefficient"* (Ellis et al., 1991). Social floor control relieves the system of regulating writing access on the data, but – in particular for real-time collaboration – it requires data consistency and conflict detection.

- **Ensure data consistency**: When multiple group members work on shared data, their operations have to be merged and all users need a coherent view on the data. Conflicting operations have to be detected and resolved. In particular for real-time collaboration, the merging of the different operations in the correct order requires a short response time of the system (Ellis et al., 1991). Mechanisms for merging operations and conflict detection like for example dependency detection, reversible execution, or operation transformations, are by no means trivial. They are large and complex technical issues in their own right.
- **Group processes**: The same aspects as for floor control apply for the specification of the order of work steps – they can be technically specified or socially coordinated. A restrictive system, prescribing who is allowed to do which work step when and on which artifact, helps to prevent chaos, but is inflexible and inhibits effective collaboration: *"a group's idiosyncratic working style may not be supported, and the system can constrain a group that needs to use different processes for different activities"* (Ellis et al., 1991). With more unconstrained systems *"the group must develop its own protocols, and consequently the groupware itself is more adaptive"* (Ellis et al., 1991).

Responsiveness to group dynamics A group *"is a unique combination of its constitutes, environment, and task"* (Mandviwalla and Olfman, 1994). A group also changes in its constitution over time and permanently is subject to dynamic group processes and interactions (Mandviwalla and Olfman, 1994). Consequently, besides the functional requirements that arise from the groups' tasks and goals, a groupware system must cope with the dynamics of this 'conglomerate'.

However, such dynamics depending on the interaction of multiple individuals and their environment can hardly be captured, nor can changes be reliably predicted or anticipated. Accordingly, group-specific requirements can hardly be elicited. One way out of the dilemma is to develop a flexible system that is not overly restrictive and thus is robust to changes and adaptable to group dynamics. (Mandviwalla and Olfman, 1994)

Crucial criteria for **creating flexible groupware** systems are

- **Group development and rearrangement**: A group is usually developing. Roles, responsibilities, and memberships change. The groupware should allow easy management of members as well as reassignment and redefinition of roles (Mandviwalla and Olfman, 1994).
- **Different modes of interaction**: A group can interact synchronously or asynchronously. The performed operations may be highly structured or rather unstructured (Ellis et al., 1991). A groupware system often is used for tasks which — with respect to the collaboration modes — involve heterogeneous sub-tasks: *"A meeting proceeds in a largely unstructured way, but it can contain islands of structured synchronous operations – such as voting or brainstorming"* (Ellis et al., 1991). The crux is to recognize the different characters of the sub-tasks and accordingly

design the groupware: *"This calls for integrating support for structured/unstructured activity on the one hand and for synchronous/asynchronous activity on the other"* (Ellis et al., 1991).

- **Different means of interaction**: Depending on the phase of collaboration and the time/space situation, a group needs different channels for communication and interaction. In a synchronous distributed setting, a meeting may be coordinated using an audio channel or text chat, whereas in a distributed and/or asynchronous setting, voice mail, e-mail or an electronic group calendar are required to schedule the meeting. During a distributed meeting in turn, video conferencing may be the channel for interaction. (Mandviwalla and Olfman, 1994) All of these different communication needs have to be recognized and taken into account.

Group interfaces *"Group interfaces differ from single-user interfaces in that they depict group activity and are controlled by multiple users rather than a single user"* (Ellis et al., 1991). Thus, compared to a single-user system, multi-user interfaces must deal with more combinatory possibilities of interactions. Additionally, a groupware interface must not only **design the interaction between the system and the user, but also the interaction among users**. Central aspects in that regard are:

- **Social interfaces**: Groupware must display information about other group members and provide means to interact with them and to start a collaboration (Mandviwalla and Olfman, 1994).
- **Workspace Awareness**: As already mentioned, a specific and quite relevant aspect of information about others in the context of group collaboration is workspace awareness. When jointly working towards a common goal, it is important to know what others are doing or have done. Displaying awareness information must overcome the challenge of not spamming and distracting the user with inappropriate updates about others but providing enough information so that the user can effectively integrate his own work with others' activities (Ellis et al., 1991). The granularity and update interval of workspace awareness depends on the coupling of tasks and the intensity of collaboration. (Ellis et al., 1991) Workspace awareness is addressed in more detail in Section 1.4.4 "Workspace Awareness in Real-Time Groupware".
- **Notification**: In single-user systems, the user is notified when performing wrong or invalid operations. In multi-user systems, the user must additionally be notified when others make changes that affect his own work. Thus, notification is a specific aspect of workspace awareness. In particular in *"synchronous interactions, real-time notification is critical"* (Ellis et al., 1991), whereas aspects *"such as performance, group size, and task are involved in choosing an appropriate level and style of notification"* (Ellis et al., 1991). For tightly-coupled work like co-authoring, a fine-grained level of notification is recommended, for example on character-level. Whereas for loosely coupled work, when participants work on separate tasks, usually coarser notification mechanisms are sufficient (Ellis et al., 1991). Dependency-induced notifications offer an elegant way of designing systems with 'as little as possible, as much as necessary' notifications.
- **Shared visual context**: Efficient collaboration requires a shared context and a shared mental model of the task and edited data. A similar view on the shared data facilitates a shared context and effective communication about it, e.g. referencing a line number is possible (Ellis et al., 1991). The approach what you see is what I see (WYSIWIS) provides quite a good visual shared context, as everyone sees about the same (Ellis et al., 1991). It is, however, inflexible concerning window placement and size or user-customized display settings. An improvement concerning the viewing freedom is **relaxed WYSIWIS**. It allows users to control their window size and appearance and to *"have different viewports into the workspace"* (Greenberg, 1996). According to Stefik et al. (1987), the relaxation can be along four dimensions that have proven to be useful:
  - Display space: showing only a fraction of the shared objects (windows, cursors, etc.) relaxes the display space issue.
  - Time of display: compared to strict WYSIWIS, where views are instantly updated on all sites, in this relaxation the update interval of views can be scaled down so that the respective

display updates are delayed.

  - Subset of participants: only a fraction of the group shares the same view – this not only reduces the number of people forced to have a specific view, but also the amount of information that has to be displayed in the shared view (e.g. tele-pointers).

  - Alternate views: compared to strict WYSIWIS, where participants have congruent views, participants may decide to customize their view on the commonly viewed data and are allowed to move to another part of that shared data.

  Relaxed WYSIWIS and in particular alternate views give *"people control over their own viewport into a workspace, and thus allow them to work in a more natural style, shifting their focus back and forth between individual and group work"* (Greenberg et al., 1996). On the other hand, it has drawbacks with respect to awareness *"since, when views differ, people can lose track of where others are and what they are doing in the workspace"* (Greenberg et al., 1996). As already mentioned, this dissertation focuses especially on this issue in the context of eDPP .

- **Screen space management**: Even in times of big, affordable displays, screen space remains a valuable resource. Applications or interfaces become more complex. At the same time, human attention and the primary field of view are limited. As a consequence, a group interface allowing users to create windows that also appear on other users' displays must provide means *"for managing window proliferation"* (Ellis et al., 1991). *"One approach is to aggregate windows into functional sets, or rooms, each of which corresponds to a particular task"* (Ellis et al., 1991). Users can dive into a task- or topic-related window configuration without being bothered or overwhelmed by windows that are beyond their task. Generally, however, actions resulting in cluttering up others' screens should be treated with great care. (Ellis et al., 1991)

- **Group interface toolkits**: Toolkits that support the development of single-user interfaces are sufficiently mature nowadays. Group interface development cannot refer to as many years of experience and expertise as single user-application development. Accordingly, group interface toolkits do no not have the same stage of maturity as single-user interface toolkits. (Ellis et al., 1991).

System integration and establishment Concerning the launch of groupware systems and its integration in a group setting, the following criteria should be considered.

- **Adoption**: Basically, innovation management is a highly sensitive issue. And the adoption of groupware systems is a particular challenge; a group is a dynamic, social structure where the users are quite likely to have different backgrounds, skills, needs, and goals that are not all equally considered by a system. At the same time, group collaboration requires an extended workflow. As a matter of fact, the personal benefit of these additional work steps is not always evident to all users. Groupware establishment is a socio-technical process of innovation: *"Groupware developers need to be conscious of the potential effects of technology on people, their work and interactions"* (Ellis et al., 1991). As explained in Appendix A.3.2 "Usage Context and Group Heterogeneity", the issues of personal and collective benefit have to be carefully considered.

- **Ease of Interaction**: *"Group processes offer increased synergy and parallelism, but the required coordination overhead can burden the group and dampen its effectiveness"* (Ellis et al., 1991). Accordingly, a groupware designer should always keep a close watch to provide an easy route for starting or participating in a collaboration and reduce the overhead and obstacles to start (Ellis et al., 1991).

- **Interoperability**: Although a groupware system should cover as many group tasks as possible, *"it is unlikely that a single system will ever be able to satisfy all the requirements, whereas multiple self-contained systems will bring too many usage constraints"* (Mandviwalla and Olfman, 1994). Therefore, for groupware systems, interoperability with existing organizational applications and

> data is extremely important. This avoids annoying and time-consuming manual data in- and output between different applications or other workarounds. (Mandviwalla and Olfman, 1994)

All of the above emphasizes the importance of a development procedure that very carefully examines not only the functional scope but also user context, social facets and dynamics, usefulness, and ease of usage. To meet these objectives, a user-centered development approach is vital (Schümmer and Lukosch, 2007).

1.4.3.2 Groupware Design: Real-Time Groupware

Real-time groupware, like for example for eDPP, is the top tier regarding system requirements such as response time, awareness, and data consistency. The participants collaborate synchronously in a distributed setting. They work on the same task, on the same shared artifact. They intensively communicate, their work is tightly coupled, and they need to be aware of each other's actions on a very fine-grained (keystroke) level.

For the pure purpose of supporting the eDPP activity, group development, dynamics of membership, and overall processes are not a direct issue.

For the adoption of DPP groupware, however, integration of the system in the user's existing working environment and ease of collaboration are important. Developers' working environments are usually highly customized with regards to view/window arrangement, installed plugins, key bindings, and path configurations. Developers also stated to feel more comfortable and efficient when working in their 'home' environment (Plonka et al., 2012a). Accordingly, a short and lightweight collaboration startup from within an already used programming environment seems desirable.

Concrete requirements for eDPP tools are set out in Section 2.1 "Requirements for eDPP Tools", followed by a look at actual eDPP tools and their comparison along these requirements. At first, though, the remaining sections of this chapter examine the other directly related research areas of this dissertation – workspace awareness and research on distributed pair programming.

1.4.4 Workspace Awareness in Real-Time Groupware

Awareness, and in particular workspace awareness, *"holds promise for significantly improving the usability of real-time distributed groupware"* (Gutwin and Greenberg, 2002). The different types of awareness and the importance of workspace awareness for collaboration were presented in Section 1.1.4 "Awareness". Due to its relevance, this section addresses related work on workspace awareness for real-time groupware.

Unfortunately, *"no clear overall picture of awareness has yet emerged from the CSCW community"* (Gutwin and Greenberg, 2002). There are no generalities of how to gather and present workspace awareness information in real-time collaboration tools. Awareness itself is a complex, intangible, and mostly unwitting phenomenon and accordingly it is hard for users to explicitly talk about it. Furthermore, socio-technical evolutions and real-time collaboration are rather contemporary phenomena. There is no matured wealth of experience on how to address them in specific application domains. However, the emergence of multiple distributed users interacting simultaneously opens a new dimension of interaction complexity and introduces the need of workspace awareness.

To display workspace awareness information on one site of the collaboration, it must first be captured on the other side. This involves two fundamental problems groupware designers face and for which there is no trivial solution (Gutwin and Greenberg, 1995):

- *"They must know what awareness information a groupware system should capture about another's interaction with the workspace"* (Gutwin and Greenberg, 1995).
- *"They must consider how this information should be presented to other participants"* (Gutwin and Greenberg, 1995).

Beyond that, *"it is infeasible to replicate the detail and size of real-world workspaces, however, designers must carefully determine what information is most important, and how it can be put to best advantage in the system"* (Gutwin and Greenberg, 2002). This involves the **challenges of capturing appropriate awareness information and present it at the right time, in an adequate granularity, and in an easily perceivable manner**. This in turn can be further refined into the following aspects that have to be considered when designing features to provide workspace awareness in groupware:

- Gathering: knowing which awareness information is relevant to collect.
- Gathering: knowing how to gather this relevant workspace awareness information.
- Gathering: information gathering must occur automatically and be unobtrusive in order not to distract the collaboration process.
- Presentation: knowing how to present workspace awareness information.
- Presentation: knowing when to present which workspace awareness information.
- Presentation: the information presentation must be unobtrusive and not require explicit actions. It means, in effect, the workspace awareness should be easily perceivable (no information overload that leads to ignorance) and interpretable.

Unfortunately, these challenges cannot be generally solved. To help designers cope with these issues, Gutwin and Greenberg (2002) developed the so-called **workspace awareness framework**. It is groundwork and state-of-the-art research with respect to a differentiated view on workspace awareness for group collaboration. They developed and conceptualized a spectrum of awareness information they considered to be relevant for face-to-face as well as for distributed real-time collaboration. The framework shapes a vocabulary on workspace awareness and is helpful for comparing awareness support of real-time collaboration tools as well as for explaining observations. (Gutwin and Greenberg, 1999, 2002)

Furthermore, the workspace awareness framework facilitates structured reflection about basic workspace awareness requirements in real-time collaboration. With that, it addresses the workspace awareness aspects a groupware designer has to consider. It was **developed iteratively over several years and evolved from psychological and awareness research, while observing face-to-face collaboration of small groups as well as from testing resulting assumptions with awareness widgets in real-time groupware**. (Gutwin and Greenberg, 1999)

The workspace awareness framework has three components that provide **insights about the actual usage of workspace awareness in face-to-face collaboration**:

- **Part one: what makes up workspace awareness**: *"what kinds of information people keep track of in shared workspaces"* (Gutwin and Greenberg, 1999).
- **Part two: sources of workspace awareness**: *"how people gather workspace awareness information"* (Gutwin and Greenberg, 1999).
- **Part three: usage of workspace awareness**: *"how people use workspace awareness information in collaboration"* (Gutwin and Greenberg, 1999).

These aspects are evidently relevant when considering workspace awareness information in face-to-face collaboration (Gutwin and Greenberg, 1999), so that they are a plausible basis for considering workspace awareness in distributed real-time collaboration as well.

The following sections introduce the three components of the framework in detail. Prior to this, a sketch of the situational frame, for which the workspace awareness framework was developed, might facilitate understanding its transferability to the collaborative situation of eDPP. As part of the discussion of requirements for eDPP tools, Section 2.1.1 "Adapting the Workspace Awareness Framework to eDPP" takes a closer look at the applicability of the workspace awareness framework for eDPP.

The situation for which the workspace awareness framework by Gutwin and Greenberg (2002) is applicable is characterized as follows:

- **Environment: shared workspaces** – A shared workspace is a *"bounded space where people can see and manipulate artifacts related to their activities"* (Gutwin and Greenberg, 2002). In the physical world, this is any place or area, like a table, countertop, work bench, atelier or gymnasium floor, where people jointly perform activities on physical artifacts. Gutwin and Greenberg concentrate on *"flat, medium-sized surfaces upon which objects can be placed and manipulated, and around which a small group of people can collaborate"* (Gutwin and Greenberg, 2002), for example when developing a project timeline with sticky notes on a wall. In these types of workspaces, the participants create an artifact containing some type of visible and touchable objects. Thus, the focus of their joint activity are these objects and they are physically manipulated to accomplish the group task. (Gutwin and Greenberg, 2002)

  Clark (1996) describes such a workspace and its objects as *"an external representation of the current state"* of the joint activity. He compares it to a chess board, where the board and its figures are a visible representation of the current state of the game. Such a visual representation of the joint activity is part of the group's common ground and an effective help to remember what has just been done and determine what to do next (Clark, 1996).

- **Task type: generation and execution** – The group jointly develops something and this involves creating, manipulating, or arranging physical artifacts.

- **Groups: small groups and mixed-focus collaboration** – *"Small groups of between two and five people primarily carry out tasks in these medium-sized workspaces. These groups often engage in mixed-focus collaboration, where people shift frequently between individual and shared activities during a work session"* (Gutwin and Greenberg, 2002).

- **Systems: real-time distributed groupware** – Based on workspace awareness in group collaboration on physical objects, Gutwin and Greenberg (2002) examined real-time collaboration in groupware, like *"shared editors, group drawing programs, multiplayer games, and distributed control systems"*.

1.4.4.1 Part One: What makes up Workspace Awareness

To structure workspace awareness information, Gutwin and Greenberg (2002) divide relevant workspace information into five basic categories:

- **Who**: information about who we are working with.
- **What**: information about what the other ones are doing in the shared workspace.
- **Where**: information about the physical position of the others in the shared workspace.
- **When**: information about when events/activities happened in the shared workspace.
- **How**: information about how events occurred (how operations were performed).

Within these categories, there are **elements of information**, i.e. the concrete aspects of a work situation that people keep track of. Within each category, these elements cover three perspectives: What is relevant to know about the workspace, the other person(s) and the artifact? (Tam and Greenberg, 2006)

"Although there will also be additional kinds of information specific to the task or the work setting, these basic elements provide a high-level organization of workspace awareness" (Gutwin and Greenberg, 2002).

These **elements are specified by questions** that lead to answers providing the concrete pieces of information which are relevant for a person in a specific collaborative situation (Gutwin and Greenberg, 2002; Tam and Greenberg, 2006).

The following categories and elements with concrete questions are basically extracted from Gutwin and Greenberg (2002)):

- **Who** we are working with: A category representing social aspects of collaboration, providing knowledge about the others' presence in the workspace, their background and mapping observed activities to identities.
  - **Presence**: Is anyone in the shared workspace?
  - **Presence history**: Who was here and when?
  - **Identity**: Who is participating?
  - **Authorship**: Who is making the changes in the artifact?
- **What** others are doing or what they have been doing. This category comprises knowledge about which activities others perform on which objects in the shared workspace. It also helps to infer their intentions by knowing what overall task their activities are associated with.
  - **Action**: What are they doing?
  - **Action history**: What has a person been doing?
  - **Intention**: What goal is that action part of?
  - **Artifact**: What object are they working on?
- **Where** others are working. This component refers to the whereabouts of the others – involving which areas of the workspace they look at and which they can see, where they have been working, and where in the workspace they can make modifications.
  - **Location**: Where are they currently working?
  - **Location history**: Where has a person been working?
  - **View**: What can they (potentially) see? / What is their field of vision? (Tam and Greenberg, 2006)
  - **Gaze**: Where do they currently look?
  - **Reach**: What can they manipulate? *"Awareness of reach involves understanding the area of the workspace where a person can change things, since sometimes a person's reach can exceed their view"* (Gutwin and Greenberg, 2002).
- **When** various events happened. This category exclusively relates to the past and comprises the knowledge about when events happened in the past.
  - **Event history**: When did that event happen?
- **How** these events occurred. This category also relates to the past and is concerned with details about events and changes.
  - **Action history**: How did that operation happen? *"Action history describes the unfolding of events that changed the workspace"* (Tam and Greenberg, 2006).
  - **Artifact history**: How did this artifact come to be in this state? *"Artifact history includes details about the process of how an object was changed over time"* (Tam and Greenberg, 2006).

Elements about future events are not part of the workspace awareness framework, because, in contrast to present and past happenings, they cannot be derived from the actual happening but involve *"inference, extrapolation, and prediction"* (Gutwin and Greenberg, 2002).

Although workspace awareness is a combination of all the elements outlined above, there is no need for the designer to support all of these equally in the interface: **The relevance of specific**

awareness information, its appropriate granularity, update interval, and visual representation is context-specific and needs to be adapted to concrete application domains. The intensity of the collaboration is relevant to determine an adequate level of detail for the provided awareness information. The dynamics of the information, i.e. how often it changes, are relevant for the update interval of awareness information in the interface. (Gutwin and Greenberg, 2002)

For example, in eDPP, the typing activities in the editor are very dynamic in contrast to intense coding phases where artifact switches are less dynamic.

1.4.4.2 Part Two: Sources of Workspace Awareness Information

Gathering workspace awareness in face-to-face collaboration occurs easily and effortlessly. This should also be possible in distributed collaboration. Users can better perceive and understand awareness information if they *"can gather information in familiar ways, even though the actual interface devices in a groupware system may not be familiar"* (Gutwin and Greenberg, 2002). To be able to present awareness information in a familiar[30] way, in the first instance, it is important to understand *"how people find the answers to the who, what, where, when, and how questions"* (Gutwin and Greenberg, 2002). In the workspace awareness framework, three main sources have been identified that are used by group members in face-to-face collaboration to obtain this information: other people's bodies in the workspace, workspace artifacts, and verbal communication. And according to Gutwin and Greenberg (2002) there are several mechanisms for gathering information from team mates and the shared working environment.

People's bodies in the workspace The first important source of awareness information is people's bodies in the workspace. The body of a person provides relevant information by its

- position and posture. I.e. where someone stands in front of a whiteboard and how he stands there (arms crossed or not, etc).
- motions when executing actions, either of the whole body or parts of it like hands, arms, head, or eyes. I.e. when writing something at a whiteboard or sticking a note on it.

These information are not deliberately 'produced' to convey information to others *"in the way that explicit gestures are"* (Gutwin and Greenberg, 2002). They are simply the **consequence of someone's acting in the workspace**. Accordingly, the awareness is created through **consequential communication** by watching and others' bodily actions in the workspace. Bodily communication as well as hearing others in the workspace provides a lot of knowledge on what is going on in the workspace. Finally, it is a fundamental aspect to make a collaboration smooth and efficient. (Gutwin and Greenberg, 2002)

Workspace artifacts The second important source of awareness are the workspace artifacts themselves.

They provide information by their

- spatial relationships: their position relative to the workspace and their proximity to one another. I.e., on which walls or whiteboards in a room sticky notes are placed and which ones are grouped.
- physical appearance: their graphic or textual presentation as well as their nature and condition. I.e., the text or sketches shown on the sticky notes.
- applied changes: observing changes in artifacts allows to infer actions of the actor. For example, when seeing text disappear from a whiteboard and someone is acting in front of it, even from a bigger distance, it is clear that this person is erasing text and not writing something.

[30] For an interesting discussion 'intuitive equals familiar' in software usability by Jef Raskin see (Raskin, 1994)

The awareness is created **through the artifacts** by observing the effects of others' changes: *"In a cooperative setting not only is it important to see one's own updates, but also to see the effects of other people's actions. This is feedthrough"* (Dix, 1997).

"When both the artifact and the actor can be seen, feedthrough is coupled with consequential communication; at other times, there may be a spatial or temporal separation between the artifact and the actor, leaving feedthrough as the only vehicle for information" (Gutwin and Greenberg, 2002).

"In real life, cooperating over physical objects, this communication through the artefact is often more important than direct communication. For example, imagine you are moving a large piano. You may say things to each other – 'move your end up a bit', 'careful of the step' – but in fact the most important thing is the feel of the other person's movements through the movements of the piano. This sort of communication is effective partly because it is tied so intimately to the work itself, and partly because it is implicit, unconsciously noticed and acted upon." (Dix, 1997)

Conversations and gestures The third source for workspace awareness is **explicit verbal and non-verbal communication**. This includes utterances and gestures that are explicitly – and mostly intentionally – yielded by the sender to convey information to the conversation partner. Accordingly, awareness is created through **intentional communication**. (Gutwin and Greenberg, 2002)

In the context of the workspace awareness framework, intentional communication refers to verbal and non-verbal information explicitly 'produced' by the sender, in contrast to unintentional communication that drops off as a by-product of bodily actions.

A conversation can include **direct statements** communicating workspace awareness aspects such as 'I am currently working on artifact X', or it can more indirectly provide information related to the shared work. For example, a question concerning the task or something in the workspace reveals information like which artifact the person is concerned with or what (mental) activity he is engaged in. Accordingly, being part of a conversation or overhearing a conversation of others is a rich source for workspace awareness. Additionally, overhearing others' conversations can also provide side benefits like learning from someone else's expertise, monitoring a novice, etc. (Gutwin and Greenberg, 2002)

Verbal shadowing, i.e. utterances people make alongside their work, can also be picked up in a face-to-face setting. Verbal shadowing can be of explicit nature like statements about the current activity such as 'I am going to do this'. Generally, they are not directed to anybody in particular. They can also be more implicit, for example when mumbling something to oneself or when making fragmentary comments alongside one's actions. (Gutwin and Greenberg, 2002)

Intentional non-verbal communication are gestures and other elements of visual communication that are deliberately used and that substitute, support or complement phonetic language. To convey information, visual communication uses either illustration or emblem. Illustration denotes that *"speech is illustrated, acted out, or emphasized. For example, people often illustrate distances by showing a gap between fingers or hands"* (Gutwin and Greenberg, 2002). Emblem indicates that a word is substituted by an action, for example when shaking one's head instead of saying 'no'. (Gutwin and Greenberg, 2002)

In the workspace awareness framework, intentional communication comprises several types of verbal or non-verbal communication. It involves directed as well as undirected communication, addressing one or several persons, or no one in particular such as when talking to oneself. The level of consciousness of the utterances is not further distinguished, explicit questions or statements as well as more incidental commentary to oneself count as intentional communication.

1.4.4.3 Part Three: How Workspace Awareness is Used

Workspace awareness information are crucial for efficient collaboration. They are used to simplify verbal communication, manage the transition between individual and group work, anticipate other's actions, coordinate work, and provide assistance to peers in a collaboration (Gutwin and Greenberg, 2002).

These aspects are fundamental to facilitate smooth collaboration and will be further elaborated in the following.

Simplification of verbal communication Verbal communication can be enriched by using visual elements of the workspace and the artifacts as *"conversational props"* (Gutwin and Greenberg, 2002). This can make the verbal communication more effortless and efficient by reducing the *"length and complexity of utterances"* (Gutwin and Greenberg, 2002). In face-to-face collaboration, there are four types of **communicative actions** that ease communication:

- **Deictic reference**: indexical expressions like 'me', 'you', 'here', 'there', 'this, etc. obtain their meaning by their reference object. In particular, local deictic ('here' and 'there') and object deictic references ('this', 'that', 'it') are prevalent in collaborative situations.
- **Demonstration**: Gestures to demonstrate dynamic aspects like *"actions or the behavior of artifacts"* (Gutwin and Greenberg, 2002), for example *"tracing a path in the workspace with a finger"* or turning over one's hand *"to demonstrate how a card would flip back and forth"* (Gutwin and Greenberg, 2002).
- **Manifesting Actions**: Actions in the workspace that replace verbal communication (Clark, 1996). For example, taking a sheet of paper and scissors indicates that someone is going to cut the paper.
- **Visual Evidence**: During a conversation, we need active feedback from our conversation partner that our message reached him. This feedback can be verbal (e.g. 'um' or 'uh-huh') or non-verbal (e.g. nodding one's head). Other visual actions in the workspace can also provide evidence that we understood our counterpart. For example, to move an artifact to another position in response to a suggestion signals understanding and consent. (Gutwin and Greenberg, 2002)

Common to all these types of visual communication is that they must be visually perceivable and require a common context for correct interpretation. Thus, *"workspace awareness is part of conversational common ground in a shared workspace"* (Gutwin and Greenberg, 2002).

Common Ground *(Explanation of terms)*

"Two people's common ground is, in effect, the sum of their mutual, common, or joint knowledge" (Clark, 1996). It is the shared knowledge base between two interacting people. It also involves meta knowledge about what is part of the current common ground and what is not: the awareness about one's own as well as the other one's knowledge. This awareness determines how we communicate with a counterpart. (Clark, 1996; Menold, 2007)

A good common ground allows efficient communication (Clark, 1996): *"Sharing knowledge and knowing how the remote collaboration partners interpret and create meaning is critical to making the collaboration function well"* (Bjørn et al., 2014). A less good common ground both parties are aware of requires more extensive communication. Wrong assumptions or ignorance regarding the other one's knowledge lead to communication problems. Common ground is dynamic, which means it is updated during a conversation.

For example, usually a doctor and a patient do not have a good common ground – the patient does not know much about medical science and the doctor does not know much about the patient's way of life. When the doctor talks to the patient as if he had a medical background, it is very likely the patient does not know the latin terms the doctor uses and thus does not understand the message. To make his explanations comprehensible for the patient, the doctor has to use plain language that is part of their common ground. This makes the communication with the patient more 'inefficient' compared to talking to someone who shares his medical vocabulary. If the doctor explains the latin term he uses, he can use them from then on in the conversation with that patient – as he extended their common ground.

For remote collaboration common ground remains of major importance. Bjørn et al. (2014) found that for distributed teams common ground involves two aspects: shared knowledge about the application domain as well as about the processes and work practices. In particular the latter is specific for distributed teams since often the processes differ at the sites (Bjørn et al., 2014).

Management of coupling *"Shared work often involves fluid transitions between relatively focussed collaboration, division of labour, general awareness and serendipitous communication"* (Gaver, 1991). Dourish and Bellotti (1992) observed that during collaboration the participants *"continually moved between concurrent, but more or less independent, work, through discussions and coordination, to very tightly focused group consideration of single items"*. In that regard workspace awareness plays an integral role because these transitions are *"opportunistic and unpredictable, relying on awareness of the state of the rest of the group"* (Dourish and Bellotti, 1992).

Without *"workspace awareness information, people will miss opportunities to collaborate, and will often interrupt the other person inappropriately"* (Gutwin and Greenberg, 2002). For example, when seeing a collaboration partner engrossed in thought, we would less likely interrupt him as when seeing him having a glass of water and looking around.

Coordination Coordination is an essential part of group collaboration. In particular real-time collaboration requires *"concerted action"* (Gutwin and Greenberg, 2002) of people. Activities must *"happen in the right order, at the right time, and [. . .] meet the constraints of the task"* (Gutwin and Greenberg, 2002). This applies to activities of different granularity, *"from small hand movements to large-scale divisions of labour"* (Gutwin and Greenberg, 2002).

Knowing the current state of work and what others are doing allows to align individual activities with the other's activities and thus to sensibly integrate them into the overall 'bustle'. This is particularly relevant in synchronous collaboration, when *"people are working with the same object"* (Gutwin and Greenberg, 2002). For example, when a group of people brainstorms in front of a whiteboard, workspace awareness helps that they do not bump into each other and that their hands do not collide at the whiteboard. The notes written on the whiteboard help to decide whether to add an own idea and where to place it.

PP and eDPP require very concerted (mental) action of two people and thus very fine-grained coordination of physical and mental activities.

Anticipation Anticipation means that the participants of a collaboration align their actions not only based on what others actually do but also on what they assume the others will do next. Anticipation helps to align the next task-related action, to avoid conflicts with others, or to prospectively offer resources to others they will need for their next action. (Gutwin and Greenberg, 2002)

The intention to correct a typo just made by someone during brainstorming on a whiteboard is an example of anticipation. A prospective offer based on that observation could be that someone hands him the eraser.

Jump In Helping each other is an essential element of collaboration. To sensibly help others, one needs *"to know what they are doing, what their goals are, what stage they are at in their tasks, and the state of their work area"* (Gutwin and Greenberg, 2002). Only then, appropriate intellectual or material help can be provided (Gutwin and Greenberg, 2002).

Applying this to the brainstorming situation at the whiteboard, when someone tries to sketch an idea but has difficulties to express the idea through drawings, someone with a talent for drawing can jump in.

After all, **workspace awareness strikingly impacts the usefulness of a groupware system**. Thus, providing adequate workspace awareness is a fundamental design issue when developing groupware.

1.4.5 Distributed Pair Programming (DPP)

DPP, involving the *"two key fundamentals of pair programming [...] communication and collaboration"* (Bandukda and Nasir, 2010), is **promising to help overcoming some of the tremendous challenges of distributed teams** (Canfora et al., 2003). PP is a tight collaboration where participants intensively communicate using a common language to exchange their ideas and knowledge. They exchange knowledge, learn to assess the other one's expertise, personality as well as communication and working style.

This in turn helps to build trust, also in a distributed team: Over time, and in particular when the pair combinations change, the team members become familiar with each other due to the intense collaboration and communication during eDPP sessions. They also know whom to reach for which kind of questions and feel less timid to contact relevant persons. In summary, this fosters better team spirit and more effective team work. (Stotts et al., 2003)

After all, to judge over the effective usefulness of eDPP, it is important to understand how eDPP actually works out in a distributed setting: whether it is efficient and whether virtual teams can benefit from the team and product advantages as well as from the *"pair feelgood factor"* (Bandukda and Nasir, 2010) of PP.

Even for PP, no one was able to establish a holistic process understanding. As mentioned before, the research results in this respect are rare. Research so far mainly focused on measuring effectiveness of PP in university contexts, but was not able to come up with consistent, replicable findings (Hannay et al., 2009; Sillitti et al., 2012). This unsatisfactory and inconsistent state of research is a possible indicator for the complexity of developing a qualitative understanding of the interactions during PP and their interplay. **Presumably, the distributed situation in eDPP adds an extra level of complexity** to the process and thus to understand it.

In order to achieve reproducible results and to be able to give sound advice, however, **a qualitative understanding of the eDPP process and the circumstances that make it successful is vital**. As will be seen in the next paragraphs, this critical basis has rather been neglected in research so far.

1.4.5.1 Efficiency of DPP

"Teams involved in distributed pair programming are not shown to be worse in terms of productivity than those forming virtual teams without distributed pair programming" (Baheti et al., 2002a).

Experiments compared the efficiency of DPP and PP as well as that of virtual teams doing DPP and virtual teams not doing DPP. Neither could clearly and statistically significantly show that one outperforms the other. The results indicate that the degree of productivity of distributed pairs doing DPP is slightly higher than that of collocated pairs or virtual teams not doing DPP. (Baheti, 2002; Baheti et al., 2002a; Bandukda and Nasir, 2010; da Silva Estácio and Prikladnicki, 2015; Hanks, 2008; Stotts et al., 2003)

An obvious conclusion of these mixed results, though, remains that the performance of DPP is about the same as that of PP. The results' quality of DPP, PP, and virtual teams not doing DPP is comparable. (Baheti, 2002; Baheti et al., 2002a; Bandukda and Nasir, 2010; da Silva Estácio and Prikladnicki, 2015; Hanks, 2008)

After all, DPP *"is a feasible way of developing software"* (Stotts et al., 2003). It can produce *"functional software in reasonable time"* (Baheti et al., 2002a) while maintaining PP benefits like *"pair pressure, pair learning"* and *"two brains"* (Stotts et al., 2003). When the efficiency of DPP seems to be okay and distributed teams can benefit from most of the PP benefits, a more detailed examination of DPP regarding its successful implementation and use for industrial purposes seems to be worthwhile.

Beyond that, DPP efficiency-studies used tools that were not explicitly designed to support DPP and thus were rather cumbersome or limiting the interaction possibilities of the participants. **It is quite**

plausible that the productivity of DPP can be improved by using tools that explicitly support the activities characteristic of PP or DPP (Urai et al., 2015).

After observing local pair programmers, Hanks (2003) identified three characteristic behaviors in PP: role changing, conversation, and gesturing. He conjectures that DPP is quite possible with a tool supporting these activities. Urai et al. (2015) explicitly name a synchronized editor, a shared file repository, the possibility to change the programming partner, and means for communication that must be easy to use and as close to the natural way of human communication as possible. Rosen et al. (2010) add the need for workspace awareness support in an efficient DPP tool. This points out that, beyond the need for understanding the process and human aspects, **the successful and efficient usage of DPP also involves tool-related research and development**.

With respect to video usage to support a natural and efficient communication research cannot draw a clear picture (Schümmer and Lukosch, 2007) about its benefits for group collaboration. In particular for DPP, its value is quite questionable. It has been observed that collocated pair programmers mainly look at their screen, even while talking to each other (Hanks, 2002). Distributed pairs turned their video off for performance reasons. They did not find it large enough to meaningfully transmit non-verbal communication cues like facial expressions, it *"was too small to provide them with any communications enhancements"* (Stotts and Williams, 2002). An interesting perspective on this disinclination is that of Hollan and Stornetta (1992), stating that a video is an imitation of eye contact (whereas VoIP is original) and when *"we make a choice between two channels to use for informal interaction, discrepancies between the two channels are decisive. Thus, if one channel is half as good as another, we don't use it half as often, we probably don't use it at all, so long as the other is readily available"* (Hollan and Stornetta, 1992). Accordingly, the benefit of an integrated video-chat is not clear. Nevertheless, for developers preferring to have a video-chat, a set-up in the manner described by Kutner (2013) is conceivable: *"Joe has multiple monitors and a retractable arm that holds an iPad for running his* Skype *sessions"*. The impact of such an aside video-chat on the DPP process could be an interesting topic for further research.

In other settings, video-walls in a coffee lounge were stated to be valuable for face-to-face communications (Stotts and Williams, 2002). This, however, is a different collaboration situation, not involving a specific, focused collaboration viewport like in DPP. It stands to reason that video only unfolds its potential regarding eye contact and non-verbal cues when made large, when being of high quality, and when not distracting from an object of tight collaboration.

1.4.5.2 DPP – Benefits, Challenges and Recommendations

DPP fosters teamwork with good communication and cooperation: this was confirmed by several student experiments as well as by a field study in a professional context. (Baheti, 2002; Baheti et al., 2002a; Rosen et al., 2010; Stotts et al., 2003)

Despite the distance, many benefits of PP remain, such as knowledge transfer, (positive) pair pressure, or synthesis effect of two brains (Stotts et al., 2003).

In newbie-expert DPP sessions, the expert stated to have successfully conveyed all relevant information to the newbie. The newbies used the DPP sessions to ask questions and discuss open issues while jointly looking at the respective artifact. This applied to sessions where they jointly developed code as well as to sessions where they walked through code that had been written by the newbie. (Rosen et al., 2010) The experienced developer wrote code while explaining it, *"showing a second benefit of DPP as combining teaching with productive work"* (Rosen et al., 2010).

Beyond that, there are more **advantages of DPP compared to PP**:

- No scrambling in front of one monitor: both participants have their own monitor in a comfortable position allowing them to easily see what is on the screen. (Stotts et al., 2003)

- Regardless of the driver, the observer can use his machine to look up something, for example on the internet. (Stotts et al., 2003)
- The collaboration can happen independently of the current abode of the partners — no traveling or location change is necessary (Stotts et al., 2003). This also helps to reduce delays by increasing the pool of potential partners for help. Questions, work results, etc. can easily be reviewed and discussed independent of the location of other (more experienced) teammates (Rosen et al., 2010).
- Jointly created notes or sketches during a session are digital by nature, thus being easy to keep and get out when necessary. (Stotts et al., 2003)
- As in PP, in DPP the participants are less likely to do something off-topic. (Stotts et al., 2003)

DPP also involves several drawbacks:

- The collaboration is highly dependent on specific technology on two computers and thus even more prone to hardware, software, and network malfunction (Stotts et al., 2003).
- Due to the limited suitability like size and transmission quality of webcams, pair members in DPP cannot see each other's facial expressions (Stotts et al., 2003).
- Office colleagues of the pair members may not recognize that the developers are engaged in a remote collaboration and thus may unwittingly disturb them. A solution for this issue could be a simple 'do not disturb' sign at each pair member's desk. (Stotts et al., 2003)
- Quick and easy means for visualizing thoughts like a whiteboard or a pen and a paper cannot be used in DPP. Digital means are still not as natural and straightforward to use as their analogous counterparts. Thus, the pair may spend more time on verbally explaining things that could be sketched more quickly. (Stotts et al., 2003)
- DPP involves a learning curve (Stotts et al., 2003). Rosen et al. (2010) showed that team members first need to become used and sensitized to certain aspects to have successful DPP sessions. They identified the following issues when establishing DPP in a multi-cultural environment:
  - Conflicts with understanding of roles: Newbies did not feel capable to take over the driver role but saw themselves in the role of a passive learner. The expert in turn would have liked the newbie to actively participate in the collaboration and felt exhausted after a session of being the solo entertainer.
  - Ambiguity about session goals: Each participant has different implicit expectations of a session. When reflecting a walkthrough-session, the expert stated that he expected a presentation of the newbie's code, whereas the newbie expected a commented review of his code by the expert. Accordingly, the newbie was not properly prepared and the ad-hoc presentation of his code was humpy. As a result, the session was inefficient.
  - Missing awareness: The participants did not verbalize their thoughts alongside their actions and did not use the tool's built-in awareness features either despite being told about them. For example, they did not use the tool's awareness annotation about the other one's location in the workspace (current file and line) nor used their remote cursor to highlight a code segment they were referring to. Instead, they asked the other one in which file he is and referred to line numbers when talking about a specific section in the artifact.

The aforementioned issues lead to a less efficient DPP process, in particular with respect to code creation and knowledge transfer (Rosen et al., 2010). Based on these insights, the following recommendations for making DPP more pleasant and efficient have been derived:

- Before working together, people should be familiar with each other (Canfora et al., 2003; Stotts et al., 2003).

- An understanding of the PP/ DPP principles, roles and the importance of role switches helps to avoid unproductive sessions. The participants will neither be bored nor exhausted (depending on the role and activity level they were in before). (Rosen et al., 2010)
- Participants of DPP must be absolutely willing to verbalize their actions and thoughts – otherwise the other one, in particular the observer, gets lost (Stotts et al., 2003). This is vital for the mutual awareness of the other one's actions and train of thought. Stotts et al. go as far as saying that personalities not able or willing to do this thinking aloud should refrain from doing DPP. Beyond that, like in PP, the success of DPP depends on other personality traits and openness of a person, for example their fears and attitudes (Rosen et al., 2010).
- The technical infrastructure should at least allow the participants to exchange code snippets (Stotts et al., 2003).
- The expectations regarding a session's goal should be explicitly settled in advance and early enough in case it requires preparation of one or both participants (Rosen et al., 2010).

1.4.5.3 DPP Research – Outstanding Issues

DPP research aiming at the efficiency and product quality of DPP has mostly investigated students. Few details about the characteristics, environment, and background of the students were given (da Silva Estácio and Prikladnicki, 2015). The students often used screen- or application sharing tools that do *"not provide any specific tool support for distributed pairing"* (Hanks, 2008) although *"it seems clear that a tool that supported the needs of the distributed pair would allow them to collaborate more effectively"* (Hanks, 2008). Consequently, **the actual impact of workspace awareness in DPP or eDPP has not yet been explicitly addressed**. Beyond that, it is quite likely that the collaboration process differs depending on the awareness information and interaction possibilities a tool provides.

To sum up, **little is known about eDPP done by professionals using dedicated tools**. Some initial recommendations were provided by Rosen et al. (2010). Nevertheless, the eDPP process with appropriate tools allowing to unfold its potential has not yet been examined.

To understand the professional usage of eDPP and to formulate patterns and anti-patterns for successful eDPP, the following questions should be answered:

- What actually is an eDPP-specific phenomenon?
- What is the direct effect of an eDPP-specific phenomenon on the process?
- What is the session-wide effect of an eDPP-specific phenomenon?
- Zieris and Prechelt (2014) criticize the ignorance of the human factor *"PP skill"* in PP research. This applies to eDPP, too. So, what are eDPP skills that are helpful for successful eDPP? Such an understanding is the base to be able to provide advice about beneficial or harmful behavior in eDPP sessions.
- What are other relevant contextual factors for eDPP, like session goal, type of task, etc.?
- What are relevant personal factors for eDPP, i.e. personality type, language skills, cultural background? Plausibly, eDPP is even more prone to conflicts than PP due to cultural differences or problematic character traits.
- Hannay et al. (2009) claim that PP research does not adequately address the moderating factors of PP. What are moderating factors of eDPP?
- How are DPP tools designed, implemented, and used? (da Silva Estácio and Prikladnicki, 2015)
- DPP is often compared to solo programming or PP. How could it be used for other types of collaborative programming, i.e. with more than two participants? *"Although some DPP tools*

enables [sic!] more than two programmers, for instance Saros, there is no empirical evidence related to distributed group programming" (da Silva Estácio and Prikladnicki, 2015).

- DPP was often examined in university settings. How is it applied in professional contexts? (da Silva Estácio and Prikladnicki, 2015).
- What are the principles and lessons learned from DPP in professional contexts? (da Silva Estácio and Prikladnicki, 2015).

This thesis addresses several of the aforementioned questions: It aims at **understanding how and why professional developers may successfully collaborate via a dedicated eDPP tool on a regular daily basis**.

Before introducing the particular tool used for this study, current DPP tools shall be presented in an overview.

Chapter 2

eDPP Requirements and Tools

"Removing the collocation requirements of pair programming would increase its flexibility so that its benefits could be enjoyed by a wider audience. A tool that supports distributed pair programming (DPP), in which the driver and navigator pair from separate locations, would remove this impediment to the adoption of pair programming" (Hanks, 2008).

To investigate the eDPP process, an eDPP tool is needed. **Although the study's main purpose is neither testing nor developing eDPP tools, it requires an eDPP tool**. And in order to observe a realistic process and not tool-specific process phenomena, the tool has to be an unobtrusive, state-of-the-art alternative for eDPP. The testbed used in this study is *Saros*. This chapter discusses its maturity and adequacy for eDPP.

Chapter Structure – *What to expect in this chapter and why*

The chapter starts with discussing requirements of eDPP tools in Section 2.1 "Requirements for eDPP Tools". Subsequently, an overview of the actual product landscape of DPP tools is given in Section 2.2 "eDPP Tools" and they are compared according to the concrete requirements for eDPP tools. Finally, Section 2.3 "Saros" describes the features and awareness annotations of *Saros* in detail. Since *Saros*' development was in no way straightforward and took a good deal of the time of this research project, Section 2.3.1 "Saros History" provides an overview of the historical development of *Saros*.

2.1 Requirements for eDPP Tools

This overview of requirements for eDPP tools is based on the two pioneering works in the specific area of requirements for real-time collaboration. First, Section 2.1.1 "Adapting the Workspace Awareness Framework to eDPP" goes through the workspace awareness framework of Gutwin and Greenberg (2002) with the focus on its applicability for eDPP. Second, concrete patterns for eDPP collaboration are discussed in Section 2.1.2 "CMI Patterns for eDPP".

Before discussing requirements of eDPP tools, the general purpose and usage context of an eDPP tool is specified as follows:

- an eDPP tool is considered to be part of a landscape of development tools like for example for project planning, task management, version control, configuration management, etc. Information

and functionality provided by these tools are not in the focus of an eDPP tool. The considered purpose of an eDPP tool is to support the activity of real-time collaboration and its setup.

- Collaborative programming like eDPP is a synchronous, mentally tightly coupled activity where the participants work with a shared mind on the same task. The outcomes of the collaboration, that is the artifacts, evolve from this collaborative effort and not from the integration of individual, asynchronous work contributions.
- The participants in PP as well as in eDPP usually know each other or, if not the case, they get to know each other prior to a collaborative session. During a session, the partners do not change – in contrast to a collaboration in a (virtual) room where participants may leave and others come in. Accordingly, features for supporting 'surrounding' group processes, like group forming, getting to know each other, or partner change, are not considered as a requirement for eDPP tools.

Although different tools may provide several features that go beyond session initiation and the pure collaborative editing process, they are not considered here due to their irrelevance for the pure eDPP process which is subject of this work.

2.1.1 Adapting the Workspace Awareness Framework to eDPP

Since the workspace awareness framework initially considers other types of collaborations than eDPP, this section is about the adaption of the workspace awareness framework to the eDPP setting.

Workspace awareness support is a crucial requirement for eDPP tools. The workspace awareness elements defined in the workspace awareness framework by Gutwin and Greenberg (2002) are a suitable yardstick for comparing the awareness information provided by different eDPP tools. Nevertheless, the eDPP situation and the collaborative situations considered in the workspace awareness framework differ in some aspects. This in turn has an impact on the applicability of the awareness elements of the framework for eDPP. To see the relevance or non-applicability of the various awareness elements for eDPP, the similarities and differences of the situations considered in the workspace awareness framework and the eDPP setting are reflected.

The elements of the workspace awareness framework were derived from face-to-face collaborations of two to five participants working on flat workspaces on artifact-producing tasks like 1. two people organizing slides on a light table, 2. a research group generating ideas on a whiteboard, 3. project managers planning a task timeline. (Gutwin and Greenberg, 2002)

A crucial difference between the observed face-to-face interactions and eDPP (as well as PP) is that the **artifacts in eDPP (and PP) are of immaterial or virtual nature. They are modified indirectly by operating the computer**. The artifacts do not give off characteristic sounds by themselves when being modified. Only the interactions with the computer, like keystrokes, mouse movements or clicks, can be overheard, but these are not artifact-specific.

Such aspects considerably affect the availability of awareness information and how people can gather awareness information in eDPP. The following overview outlines the significant differences of the types of interactions considered in the workspace awareness framework and those of eDPP.

- **Group Size**
  - Workspace awareness framework: 2-5 people (Gutwin and Greenberg, 2002)
  - eDPP: 2 people
- **Environment**
  - Workspace awareness framework: The workspace is a flat surface, *"upon which objects can be placed and manipulated"* (Gutwin and Greenberg, 2002). These objects are the focus and subject of the collaboration (Gutwin and Greenberg, 2002).

  - eDPP: Workspace consists of non-physical, editable source code files, usually organized in a folder structure. The artifacts are manipulated via non-physical means instead of physical tools.

- **Types of activities**
  - Workspace awareness framework: Mixed-focus collaboration where the participants switch between individual and shared activities. Tasks are accomplished by creating or manipulating visible artifacts. This involves activities like *"construction (page layout, diagram assembly), organization (arranging, ordering, or sorting artifacts), design (drawing, generating an outline), or exploration (finding certain types of artifacts in the space). Other types of tasks (e.g. decision-making) also involve workspace awareness, but as these types involve less interaction with the artifacts, we do not consider them as primary for the framework."* (Gutwin and Greenberg, 2002)
  - eDPP: Tightly coupled activity with a shared mind. Since in eDPP the participants can independently move in their workspaces, also mixed-focus collaboration in different nuances is possible (it is these nuances this thesis focuses on). Tasks are accomplished by creating/modifying program artifacts in written form. This also involves activities like exploring, searching, testing, etc. eDPP essentially comprises intellectual activities like problem solving and decision making which are usually strongly connected to the focused artifact.

- **Considered groupware systems**
  - Workspace awareness framework: Gutwin and Greenberg (2002) applied their insights from observing face-to-face collaborations to real-time distributed groupware providing a shared workspace.
  - eDPP: Real-time distributed groupware providing a shared workspace.

2.1.1.1 Adapting Part One: How Workspace Awareness Information is Gathered in eDPP

As already detailed in Section 1.4.4 "Workspace Awareness in Real-Time Groupware", the workspace awareness framework lists three main sources of awareness information:

- **People's bodies in the workspace**: the position and movements of the body or parts of it (hands, arms, head, eyes) when performing an action. Such information are a significant part of the overall picture of what is going on in the workspace. (Gutwin and Greenberg, 2002)
- **Workspace artifacts**: their spatial relationships[31] as well as their physical appearance[32] represent the status of the task. This information helps to grasp the state of the current work. Moreover, when seeing modifications of an artifact, this allows to infer what someone is currently doing with that artifact. (Gutwin and Greenberg, 2002)
- **Conversations and gestures**: Verbal communication is a rich source of information for those involved as speakers or those who listen. Non-verbal communication like facial expressions or gestures that substitute, support or complement phonetic language are also an essential information source in a shared workspace. (Gutwin and Greenberg, 2002)

In PP and eDPP, the workspace is different to that of groups collaborating face-to-face on a flat surface. Accordingly, there are also differences in the available awareness sources. The workspace situation is even different for PP and eDPP. Since the eDPP setting is more limited regarding the availability of awareness information, the deviations will first be discussed for PP, and then for the more constrained eDPP situation.

[31] position relative to the workspace and proximity to one another
[32] graphic or textual presentation, their nature, and state of development

PP/ eDPP deviations with regard to people's bodies in the workspace In PP, all artifacts are in a virtual workspace represented on a monitor. People's bodies are not part of that workspace and accordingly, the position and posture of them do not play a role within that workspace. Only for the driver there are virtual tools, like his text and mouse cursor and his focused files, which indicate his mental focus in the workspace.

The workspace in PP is of mixed nature. In addition to the virtual workspace, the pair members sit next to each other and share a desk and its surroundings, a physical workspace. Accordingly, the interactions with the computer are visible for the other part. The participants are not handling physical artifacts, though, and all artifacts in the virtual workspace are edited by similar, rather subtile interactions with the computer. There are no characteristic bodily actions for certain modifications of artifacts. Movements of heads, arms, eyes, and hands are more subtle and therefore harder to recognize. Additionally, these information are unilateral since only the driver is operating the computer. Nevertheless, the bodies of both participants as well as their level of activity, their posture and facial expressions provide hints about their presence, mood and level of interest.

In eDPP, the participants do not share a physical environment, and their physical interactions with the computer are not visible for the other one. Both are limited to the remote virtual tools and the awareness information provided by the groupware. Examples of such tool-provided awareness information are annotations about the other one's opened files, his focused artifact, or his current viewport.

Virtual Tool *(Explanation of terms)*

> Virtual artifacts cannot be physically handled. Users need technical means to operate them. A text cursor or a mouse pointer are virtual tools. They are non-physical means to manipulate virtual artifacts and to navigate in a virtual workspace. Therefore, they represent the users' position in the virtual workspace and are a kind of lengthening of users' physical limbs.

Remote Virtual Tool *(Explanation of terms)*

> A remote virtual tool is a virtual tool of one user which is visible in the workspace of another, remote user. For example, one user also sees the caret position of the remote participant in his own workspace. Or when one participant uses his mouse cursor to select a code snippet, this selection is highlighted in the remote partner's view, too. For more details see Remote Selection, Remote Cursor, or Telepointer in Section 2.1.2.4 "Group Support – Synchronous Group Awareness".

Compared to the face-to-face situations considered in the workspace awareness framework, as a result, **in PP as well as in eDPP, the position of the body as well as bodily actions in the shared workspace, and thus consequential communication (see page 70), lose some of their relevance**. The mouse and keyboard operations observable in the virtual workspace are more expressive concerning the events in the workspace. Both participants' position and actions are equally represented in the workspace. Thus, **compared to PP, the situation turns from an asymmetric to symmetric setting but limited to the virtual shared workspace and the communication channels**.

PP/ eDPP deviations with regard to information provided by workspace artifacts In PP, the artifacts in the virtual workspace are usually arranged in folders and packages, which to some degree expresses something about their semantic relationship. They do not have physical distances in relation to each other, however, like physical objects on a flat surface have. Virtual artifacts do not have a physical appearance in that sense but their textual/code representation reveals their meaning and reflects the state of progress. In PP, the driver makes changes in the artifacts. In combination with other knowledge, this allows the observer to infer his train of thought and his subsequent actions. Vice versa, for the driver it is much harder to infer or 'read' the thoughts of the observer, in particular if he is not communicative enough.

In eDPP, the participants jointly edit the artifacts. Due to the non-existence of a shared physical environment, the changes in these artifacts are a major source of information of what is happening in the shared virtual workspace. Accordingly, feedthrough (see page 70) is the major mechanism for inferring actions of the other one. Some changes can be directly observed, like for example the steady flow of the partner's typing. Other changes, however, such as the result of an indirect manipulation, are harder to grasp. For example, when the partner uses shortcuts, unshared dialogs or menu commands to modify the artifact, suddenly the result appears (e.g. a code refactoring), and it remains unclear how the artifact came into that state.

PP/ eDPP deviations with respect to information provided through conversations and gestures The verbal communication channel remains available in PP as well as in eDPP due to an assumed audio connection. In PP, it is available in its full richness, whereas in eDPP, the quality of the technical connection very much determines the perceptibility of subtle nuances and utterances. It can also be more difficult to understand heard utterances if the complementing visual context information, that would provide a point of reference for this utterance, is missing.

In PP, the pair members can use gestures and facial expressions. Since the participants sit next to each other, jointly looking at the monitor, physical gestures and in particular facial expressions require an explicit look at each other. It is not automatically part of their respective fields of vision when both gaze at the monitor. Additionally, the driver can perform gestures in the virtual workspace via his virtual tools. However, this possibility is unilateral, only the driver can gesture in the virtual workspace. The observer can use his forefinger to point at something on the shared monitor (and usually does it in PP).

In eDPP, gesture and facial expressions cannot be used. Nevertheless, both participants can gesture like pointing or selecting something with their virtual tools.

2.1.1.2 Adapting Part Two: Deriving Awareness Requirements for eDPP

As discussed in Section 1.4.4.1 "Part One: What makes up Workspace Awareness", for face-to-face collaboration the workspace awareness framework considers five basic categories of workspace awareness elements, i.e. aspects of a collaborative situation that people perceive. In the following paragraphs, they shall be considered in the context of PP and eDPP.

The awareness elements are *"commonsense things that deal with interactions between a person and the environment"* (Gutwin and Greenberg, 2002). Their concrete nature and level of detail depends on the specific collaboration – the domain, task, dynamic of collaboration, and work environment (Gutwin and Greenberg, 2002). Due to their fundamental nature, these *"elements are a starting point"* (Gutwin and Greenberg, 2002) to consider awareness information in a particular collaborative situation, and thus to derive awareness requirements for groupware supporting such a situation. For that reason, this section discusses the awareness elements for the PP situation (the 'original' collaborative situation) and, based on that, derives plausible awareness requirements for eDPP groupware. Of course there are *"additional kinds of information specific to the task or the work setting"* (Gutwin and Greenberg, 2002) which have to be figured out for specific collaboration settings. Specific DPP and eDPP collaboration patterns are addressed in the next Section 2.1.2 "CMI Patterns for eDPP".

'Who' Awareness 'Who' encompasses information about the actual and past presence and identity of others in the workspace. This involves the ability to ascribe observed activities to the identity of their originator. (Gutwin and Greenberg, 2002)

As shown on page 69, this comprises the following elements:

- Presence: Is anyone in the shared workspace?
- Presence history: Who was here and when?

- Identity: Who is participating and who is that?
- Authorship: Who is active?

Prior to a PP session, this is the knowledge about who is and was available for a collaboration, and during the session, it is the physical attendance and identity of the partner. The observer perceives the driver with all senses, and in particular sees his actions. In this way, the observer easily knows that the changes in the artifacts are caused by the driver's interactions with the computer.

Requirements for 'who' awareness support in eDPP tools:

- Authorship: eDPP tools allow concurrent editing activities. The authorship of changes a participant has not initiated by himself can easily be ascribed to the other participant. To facilitate matters, changes in the workspace usually are annotated with authorship-information, often by assigning a specific color to each participant and highlighting his changes with that color.
- Presence prior to a session: For a quick and easy session initiation, an eDPP tool may show who is available for a joint session, although this could also be agreed upon via other channels like email or instant messaging.
- Presence during a session: During a session it is relevant to know that the collaboration partner is still in the shared workspace, when he has switched to another application, or if he has left his desk.
- Identity information prior to a session: This information is not directly in the focus of a pure eDPP tool. However, if the tool has a kind of buddy list for session initiation, it is helpful to identify a collaboration partner from a list of available partners.
- Identity: During an eDPP session, this information does not change and the participants usually know each other.
- Presence history: Presence history is not an explicit subject matter of eDPP tools. To some degree, version control systems like svn[33] or git[34] provide means to see who made changes to an artifact and when, like for example with 'blame' commands[35].

'What' Awareness 'What' awareness is knowledge about the activities of others: what are they doing in which artifact and what overall goal does this action relate to (Gutwin and Greenberg, 2002).

As shown on page 69, this type of knowledge comprises the following elements regarding the collaborators in the workspace:

- Action: What is a person doing?
- Action History: What has a person been doing?
- Intention: What goal is that action part of?
- Artifact: What object are they working on?

In PP, the observer is part of the driver's intellectual work and carefully observes his activities. Therefore, he knows on a very detailed level which artifact (i.e. class, method, expression, test case) the driver is working on, what he is doing, and which overall session goal or task this relates to.

In eDPP, both participants can independently edit and move within and among artifacts. In theory, knowing what the other one is doing and why he is doing that is independent from actually seeing those activities. Plausibly, however, there is a strong relationship between the visual congruence (same focus, same viewport with different focus, or different viewports) and the degree of awareness and

[33] `https://subversion.apache.org/` (accessed November 11, 2017)

[34] `https://git-scm.com/` (accessed November 11, 2017)

[35] `https://help.github.com/articles/using-git-blame-to-trace-changes-in-a-file/` (accessed November 11, 2017), `http://svnbook.red-bean.com/en/1.7/svn.ref.svn.c.blame.html` (accessed November 11, 2017)

understanding of the other one's actions. Compared to a physical flat surface with several movable and visible objects, in a virtual eDPP workspace, only one object is visible at a time, or, in case of split screen, maybe two objects next to each other. Accordingly, in eDPP, the participants can only monitor the other one's actions when they are in the same viewport.

Requirements for 'what' awareness support in eDPP tools:

- Artifact: An eDPP tool has to show what artifact the partner is working on and, if part of the shared workspace, it should provide means to directly open this artifact.
- Action: When working in the same artifact in a tightly coupled work mode, detailed editing activities and other operations of both participants must be visible in real-time for the other participant. This steady tracking is quite important. Seeing the continuos development of the artifacts helps the participants to get *"a chance to form a correct mental model about cause and effect"* (Tidwell, 2010) with regard to events in the workspace. In that way they can follow the intellectual process of the partner. This allows the collaboration to be a fluent process, which in turn *"can induce a state of flow in the user"* (Tidwell, 2010). Real-time feedback about the other one's actions is vital in distributed collaboration because otherwise the users can hardly understand what is happening on the remote site and thus not establish a fluent collaboration process. Code artifacts may change due to direct manipulation like writing code, or indirectly by executing commands that generate/modify code. Ideally, all types of activities are made transparent and transmitted to the remote partner.
- Action History: Although it is not required for the actual session, eDPP tools can track the actions of the participants (on an appropriate level of detail) and provide a kind of session recording or timeline of events. This can be helpful in particular for teaching others or help them getting into a part of code.
- Intention: The intention of a particular action cannot be represented in an eDPP tool. The understanding of the overall task and its sub-steps arise from a common understanding of the task and a shared mental model of how to approach it. Even in PP, intention cannot be read from the partner's mind. In addition to knowing the shared work state and goal, nonverbal communication cues and observing the other one's activities help to understand the partner's actions and the intention behind them.

'Where' Awareness 'Where' Awareness is knowledge about where in the shared workspace a collaborator is located, what parts of the workspace can be seen and which areas of the workspace are available.

As shown on page 69, this type of knowledge comprises the following elements:

- Location: Where are they working?
- Location history: Where has a person been?
- Gaze: Where are they looking?
- View: Where/what can they see?
- Reach: Where can they reach?

"In some situations, certain elements never change, and so do not require explicit support in the interface" (Gutwin and Greenberg, 2002). In PP as well as in eDPP, the workspace is of immaterial nature, the participants cannot reside at a specific physical position within the workspace. So, assuming that both participants properly sit in front of the computer, the detailed physical location or position of the participants is not relevant with regard to what they can see or reach. In the virtual workspace, the information on what artifact a person is working is more substantial, as discussed above in the context of 'what' awareness.

In PP, the pair sits next to each other in front of the same computer, having a congruent view. When the driver is operating the computer, the observer sees where the driver is working and – fairly precisely – knows what he is looking at. While gazing at the monitor, it is not that clear which precise part of an artifact the driver is actually focusing on, but the observer still has a fairly good idea of the area the driver views.

'Where' awareness can in eDPP tools be supported in the following manner:

- View: In an eDPP tool, annotations about the current viewport of the remote partner narrow down his possible focus in an artifact. Nevertheless, this cannot be used as definitive evidence of his mental focus.
- Gaze: In theory, gaze information could be traced with eye-tracking cameras. However, in reality they are not used in eDPP and thus gaze information cannot be gathered and shown. Only visible actions of the remote partner provide fairly reliable indicators about what he is gazing at.
- Reach: An eDPP tool should provide means to select which files of a project or folder to share with the partner. Moreover, it can allow to limit the editing rights for shared files, for example by supporting and annotating the differentiation of read and write access.

'How' Awareness 'How' Awareness relates to the past and is knowledge about how artifacts developed and how operations were executed.

As described on page 69, this category comprises the following elements:

- Action history: *"How did that operation happen?"*
- Artifact history: *"How did this artifact come to be in this state?"*

In PP, the driver performs the operations while the observer follows him. Both have an understanding of what happened in which way in the workspace during their session. In eDPP, this knowledge depends on the visual coupling of the participants and their consistent understanding about what the other one is doing in which artifact. Such information relating to the past could be supported by eDPP tools with an action history discussed in Paragraph 2.1.1.2 "'What' Awareness", but for a running session it would rather hinder the collaboration process.

'When' Awareness 'When' awareness relates to the past. The 'event history' is knowledge about the time of occurrence of operations/events in the workspace.

In PP, the participants easily perceive what is happening in their workspace and thus have an idea of the chronology of events. As with 'how' awareness, in eDPP, this knowledge strongly depends on the visual coupling of the participants and the quality of the transmitted awareness information about the other one's activities. Again, in theory, such information could be provided to some degree of granularity as part of the action history discussed in Paragraph 2.1.1.2 "'What' Awareness", but for the actual session its practicability is quite doubtful since usually the final artifact or work result is more relevant than the chronology of events that led to that result.

After having discussed some workspace awareness requirements for eDPP groupware based on the workspace awareness framework by Gutwin and Greenberg (2002), a sound base was provided for understanding the general aspects to consider when developing eDPP groupware. In the context of the workspace awareness framework, some awareness widget prototypes were developed and tested, but in collaboration situations different from eDPP. Beyond that, the rather conceptional framework does not talk about concrete approaches of how workspace awareness needs can be realized in groupware. To address that gap, the next section discusses concrete patterns for groupware systems to facilitate the interaction of groups.

2.1.2 CMI Patterns for eDPP

Schümmer and Lukosch (2007) summarized *"the best practices in computer-mediated interaction and captured them in a pattern language"*. That pattern language condenses work of several years in the area of computer-mediated interaction (CMI), is of great length and detail found nowhere else in literature, and explicitly covers issues of real-time collaboration. With this it provides structure and a uniform vocabulary to describe and compare the functional elements of eDPP tools:

"*These patterns capture commonly used collaborative system design solutions and thus allow us to describe a hypothetic DXP [(distributed extreme programming)] system. We also use the patterns to compare existing solutions with the hypothetic solution by identifying the presence of patterns in the existing solutions.*" (Schümmer and Lukosch, 2008)

In that sense, this section establishes a vocabulary as a basis for the comparison of eDPP tools in the following Section 2.2 "eDPP Tools". It starts by discussing the structure of the pattern language; subsequently the eDPP-relevant elements are described in more detail.

The main task of eDPP groupware is to support the collaborative editing of artifacts in a shared workspace and to provide means to start and end such a collaboration in a simple manner. Beyond that, eDPP tools may provide supplemental functionality. For this work, however, the ongoing eDPP session is of major interest. Accordingly, the following CMI patterns are considered with regard to that core business.

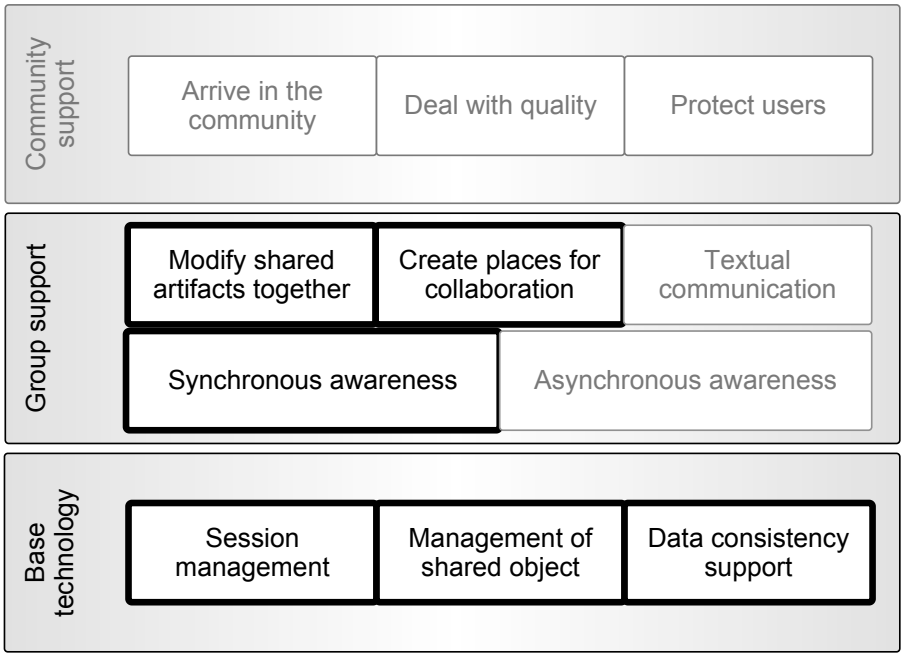

Figure 2.1: Structure of the pattern language by Schümmer and Lukosch (2007)

The pattern language consists of three layers, each involving clusters of general issues to consider for the development of groupware systems (Schümmer and Lukosch, 2007). Layers and clusters which are not directly relevant for eDPP tools are shown in light grey.

Figure 2.1 illustrates the three **layers of the pattern language**. Each layer addresses different aspects of computer-mediated group collaboration. The layers build on one another, each involving general issues that have to be considered when developing groupware. The issues raised in each cluster are addressed by the concrete patterns which represent a functional component of a groupware application:

- **Base technology**: The patterns on this level describe approaches for handling infrastructural

challenges like managing the collaborative activities, the shared data, and data consistency. They aim at developers *"who have to work out how shared objects should be managed and how information exchange is mediated by the computer system"* (Schümmer and Lukosch, 2007).

- **Group support**: This level focusses on the actual collaborative work of the group members – how to support their communication, their shared activities, and their mutual awareness. Patterns on this level address the establishment of collaborative work as well as the actual collaboration.
- **Community support**: Patterns on this level are about building and maintaining a community – how to welcome new users, how to keep members in line, and how to protect their work and privacy. The patterns primarily describe social processes and how to support them in groupware. Intention of these patterns is to keep the interest in the community *"as high as possible and make the threshold for entering the community as low as possible"* (Schümmer and Lukosch, 2007).

Except for the base technology, the patterns mainly target at the end user. Each pattern is an abstract solution for a common CMI issue which has to be adapted to the specific context it will be used in (Schümmer and Lukosch, 2007). For groupware developers, the patterns can act as a toolkit or collection of possibilities from which they can choose to manage the various challenges in groupware design. To help to assess the appropriateness of a pattern for a specific problem domain, Schümmer and Lukosch (2007) describe each pattern in the following structure:

- Name: The name of the pattern.
- Sensitizing picture: A picture illustrating an aspect of the pattern and visual aid for a better memory.
- Intent: The key point of the pattern, formulated in one sentence.
- Context: A description of a possible *"situation in which the pattern is intended to be used"* (Schümmer and Lukosch, 2007).
- Problem: A description of problems that can occur in that situation.
- Scenario: *"Scene-setter"* (Schümmer and Lukosch, 2007) illustrating the problem with an example setting used throughout all patterns.
- Symptoms: Specification of contextual conditions that help to identify when the pattern is appropriate.
- Solution: How to tackle the stated problems and symptoms.
- Dynamics: The pattern's components and actors and how they interact.
- Rationale: Explanations how the pattern works and why it provides a solution for the stated problems.
- Check: Questions that help to adapt the abstract pattern with application-specific details.
- Danger Spots: Potential negative side effects of the pattern.
- Known Uses: Real-world usage examples of the pattern.
- Related Patterns: Other patterns concerned with similar issues.

As with awareness requirements, not all CMI issues and patterns are relevant for each groupware. The domain-relevant elements must be identified for a specific application. In the following, each level of the pattern language is discussed with regard to its relevance for eDPP groupware.

For an eDPP session and thus in the context of this dissertation, the community support level is not considered relevant. Applications that support eDPP have a very narrow intended use. They focus on the synchronous, tight collaboration with one known partner. The work result of eDPP is produced via a synchronous, shared activity and not by the integration and exchange of personal contributions and

messages. Accordingly, building and maintaining a community – welcoming newcomers and motivating, managing, and protecting existing members and their work – is outside the scope of eDPP applications.

The **middle layer about group support** refers to the support of the actual group collaboration and is quite important in the context of this work. Users must be able to find together and to establish a collaboration, and to collaboratively access and modify the shared material within some kind of shared virtual place. **The requirements 'modify shared material together' and 'create places for collaboration' are fundamental for eDPP** groupware. Since eDPP is a highly dynamic and synchronous activity, supporting **'synchronous awareness' is also a crucial requirement**. Instead of 'textual communication', for eDPP, verbal communication is essential. Since the verbal communication is orthogonal to the interactions in the workspace, it must not necessarily be provided by the collaboration tool itself. The 'asynchronous awareness' cluster involves patterns about *"how absent users can stay informed about group processes, and how the group can in turn stay aware of absent users"* (Schümmer and Lukosch, 2007) and this is not part of core eDPP functionality.

The patterns of the middle layer consider different types of collaboration. Not all patterns depict reasonable functionality for the tight, synchronous collaboration of two individuals like in eDPP. Section 2.1.2.1 "Group Support – Modify Shared Artifacts Together" outlines the patterns on this level and discusses their relevance for eDPP.

The **base technology layer addresses the technical prerequisites** for the patterns on the middle layer. Users collaborate in a joint technical session. A session is a technical construct to realize the data exchange among the participants and can be considered as *"a technical representation of users, the computer systems on which they act, and the artifacts used"* (Schümmer and Lukosch, 2007). In the session, the users access and modify shared objects. In doing so, **modifications need to be synchronized and shared data must be kept consistent**. Independent from the actual usage context, a system for computer-mediated synchronous or asynchronous interaction must 1. manage shared data (management of shared object), 2. handle parallel input from multiple users ('data consistency support'), and 3. technically connect collaborating users ('session management') .

To have an unobtrusive tool, these technical requirements must be met and work smoothly. Their actual technical realization is, however, not of interest to the user or the collaboration process. Accordingly, they will not be examined furthermore here. Only the base technology of the used testbed, *Saros*, shall be roughly described in Section 2.3 "Saros".

For a specific issue, the available patterns sometimes represent alternatives, and sometimes they can be used in addition to each other. For example, for the decision about how artifacts are accessed, how floor control is realized, or the concept of the virtual workspace, only one approach is used in the groupware. To increase the amount of available awareness information in the interface, for example, multiple patterns may be applied to represent synchronous awareness in various ways.

2.1.2.1 Group Support – Modify Shared Artifacts Together

Patterns in this cluster are about supporting the actual collaborative interaction of small groups. Their appropriateness for a specific *"collaboration context depends on the desired strength of interaction"* (Schümmer and Lukosch, 2007).

- GROUP: When a groupware only supports the interaction among individual members of a community, the community is rather perceived as a set of individuals than as a group – the group awareness is poor. Subgroups of specialists are not perceived as such either, but only individual members are visible and available with their expertise instead of the whole expert group. In general, people cannot interact with a set of members but only with individuals. As a solution, this pattern offers to support the organization of members as groups and that a group can be contacted the same way as an individual user. (Schümmer and Lukosch, 2007)

In the eDPP context, this pattern may be relevant for example to organize a contact list according to the people's expertise, but in view of an ongoing eDPP session, it is not relevant.

- Shared File Repository: One way of loosely coupled group collaboration is that the individual group members individually create their work contributions and then share their (intermediate) results with the group. When these individual contributions are manually passed around, for example as e-mail attachments, this procedure is inconvenient and risky with regard to data loss and incoherence. As a solution, this pattern proposes to store shared files in a shared repository where the users can store and access data. (Schümmer and Lukosch, 2007)

  For managing asynchronous access to the code artifacts in software development, version control systems like Subversion[36] or Git[37] are common Shared File Repositories. However, this pattern is about sharing and accessing files, it does not address collaborative editing. This is where the pattern Shared Editing comes into play. It addresses how to simultaneously edit shared data. (Schümmer and Lukosch, 2007).

- Shared Editing: When group members want to equally and synchronously collaborate on shared data, they need technical means to concurrently edit shared artifacts. As a solution, this patterns proposes to support this kind of tightly coupled collaboration via a shared editor that allows the concurrent manipulation of shared data and which broadcasts changes in real-time to all members while ensuring data consistency. (Schümmer and Lukosch, 2007)

  eDPP is exactly this type of tight collaboration where shared artifacts are collaboratively manipulated. **This pattern reflects the core functionality of eDPP tools**.

- Shared Browsing: When group members collaboratively want to explain, explore, discuss, or review work artifacts, a common visual context facilitates their interaction (in addition to other communication channels). As a solution, this pattern proposes to provide a congruent view on the respective artifacts for all involved. This means that the scrolling and movements need to be replicated for all participants. This replication can follow different navigation strategies: master-slave browsing, anarchistic browsing, or democratic browsing. In master-slave browsing, one participant navigates and the others automatically follow his viewport. Anarchistic browsing may be best summarized by saying 'no matter who moves, all others follow'. There is no specific navigator role, but everyone can navigate in the workspace and all others follow. In democratic browsing, movements in the workspace have to be agreed upon democratically beforehand. (Schümmer and Lukosch, 2007)

  In eDPP, this pattern, in particular with master-slave browsing, can be useful for explaining code to the partner, to make a joint code review, to collaboratively explore source code, or to stay automatically visually coupled during coding.

- Vote: For a distributed group it is difficult to make decisions based on their members' opinions or to generally collect their members' attitudes on certain topics. As a solution, this pattern proposes to provide functionality to run and present a vote. (Schümmer and Lukosch, 2007)

  In a synchronous two-party collaboration like eDPP where the participants communicate verbally, this pattern does not apply.

- Application Sharing: When group members want to view or edit data that is only available via single-user applications, they need collaborative viewing and controlling capabilities for that single-user application. As a solution, this pattern proposes to run the single-user application on a host machine, replicate its view to all other remote participants, and to also allow a host-governed interaction with the single-user application for the remote users. (Schümmer and Lukosch, 2007)

  An eDPP session is realized via Shared Editing. For the collaborative code editing Application Sharing is not appropriate. It may be used to share other development activities than

[36] `https://subversion.apache.org/` (accessed November 11, 2017)
[37] `https://git-scm.com/` (accessed November 11, 2017)

editing, for example for making unilateral operations visible to the partner, like the visual representation of test or search results, or other dialog windows. Application Sharing as an approach for distributed pair programming is discussed in Section 1.2.2 "Remote Pair Programming (RPP)".

- Floor Control: Concurrent actions of multiple users in the same artifact can result in colliding actions. As a solution, this pattern proposes to technically regulate who can access and modify a shared artifact by the possession of a token. There are several possible turn-taking protocols for floor control like time-slot-based, task-based, managed by a human 'floor-chair', etc. (Schümmer and Lukosch, 2007)

  As described on page 62, technically induced or explicit floor control is not unproblematic with regard to the dynamics and smoothness of dynamic collaboration like eDPP.

2.1.2.2 Group Support – Create Places for Collaboration

The following patterns focus on the representation of the virtual place where the group members can meet and jointly work.

- Room: The members of a group need several tools for their collaborative tasks, such as audio or video communication, tools for sharing data, and to collaboratively edit artifacts. The set-up of this piecemeal-infrastructure is time-consuming and inconvenient. As a solution, this pattern proposes to set-up a virtual room to be entered by group members which is equipped with the infrastructure needed for the collaborative tasks. A virtual room holds members, shared objects, communication channels, and provides tools for collaboration, as for example a shared editor. In a room, the members have a shared workspace with a coherent view on the shared data and can easily communicate and collaborate. They may enter and leave the room while the content remains unaffected. (Schümmer and Lukosch, 2007)

  In any case, eDPP groupware shapes a shared workspace. It allows participants to join the virtual workspace, provides means for communication, access to shared artifacts, a distributed editor, potential additional functionality, and workspace awareness. Its conceptual representation (i.e. as a permanent room or a transitory session), naming, and functional spectrum depend on the concrete tool.

- Active Map: This pattern is also referred to as Overview Diagram or Radar View. When multiple members collaborate in different parts of a virtual workspace, they can easily loose orientation about where the others are working and what they are doing. As a solution, this pattern proposes to ease orientation in the workspace by creating an overview (a visual miniature) of the virtual workspace that shows indicators about others in the workspace and their activities. (Schümmer and Lukosch, 2007)

  Originally, this pattern is useful for big, flat workspaces that can be mapped to a miniature representation, and it helps to provide orientation when the group members have non-overlapping viewports. In eDPP, the shared workspace is shaped by the project files and folders. The predominant work mode in eDPP is tightly coupled and implies a visual coupling of the participants. However, due to the independent workspaces, deviations from that work mode are possible and that exactly is the point where this pattern can provide help or orientation. For example, a project outline representing the files and folders in a tree structure depicts a visual miniature of the workspace and could be used for annotations about in which artifact the remote participant is working. When the participants have non-overlapping viewports but are in the same artifact, a Radar View is a possible approach to provide orientation on what section of the artifact the remote partner is working.

- Interaction Directory: Users may find together and collaborate on a specific topic which then depicts a specific collaboration context. Such a collaboration context may involve multiple participants, and in general, a group member can be part of various collaboration contexts. In

the first place, however, the group members must somehow be aware of available collaboration contexts. As a solution, this pattern proposes to arrange a shared space in which collaboration contexts are listed and described, so that users can easily join them or create new ones. (Schümmer and Lukosch, 2007)

During an ongoing eDPP session, this pattern does not apply. Nevertheless, it is conceivable that for example in a company, ongoing or pausing eDPP sessions may be listed on a central server alongside with their task descriptions, and participants can join them according to their interest or expertise.

- Bell: When a group member wants to join an ongoing collaboration session, due to privacy reasons, he should not be able to sneak into the session. Instead, the other participants should be informed that someone wants to join their collaboration. As a solution, this pattern proposes to equip the collaboration space with a bell that allows latecomers to 'knock' on an ongoing session, informing the group that someone wants to join. (Schümmer and Lukosch, 2007)

  For eDPP, this pattern can become relevant if an eDPP session is broadened to multiparty programming, but for a running eDPP session limited to two participants the pattern is not relevant.

- Invitation: When a group member wants to pair up with one or multiple other users, he needs a mean to communicate and arrange his desire. As a solution, this pattern proposes to allow a participant to invite others to a session, instantly or for a specific time. The invitees get the invitations alongside with some descriptive information and can decide to accept or decline the invitation. Members who accept the invitation are automatically added to the collaborative session. (Schümmer and Lukosch, 2007)

  This pattern applies for eDPP, because to start an eDPP session, participants need to come together and choose the artifacts or a project to work on. This can be initiated by one participant inviting another to a project.

- Blind Date: In a big, homogeneous group often not everyone knows everyone else and the expertise, interests, and availability of others. When a task needs a certain number of participants, possibly with specific expertise, it is hard to find the right collaborators. As a solution, this pattern proposes that group members can announce topics in a virtual meeting area, and other users may show their interest in that topic or collaborative task. Once the required number of members gave notice, all those interested can start the session. (Schümmer and Lukosch, 2007)

  For an ongoing eDPP session, this pattern is not relevant, but of course in a community or company this pattern may be a way to find a partner for an eDPP session.

2.1.2.3 Group Support – Textual Communication

Although Schümmer and Lukosch (2007) neglected audio or video communication, this study considers an audio connection to be necessary for a fluent eDPP process. The relevance of video usage was already discussed on page 75. Often, however, textual communication is the means of choice to arrange a collaboration, to share textual information, or for asynchronous communication.

- Embedded Chat: When group members loosely collaborate and from time to time need a simple synchronous communication channel for spontaneous, short questions, information, or appointments, this can be difficult if the members use different tools. To ease such spontaneous communication, this pattern proposes to integrate a chat in the collaboration system that allows to contact a specific user or several group members. (Schümmer and Lukosch, 2007)

  In the eDPP context, such a chat can be useful to arrange a session, or, during a session, to exchange web addresses or other text snippets.

- Forum: A community member wanting to exchange ideas about a specific topic may be challenged in his search of discussion partners when not knowing who else is or will be interested in that topic. As a solution, this pattern proposes to establish a forum as a central communication platform where messages can be asynchronously read and written. (Schümmer and Lukosch, 2007)

  For the intense, synchronous communication, which is part of an ongoing eDPP session, this pattern does not apply.

- Threaded Discussions: The accumulation of all messages in a forum is unstructured, and users have to manually filter information important to them. This makes it hard to understand and follow a specific discussion thread. As a solution, this pattern proposes to reduce the complexity of a forum in form and content by structuring it into discussion threads. (Schümmer and Lukosch, 2007)

  As with Forum, this pattern does not apply for eDPP.

- Flag: When an individual group member contributes information or artifacts, it is hard for the other group members to understand that 'data package' or estimate its importance. As a solution, this pattern proposes to provide flags that allow members to tag their contributions and thus provide guidance for the other group members. (Schümmer and Lukosch, 2007)

  In an eDPP session, the work results are not produced by individual, asynchronous contributions, and accordingly, this pattern does not apply for eDPP.

- Shared Annotation: When a group shares artifacts or information for example in a forum, the individual group members may not just want to passively consume this information. Maybe they want to share their opinion on that artifact, or the author himself wants to annotate it with further information. As a solution, this pattern proposes to provide means to annotate the content or parts of it and display these comments alongside the content data. (Schümmer and Lukosch, 2007)

  The purpose of this pattern is to support the asynchronous exchange of ideas, linked to content data. Accordingly, in its intended usage, this pattern does not apply for the eDPP process. A somewhat alienated version of this pattern for an eDPP session could be that participants annotate certain parts of an artifact with reminders, thoughts, or ideas. These notes may be considered as virtual sticky notes facilitating prospective memory[38] (Tidwell, 2010). Such markers are usually explicitly supported in IDEs by comments that can be tagged with 'ToDo' and which then are automatically transferred into a to-do list.

- Feedback Loop: Due to the ambiguousness of written content, questions arise when group members read text or try to understand content produced by others. There is no other explanatory material, just the content itself. Nevertheless, the author of the content is a valuable resource to clarify questions. As a solution, this pattern proposes to integrate an easy way to contact the author of an artifact next to it. For example, a button opening a contact form with pre-filled information regarding the referenced content would allow a quick and easy feedback to the author. (Schümmer and Lukosch, 2007)

  This pattern aims at a simplified, asynchronous feedback mechanism between content consumers and the content author, and thus does not apply for eDPP.

- Digital Emotions: Textual communication completely lacks paralinguistic elements or even gesture and facial expressions. These elements, however, convey emotional information and greatly facilitate the understanding of irony and humor. Accordingly, their absence can result in misinterpreting funny or ironic remarks. As a result, this pattern proposes to provide means that allow to textually express emotions, for example with emoticons. (Schümmer and Lukosch, 2007)

[38] *"We engage in prospective memory when we plan to do something in the future, and we arrange some way of reminding ourselves to do it. For example, if you need to bring a book to work the next day, you might put it on a table beside the front door the night before. [...] Or if you tend to miss meetings, you might arrange for Outlook or your mobile device to ring an alarm tone five minutes before each meeting."* (Tidwell, 2010)

In principle, when an eDPP tool includes a textual communication channel like a chat, this pattern applies for eDPP. However, in eDPP, the participants are assumed to have an audio connection, and in that view, the possibility of expressing emotions digitally is not considered to have such a relevant impact on the eDPP process. It is thus not further taken into account here.

- FAQ: In big communities which involve established as well as new members, the deep-rooted members are knowledge carriers that can assist newbies and answer their questions. Notwithstanding, answering the same questions over and over again can be annoying and disturbing. As a solution, this pattern proposes to establish a knowledge base of frequently asked questions (FAQ) and to point newbies to that FAQ. (Schümmer and Lukosch, 2007)

  As discussed in the introduction of this section, patterns concerning community maintenance are not in the scope of eDPP.

2.1.2.4 Group Support – Synchronous Group Awareness

Patterns in this cluster deal with how to provide awareness among group members who work on shared artifacts. They explicitly aim at the workspace awareness elements of the workspace awareness framework by Gutwin and Greenberg (2002). (Schümmer and Lukosch, 2007)

- User List: When group members individually or jointly work on shared artifacts, the involved parties should have an idea with whom they are collaborating or who else is accessing the shared artifact. As a solution, this pattern proposes that each group member sees a list of users who also access a shared artifact or attend the collaborative session. This provides workspace awareness about presence and identity, answering the questions who is present in the shared artifact or who is participating in a collaboration. (Schümmer and Lukosch, 2007)

  For an eDPP session, such a User List indicates the partner's general presence in the session and in an artifact, alongside with his identity information.

- Active Neighbors: When group members individually work on shared data, they may not work in the same artifact but in semantically related artifacts, and there may be semantic dependencies between their tasks. In that case, knowing about each other and coordinating one's work can be beneficial and avoid conflicts. As a solution, this patterns proposes that a groupware system does not only show users in the same artifact but tracks activities of other users in semantically related artifacts and makes users mutually aware of such neighboring activities. (Schümmer and Lukosch, 2007)

  For an ongoing eDPP session, this pattern does not apply, since the members are in the same project and are mutually aware of that. Nevertheless, this pattern could be used for example in a company-wide platform to support the semantic-related establishment of eDPP sessions.

- Spontaneous Collaboration: When group members individually work on shared data, they may not become aware of others working on the same artifact. This results in missing collaboration opportunities and potential conflicts. As a solution, this pattern proposes that the groupware system tracks when other members are active in the same area or the same artifact, and visualizes this in the context of the artifacts. Based on this information, the users then can spontaneously start a collaboration. (Schümmer and Lukosch, 2007)

  For project or company-wide Spontaneous Collaboration, that pattern is a specification of Active Neighbors, but only tracks colleagues working in exactly the same artifact. In an ongoing eDPP session, a variant of that pattern that tracks the current position of each participant can be helpful when participants visually decouple. Then the groupware can provide a functionality to easily jump to the other one's position in the workspace and carry on a tight collaboration.

- Interactive User Info: Group members are aware of other members in the shared workspace, for example because they are shown in their User List, but this representation just indicates the users' presence and does not provide any means to interact with them. To communicate or start a collaboration with other members, their contact opportunities like instant messaging or e-mail accounts would have to be figured out manually. As a solution, this pattern proposes to equip the user representation with functionality for direct interaction or setting up a collaboration, for example via an associated context menu or other interface elements. (Schümmer and Lukosch, 2007)

  In the context of eDPP tools, this pattern can be used to start a collaboration in an easy way. During a running session it can be used to show more information about the user and his activity alongside with his ID, for example what artifact he has focused on. Moreover, an Interactive User Info can provide options to start Shared Browsing or, when the other one resides in a different artifact, a Spontaneous Collaboration by jumping to his current position.

- Remote Field of Vision: Variants of that pattern are Remote Scrollbars or Remote Viewport. When users work in the same artifact using a shared editor, they can scroll independently, so that their viewports are not necessarily the same. This increases the flexibility by relaxing the coupling of work, but makes a shared focus more difficult. As a solution, this pattern proposes to indicate the artifact area seen by the remote team members. For graphical artifacts, this may be an inset box, for textual artifacts, it may be an additional, inoperable scrollbar indicating their viewport. (Schümmer and Lukosch, 2007)

  eDPP is realized via a shared editor and the participants can scroll their viewports independently. Accordingly, this pattern, in particular the Remote Scrollbars variant, is relevant for eDPP tools.

- Remote Selection: When group members collaboratively work in a shared editor, both can select or highlight artifacts or parts of it, which often indicates a planned action on that selected item. Another typical user behavior is to select the object that is subject of the current train of thought or of the current discussion. (Schümmer and Lukosch, 2007)

  Such a selection may be considered a virtual forefinger pointing at the conversational or current work context, simplifying verbal communication in synchronous interaction (see page 72 for the relevance of local and object deixis in collaborative work). Accordingly, this pattern applies for eDPP tools.

- Remote Cursor: This pattern is also referred to as Remote Target Indicator, Telecursor, Remote Mouse Pointer, or Direct Tele-pointer. When users work tightly coupled and simultaneously manipulate the same artifact, they want to know what part of the artifact the others are concerned with and want to follow their actions as comprehensively as possible. As a solution, this pattern proposes to show the mouse or text cursor of the other participants in the jointly edited artifact and make these remote cursors distinct from the local cursor. (Schümmer and Lukosch, 2007)

  As discussed in Section 1.4.4.3 "Part Three: How Workspace Awareness is Used", knowing what the others are doing is essential for an efficient and smooth collaboration. A Remote Cursor is less expressive than body language, but is a virtual tool that not only shows the current focus of the others but also facilitates gestural communication and thus eases verbal communication (see page 72). Beyond that, Remote Cursor is a form of user embodiment in the workspace – it represents a user in the workspace, shows where he is working and what he is doing (Dyck et al., 2004). For the fine-grained, synchronous collaboration in eDPP, a Remote Cursor is very useful because the *"unintentional communication of the fine-grained details of activity is what enables smooth, error-free, and natural real-time interactions in shared workspaces"* (Dyck et al., 2004).

- Remote Caret: This a variant of the Remote Cursor, but refers to the text cursor. This means that the position of the other participant's caret is shown when being in the same artifact.

The remote caret and the cursor should be distinguishable from the personal cursor.

- TELEPOINTER: When group members synchronously collaborate in a shared artifact, at certain points in time, they may want to focus the attention of the others to a specific part of the artifact, for example to visually indicate what their verbal remarks refer to. This is particularly relevant in visual artifacts, where positions cannot be referenced by line numbers or words but have to be described otherwise. As a solution, this pattern proposes to offer a virtual forefinger for each user that can be activated when needed. An activated pointer is annotated with the activator's identity and transmitted to all other participants, signaling the demand of their attention. (Schümmer and Lukosch, 2007)

  Similar to a REMOTE CURSOR, a TELEPOINTER reveals rich awareness information, but only for the time it is activated. In general this pattern applies for eDPP. It can be seen as a need-based REMOTE MOUSE POINTER – when not used, it does not clutter the interface, because in eDPP, the predominant activity is text editing, which in this case is reflected by the REMOTE CARET.

- ACTIVITY INDICATOR: When group members synchronously collaborate in a shared workspace, no matter whether being in the same or in different artifacts, they want to have a general idea of what the others are doing. For some tasks, however, time is needed to carry out individual activities, and results may only become visible to others later. Such a period of invisible activity may result in conflicts, delays, or irritations that feel as if there is no progress whatsoever. As a solution, this pattern proposes to provide an unobtrusive interface area, abstractly indicating the other's actions without showing intermediate results. (Schümmer and Lukosch, 2007)

  For eDPP, this pattern can offer a general sense for activities other than editing, like for example running test cases, using unshared dialogs when doing a refactoring operation, and alike.

2.1.3 Consolidating Relevant Awareness Requirements and CMI Patterns for eDPP

The following list summarizes those awareness elements of the workspace awareness framework which are relevant for eDPP:

- WHO-Presence: (Prior to a session) Who is available to start a collaboration?
- WHO-Presence: Is my collaboration partner present in the shared workspace?
- WHO-Identity: Who is my collaboration partner?
- WHO-Authorship: Which changes were made by my remote partner?
- WHAT-Artifact: What artifact (section) is my partner working on? Either on a detailed level of code lines when having the same viewport, or on a more general level like artifact sections or project files when having non-overlapping viewports.
- WHAT-Action: What is my remote partner actually doing? Either on the level of detailed editing activities, or on a level of general operations like searching, testing, etc.
- WHERE-Gaze: Which area of the shared artifact can my remote partner see?
- WHERE-View: Where is my remote partner looking?

Some awareness information imply other awareness elements. For example, seeing the editing activities of the other one implicates knowledge about his presence in the artifact, what artifact section he is working on, what he is doing, and where he is looking.

Figure 2.2 correlates the eDPP-relevant CMI patterns with the workspace awareness information they provide. In combination with other information like the main purpose of a tool, its usage context and target user groups, this is a sound base to compare concrete eDPP tools.

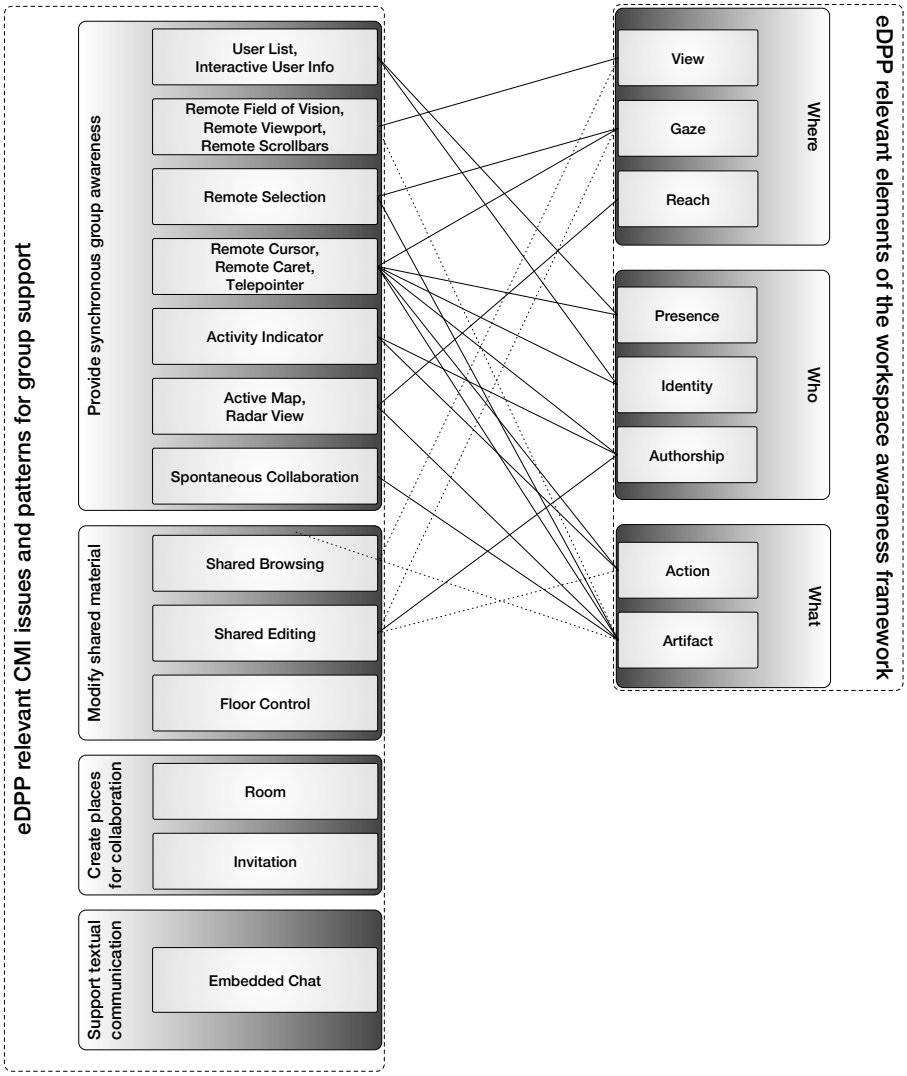

Figure 2.2: eDPP relevant CMI patterns and the awareness information they provide

eDPP relevant patterns of computer-mediated interaction by Schümmer and Lukosch (2007) are connected to the awareness information they provide according to Gutwin and Greenberg (2002). Awareness information that can be derived through a provided functionality is interlinked with a dotted line.

2.2 eDPP Tools

This section gives an overview of eDPP tool alternatives that could have been used for this inquiry.

As presented in previous sections, a tool for eDPP needs to provide the following fundamental functionality besides fulfilling the awareness requirements:

- It must support synchronous editing activities of two or more participants.
- It must provide relaxed WYSIWIS (see Section 1.4.3.1 "Group interfaces"), so that participants can separately explore their workspaces and scroll in the jointly viewed artifacts without disrupting their partner.
- Participants should be able to equally contribute to the artifacts while experiencing real-time write speed.
- To facilitate the unimpeded, synchronous editing activities of all participants, social floor control (see Section 1.4.3.1 "System requirements due to (distributed) group setting") is necessary, which in turn requires mechanisms for data consistency as well as conflict detection.
- Establishing or joining a collaboration should be straightforward, requiring as little extra work steps as possible (see page 65).
- The developers may work in their familiar environment and do not have to switch systems for individual work or remote collaboration (see Section 1.4.3.2 "Groupware Design: Real-Time Groupware").

This overview sums up the spectrum of eDPP tools that fulfill these essential requirements. Finally, the tools that qualify as mature and realistic alternatives for eDPP are compared in a feature matrix on page 104.

Tools for eDPP can be classified in five major categories according to their maturity and technical functional principle:

1. Inactive projects and research prototypes
2. Collaborative web coding pads
3. Collaborative editors
4. Collaborative cloud-based web IDEs
5. Collaboration plugins for native IDEs

The concrete products of each category are discussed in the following paragraphs.

Inactive projects and research prototypes An extensive number of tools have been developed for research purposes. Some never reached industrial maturity, others have become inactive by now. Due to their immatureness and partly crude functionality, these tools are no realistic alternatives for eDPP. Examples of these are listed in descending order according to their actuality – starting with the most recent and ending with tools that are not available any more or where information on their up-to-dateness is missing.

- *Gobby*[39]: An open source collaborative editor with an integrated group chat. Its current release on GitHub[40] is 0.5.0. dating from August 2014.

[39] `https://gobby.github.io/` (accessed February 11, 2017)
[40] `https://github.com/gobby/gobby` (accessed February 11, 2017)

- *Collabode*[41] (collab + code): A web-based IDE for synchronous collaboration of developers, technically based on *Eclipse* and Etherpad and developed for research purposes (Goldman et al., 2011). Its latest version on GitHub[42] is 1.3.0 dating from 2012.
- *Rudel*[43] (German for 'pack' as in 'pack of wolves'): An extension for the GNU *Emacs* editor, enhancing it with collaborative functionality. Its latest version on SourceForge[44] is 0.2-4 dating from 2010.
- *DocShare*[45]: *Eclipse* Plugin for sharing and collaboratively editing individual files. There is no official release on GitHub and the latest project contribution was in August 2010.
- *COLLECE* (COLLaborative Edition, Compilation and Execution): A groupware for synchronous distributed pair programming. Apart from the publication of Duque and Bravo (2008), no further activity with regard to the the development of COLLECE could be found.
- *XPairtise*: An *Eclipse* Plugin for distributed pair programming, developed in a university context (Kröger, 2008). The last version is 1.1.0 dating from 2008 on SourceForge[46]. There is no evidence that XPairtise has reached industrial maturity or is widely used. The same applies for XecliP[47], a related DPP tool *"developed in competition in another sub-team of our [the XPairtise] research group"* (Lukosch and Schümmer, 2007). Its latest version is 1.0.0 dating from 2007.
- *PEP* (Pair *Eclipse* Programming): An *Eclipse* Plugin for distributed pair programming that does not require a server to mediate data exchange between the participants. Its current version on SourceForge[48] is 0.0.3 dating from 2007.
- *ACE*: A platform-independent editor for collaborative viewing and editing of text documents. Its latest version on SourceForge[49] is 0.0.3 dating from 2006.
- *Sangam*: An open source *Eclipse* plugin for distributed pair programming. Sangam *"provides functionality for the developers to edit source code synchronously. The Sangam Launchers enable the developers to launch a Java application or JUnit test together"* (Ho et al., 2004). The last stated mission on the Sangam project web site[50] is *"Release 1 : An end-to-end plugin to be rolled out on May 12, 2004. The plugin will allow shared editing, saving, and running of programs."* The latest available version of Sangam is 1.2.0 and there is no evidence that Sangam has reached industrial maturity or is widely used.
- *TUKAN*: A programming environment for distributed teams, developed by one of the authors of the CMI pattern language. Till Schümmer[51] developed it in the context of his diploma thesis in 1999 (Kröger, 2008). It provides *"awareness information, communication channels, and synchronous collaboration mechanisms"* (Schümmer and Schümmer, 2001), but *"TUKAN was built as an extension of a relatively unpopular development environment, namely ENVY for VisualWorks Smalltalk. This is one of the reasons why it has not gained high popularity"* (Schümmer and Lukosch, 2009). TUKAN is not publicly available.
- *CloudStudio*[52]: A collaborative web IDE developed as a research project to test ideas with regard to the simplification of distributed software development. The usage of CloudStudio is limited to test it online[53], but the server is not available (latest attempt to access on October 28, 2017).

[41] `http://groups.csail.mit.edu/uid/collabode/` (accessed February 11, 2017)
[42] `https://github.com/uid/collabode` (accessed February 11, 2017)
[43] `http://www.emacswiki.org/emacs/Rudel` (accessed February 11, 2017)
[44] `http://sourceforge.net/projects/rudel/` (accessed February 11, 2017)
[45] `https://github.com/sschmidt/org.eclipse.ecf.docshare` (accessed February 11, 2017)
[46] `http://sourceforge.net/projects/xpairtise/` (accessed February 11, 2017)
[47] `http://sourceforge.net/projects/xeclip/` (accessed February 11, 2017)
[48] `https://sourceforge.net/projects/pep-pp/` (accessed February 11, 2017)
[49] `http://sourceforge.net/projects/ace/` (accessed February 11, 2017)
[50] `http://sangam.sourceforge.net/` (accessed February 11, 2017)
[51] `https://www.fernuni-hagen.de/ks/en/team/till.schuemmer.shtml` (accessed February 11, 2017)
[52] `http://se.inf.ethz.ch/research/cloudstudio/` (accessed February 11, 2017)
[53] `http://cloudstudio.ethz.ch/`

- *RIPPLE* (Remote Interactive Pair Programming and Learning Environment): An open source plugin for the *Eclipse* IDE, extending the architecture of Sangam. RIPPLE has a built-in data collection functionality and *"was designed for use in educational settings to facilitate various forms of collaborative programming problem solving including distributed pair programming and distributed one-on-one tutoring"* (Boyer et al., 2008). Unfortunately, RIPPLE is not publicly available.
- *Moomba* (*"Australian aboriginal word meaning 'let us get together and have fun"'* (Reeves and Zhu, 2004)): An environment for distributed software development that extends the awareness information provided by TUKAN and provides additional functionality like a project coordination component as well as a *"Pair-Programming Partner Finder"* (Reeves and Zhu, 2004). Moomba is not publicly available.
- *COPPER* (COllaborative Pair Programming EditoR): A *"synchronous source code editor that allows two distributed software engineers to write a program using pair programming. COPPER implements characteristics of groupware systems such as communication mechanisms, collaboration awareness, concurrency control, and a radar view of the documents, among others"* (Natsu et al., 2003). Apart from the publication of Natsu et al. (2003), no further activity with regard to the development of COPPER could be found. COPPER is not publicly available.

Collaborative web coding pads Collaborative web coding pads are text editors running in the web browser. They are very simple with regard to functionality. They support the collaboration on single files where the code must be uploaded or pasted in the text editor. The shared document is accessed via shared URLs. They provide a simple User List and transmit editing activities in real-time. They do not designate these changes to the respective author (for example by highlighting them in a color assigned to the specific user). They also do not provide advanced awareness functionality like Remote Selection or Remote Field of Vision. Their main applications are sharing code snippets, teaching or learning from peers, doing short collaborative programming tasks, or coding tests during remote job interviews of developers. They usually support syntax highlighting for programming languages and have integrated text, voice, or video chat. Some of them include a playback feature, allowing to record and playback the editing activities of a pad. They are web-based and developed for practical purposes, this is why the following compilation contains web addresses (URLs) but no literature reference.

- *Codassium – "A better way to conduct remote interviews"*[54]: A commercial online service to support remote job interviews with developers. It provides a collaborative editor for single files which allows code execution. It includes video chat, screen sharing, and a collaborative Linux terminal. In addition to that, it facilitates note taking and interview scheduling.
- *Codeshare – "Share code in real-time with other developers"*[55]: A free to use online collaborative editor with integrated video chat. Possible applications are PP, job interviews, and code reviews. To save the code from the editor, a login is required.
- *Codebunk – "The Ultimate Online Coding Interview Tool"*[56]: A commercial online service to support remote job interviews with developers. It provides a collaborative web editor with code compiling and video and text chat. Interviews can be recorded and stored.
- *CoderPad.io – "Interview your candidates in an intuitive live programming environment"*[57]: A commercial online service to support remote job interviews with developers. It includes a collaborative web editor with code execution, record and playback functionality, and a question library (for standardizing/collecting questions for interview candidates).
- *Etherpad – "Collaborating in really real-time"*[58]: Open source web editor including a simple user

[54] `http://codassium.com/` (accessed July 26, 2017)
[55] `http://codeshare.io/` (accessed July 26, 2017)
[56] `http://codebunk.com/` (accessed July 26, 2017)
[57] `https://coderpad.io/` (accessed February 12, 2017)
[58] `http://etherpad.org`,`https://github.com/ether/etherpad-lite` (accessed July 26, 2017)

list and highlighting the edits of users in their assigned color on line level. Its free to use online or can be downloaded to run it on another web server.

- *Socrates.io – "Write and read Markdown in real-time with anyone you want"*[59]: Free and simple online editor for collaboratively writing Markdown[60]. There has never been an official release and the last update was in March 2013.
- *Kobra.io – "Simple code collaboration"*[61]: Commercial collaborative online editor. Single files can be edited online and for free. Private files as well as the built-in video and text chat are available with a payed subscription. *Kobra.io* provides a user-annotated Remote Caret and Remote Selection.
- *Collabedit – "simple collaborative text"*[62]: A free to use collaborative web editor. It provides syntax highlighting and includes text chat and document versioning. *Collabedit* is still available but currently does not support new registrations since it has been replaced by *Coding Hire*[63], a commercial tool for remote interviews

Collaborative Editors A collaborative editor is a text editor allowing the collaborative real-time editing of source code files. Compared to web coding pads it provides more functionality for text editing, allows to share multiple files or whole projects and provides awareness features like for example Remote Selection, Remote Cursor, or a User List. A collaborative editor can be web-based, a plugin that equips a conventional single-user text editor with collaboration features, or a dedicated collaboration tool. Web-based collaborative editors either access the user's local file system or integrate cloud services for file storage. Collaboration is usually established via shared URLs.

- *SubEthaEdit – "Collaborative text editing. Share and Enjoy"*[64] (the name is a composition of the prefixes of Subway – Ether – Editor, referring to the Sub-Etha communication network from the science fiction novel 'The Hitchhiker's Guide to the Galaxy'[65]): A mature and commercial native text editor for the Apple operating system. *SubEthaEdit* is a very mature product with a simple interface (it won the Apple Design Award in 2003). The editor can be used for individual and for collaborative text editing as well as for Shared Browsing. Its current version is SubEthaEdit 4.1.
- *Firepad – "Open source collaborative code and text editing"*[66]: *Firepad* is a collaborative text editing pad. It is realized as an open source JavaScript library and can be embedded in arbitrary web sites and web applications. It adds collaboration features to *CodeMirror*[67] or *Ace*[68], both powerful web editors implemented in JavaScript. Among others, *Firepad* is used in *Socrates.io*, *CoderPad.io*, and *Kobra.io*.
- *Floobits – "Cross-editor real-time collaboration"*[69]: Commercial solution that provides centralized workspaces in which artifacts can be collaboratively edited from a web editor. Free plugins are available that allow to access and collaboratively edit the web workspace files from editors (*Sublime text*[70], *Emacs*[71], *Neovim*[72], or *Atom*[73]), or from the *IntelliJ IDEA*[74] IDE. The web

[59] `http://socrates.io/`,`https://github.com/segmentio/socrates` (accessed October 28, 2017)
[60] Markdown is a simple markup language that can be automatically converted to HTML
[61] `https://kobra.io/` (accessed July 26, 2017)
[62] `http://collabedit.com/` (accessed July 26, 2017)
[63] urlhttps://codinghire.com (accessed July 26, 2017)
[64] `http://www.codingmonkeys.de/subethaedit/` (accessed February 12, 2017)
[65] Derivation of the name found on `https://de.wikipedia.org/wiki/Subethaedit` (accessed February 12, 2017)
[66] `https://firepad.io/` (accessed February 12, 2017)
[67] `http://codemirror.net/` (accessed February 12, 2017)
[68] `https://ace.c9.io/` (accessed February 12, 2017)
[69] `https://floobits.com/` (accessed February 12, 2017)
[70] `https://www.sublimetext.com/` (accessed February 12, 2017)
[71] `https://www.gnu.org/software/emacs/` (accessed February 12, 2017)
[72] `https://neovim.io/` (accessed February 12, 2017)
[73] `https://atom.io/` (accessed February 12, 2017)
[74] `https://www.jetbrains.com/idea/` (accessed February 12, 2017)

editor also allows to share terminals, supports WebRTC for video chat and screen sharing, or can be integrated into *Google Hangouts* to use its screen sharing and video chat.

Collaborative cloud-based web IDEs The latest trend, in particular for web development, are cloud IDEs. These are cloud-based web applications simulating a native IDE. Compared to text editors, an IDE has a broader function scope. Among others, they support programming activities like compiling, debugging, running code and tests. They usually have an integrated version control and terminal. One of their major advantages is that they comfortably support program understanding: they provide semantic search and navigation features like showing a type's hierarchy, the call chain of a method, or finding the declaration of a method.

As with collaborative web editors, cloud-based web IDEs support collaborative work on shared workspaces and use cloud services for data storage. They can be considered as a 'workspace as a service' because they provide holistic development environments that free the developer of setting up infrastructure like version control, databases, configuration management, etc. For web development projects they provide convenient live application preview and testing capabilities. The different products vary greatly with regard to provided awareness features.

- *Cloud9 – "Your development environment, in the cloud"*[75]: A commercial cloud workspace including a collaborative online IDE and a Linux workspace. The IDE includes a simple User List allowing to specify the read and write access for the other session members. Further features are Remote Caret, Remote Selection and Embedded Chat. The editing contributions of the individual users are annotated in their assigned color on line level. A Replay function allows to revisit a file history. Moreover, the IDE provides a cloned terminal. Participants can be invited via sharing a workspace URL.
- *Codeanywhere – "A complete toolset for web development. Enabling you to edit, collaborate and run your projects on any device"*[76]: Commercial cloud IDE, providing a Remote Caret and Remote Selection. When the participants are in the same artifact, a basic User List with Interactive User Info allows Spontaneous Collaboration by jumping to the position of the user's cursor.
- *Codebox – "Code with the same editor on your desktop and in the cloud"*[77]: An open source cloud and desktop IDE. Its development started in 2014 and its current version on GitHub[78] is 1.0.0-alpha.5 dating from April 2015. The *Codebox* webpage has not been accessible for several months, the last unsuccessful attempt being undertaken in October 2017. However, screenshots showed that it implements a Remote Caret.

Collaboration Plugins for native IDEs Collaboration plugins enhance single-application desktop IDEs with real-time collaboration features. From within their local IDE, the developers share projects which they can concurrently edit while the plugin syncs their concurrent editing activities. The collaboration is realized via a session, to which one participant can invite others.

- *VSAnywhere*[79]: An extension adding real-time collaboration features to Microsoft Visual Studio[80] and thus allowing distributed synchronous development on .NET projects. Unfortunately, the *VSAnywhere* website[81] has not been available for a long time (last attempt to access in October 2017) and there is no evidence that *VSAnywhere* is still active or available. With the help of online product videos and screenshots, the following features of *VSAnywhere* could be identified:

[75] `https://c9.io/` (accessed July 26, 2017)
[76] `https://codeanywhere.com/` (accessed July 26, 2017)
[77] `https://www.codebox.io/` (accessed October 12, 2017)
[78] `https://github.com/CodeboxIDE/codebox` (accessed July 26, 2017)
[79] `https://vs.componentsource.com/product/vs-anywhere-0` (accessed February 12, 2017)
[80] `https://www.visualstudio.com/` (accessed February 12, 2017)
[81] `http://vsanywhere.com`

Shared Editing, Embedded Chat, User List showing the user's focused file, a Remote Caret as well as scroll markers indicating the partners' scroll position in the artifact. Users may choose whether they want the changes of the remote partner to be highlighted in their workspace or not. Moreover, *VSAnywhere* allows unlimited session participants and Shared Browsing.

- *Floobits* in *IntelliJ IDEA*[82]: An open source plugin started in 2014 which allows to connect the *IntelliJ IDEA* IDE with a *Floobits* workspace, and in that way integrating Shared Editing in the *IntelliJ IDEA* IDE. It has a User List with Interactive User Info, a Remote Caret, and the others' editing activities are highlighted (these highlights can be cleared). It supports Remote Selection and Shared Browsing. Users can summon other users to their cursor position (variant of Spontaneous Collaboration). When the participants are in different artifacts or have non-overlapping viewports, there are no cues about the other one's location. Its current release on GitHub[83] is 1.6.6 dating from December 2015.

- *Saros*[84]: An open source plugin for real-time collaboration for the *Eclipse* IDE as well as, currently under development, for the *IntelliJ IDEA* IDE. *Saros* supports Shared Editing with up to five participants in a collaborative session. It has a User List with Interactive User Info, showing for each participant the currently focused file (variant of Active Neighbors) and allowing to jump to that participant's cursor position (variant of Spontaneous Collaboration). It also includes an Embedded Chat, a Remote Caret and allows Remote Selection and Shared Browsing. The last 20 editing activities of the remote participants are highlighted in their assigned color and, for jointly viewed artifacts, a Remote Scrollbar indicates the other one's viewport. The project's file outline is enhanced with user-colored indicators about who views which artifact, realizing a kind of Active Map. Furthermore, *Saros* provides a simple shared whiteboard.

With this overview of eDPP solutions in mind, the question arises which of them constitute realistic alternatives for professional distributed software development. Immature research prototypes and inactive projects cannot be entertained. Due to their rudimentary functionality, collaborative web coding pads are also unsuited for professional contexts that exceed the short discussion of code snippets. In general, whether using an IDE or a text editor is a matter of taste among developers. In the context of eDPP, the collaborative text editors fulfill the minimal technical requirements but do not provide further awareness information. A considerable amount of cloud IDEs has been developed in the last years. They are particularly useful for web development, but require to transfer the data and development environment in the cloud and to use the services of the commercial providers. Their awareness features are less extensive than those provided by the collaboration plugins for single-user IDEs. *Cloud9* and *Codeanywhere* are the leaders within this field and are very close to collaboration-enhanced single-user IDEs.

Single-user IDEs upgraded with collaboration functionality provide true real-time write speed for local changes since they only have to transfer remote editing activities. They *"provide the full functionality that developers know from single-user development environments"* (Schümmer and Lukosch, 2009). Thus, they allow a smooth transition between individual and collaborative work without requiring tool switches alongside with data transfer and environment configuration. And the developers can work on their own computer in their familiar environment with their own keyboard shortcuts, and are not slowed by unfamiliar environments which rather make them feel uncomfortable (Plonka et al., 2012a).

Finally, in addition to the leading web IDEs (with their specific usage conditions), the two active collaboration-upgraded single-user IDEs *Floobits* and *Saros* de facto remain as realistic eDPP tools. The following table 2.1 compares their features and the awareness elements they support.

[82] `https://floobits.com/help/plugins/intellij` (accessed February 12, 2017)
[83] `https://github.com/Floobits/floobits-intellij` (accessed July 27, 2017)
[84] `http://www.saros-project.org/` (accessed February 12, 2017

Table 2.1: Feature matrix of eDPP tools

Saros	Floobits and *IntelliJ IDEA*	Codeanywhere	Cloud9	eDPP Tool	
✓	✓	✓	✓	Room (Workspace)	
✓	✓	✓	✓	Invitation	
✓	✓	✓	✓	Embedded Chat	
✓	✓	-	-	Shared Browsing	
✓	-	✓	-	Spontaneous Collaboration	
-	-	-	✓	Shared Terminal	
✓	-	-	-	Shared Whiteboard	
-	-	-	✓	Replay	
✓	✓	✓	✓	Shared Editing	WHO – Authorship
✓	✓	-	✓	Colored Remote Editing	
✓	✓	✓	✓	User List	WHO – Presence and Identity
✓	✓	✓	-	Interactive User Info	
-	-	-	-	Remote Field of Vision	WHERE – Viewport
✓	-	-	-	Remote Scrollbars	
✓	✓	-	✓	Remote Selection	WHERE – Gaze
-	-	-	-	Telepointer	
✓	✓	✓	✓	Remote Caret	WHAT – Action on detailed level
-	-	-	-	Remote Cursor	
-	-	-	-	Activity Indicator	WHAT Action on general level
✓	-	-	-	Active Map	WHAT Artifact (workspace overview)
-	-	✓	-	Radar View	WHAT Artifact section (artifact overview)

To study the actual eDPP process and not the maturity of a tool, a mature, unobtrusive eDPP tool that implements current research findings as far as possible was needed. In 2006, when the idea of examining eDPP was born, no passable and usable eDPP tool was available. Cloud-based eDPP solutions were not available, either, and still today they are not that widely used as their desktop counterparts. This is why the development of *Saros* started in 2006. **Even in 2010, when the current research project was initiated, *Saros* was far away from being a mature, practically applicable solution**. It was the most promising solution for eDPP in those days. Nevertheless, to make *Saros* a mature, usable, state-of-the-art tool for eDPP that can be used as a testbed for this inquiry, a lot of work had to be done. As a consequence, a significant amount of time and energy of this PhD project was spent on that.

After some years of intense work on user experience and stability, *Saros* has become a robust and usable tool. Table 2.1 shows that, compared to other realistic eDPP solutions, it has the same basic functionality, and implements a superset of workspace awareness features – finally, it can be considered a state-of-the-art tool for eDPP.

The successful work on *Saros* is nicely reflected by the statement of the DPP practitioner Joe Kutner, who in 2013 presented *Saros* as the best IDE-based eDPP tool: *"Saros is the most robust collaborative editing environment we could ask for"* (Kutner, 2013).

The following section will describe *Saros* in more detail and gives a summary of its development history.

2.3 Saros

Saros is an open source plugin for the *Eclipse* IDE that enables real-time distributed software development with up to five participants. Its first years of development have been an arduous journey as presented in Section 2.3.1 "Saros History". This section gives an overview of *Saros*' development history, its challenges and stages of development from the technical, functional and usability perspective. Moreover, it illustrates the challenge of attracting real or professional users, which has been an absolute precondition for this research project and turned out to be far more difficult than expected.

As an *Eclipse* plugin, *Saros* hooks into different areas of the *Eclipse* workbench: It has a separate view, providing a menu bar and containing the User List, the Embedded Chat, and the shared whiteboard. During a session, it annotates the editor as well as the project explorer with awareness information, apart from synchronizing the editing activities in real-time.

To explain the awareness features of *Saros*, in a first step, Figure 2.3 shows a wireframe of the *Saros* interface. It is explained with the help of the protagonists Alice and Bob who are already known from Section 1.2 "Approaches to Distributed Pair Programming (DPP)". Figure 2.3 sketches the situation that Alice invited Bob to a *Saros* session; it is her workspace which is to be seen.

The interface elements of *Saros* are listed in the following. They are named according to the pattern language of Schümmer and Lukosch (2007) with capitalized pattern names:

- Shared Editor: *Saros* extends the *Eclipse* editor to be a distributed editor. It transmits all editing activities of Alice in real-time to Bob and vice versa.
- Floor Control: *Saros* realizes relaxed WYSIWIS with social floor control (see Section 1.4.3.1 "System requirements due to (distributed) group setting"). This means Alice and Bob can concurrently view or edit the same or different shared artifacts – *Saros* does not impose explicit roles or editing right restrictions.
- User List with Interactive User Info: Bob's name is shown alongside his assigned color in Alice's user list. It shows the file Bob focuses on and whether he follows Alice (whether master-slave Shared Browsing is enabled, see page 90). For a kind of Spontaneous Collaboration, the Interactive User Info allows Alice to start following Bob or jump to

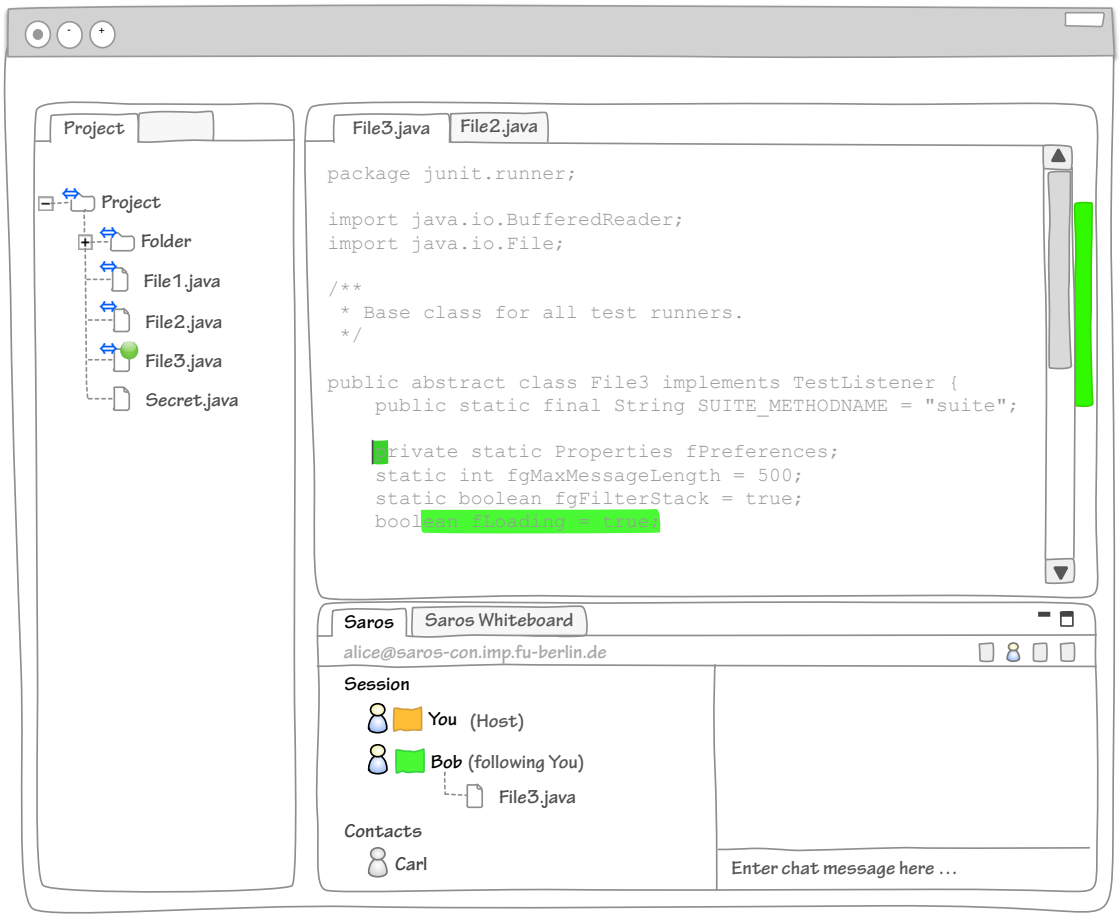

Figure 2.3: Schematic view of the *Saros* interface

The schematic view is used to facilitate matters with regard to illustrating the features and awareness annotations of *Saros*

his position in the workspace. Via the Interactive User Info Alice or Bob can invite other online users to a new session or add them to a running session.

- Embedded Chat: Alice can chat one-to-one with Bob during or prior to a session. When there are more participants, they can use the multi-user chat that is automatically started for all participants in an active session.
- Shared Whiteboard: When Alice and Bob want to visually exchange ideas, they can use the shared whiteboard. It allows simple drawings that can be exported and saved as images.
- Remote Caret: When Alice and Bob work in the same artifact, Alice can follow the position of Bob's cursor in her workspace which is highlighted in his assigned color.
- Colored Editing: In Alice's workspace, the 20 most recent editing activities of Bob are highlighted in his assigned color.
- Remote Selection: If Bob selects a code snippet, it is also highlighted in Alice's view in Bob's markup color (not shown in Figure 2.3).
- Remote Scrollbar: A scrollbar-like annotation next to Alice's scrollbar indicates the artifact section currently seen by Bob.
- Active Map: The file focused by Bob is annotated with a dot in his assigned color in the file overview tree, the so-called *Eclipse* project outline view. All files of the project that are shared in a session are annotated with a blue double-arrow.

Other *Eclipse* views, such as the artifact outline or dialogs and wizards, are not shared. For the pure manipulation of shared artifacts, though, experience has shown that after learning to pay attention to the available awareness indicators, they provide good information about where the partner is working and what he is doing.

To get an impression of *Saros* in real operation, Figure 2.4 shows a screenshot of *Saros*.

Technically, *Saros* works as follows:

- Collaborative Session: The collaboration context of Alice and Bob is conceptually realized as a session.
- Invitation: When Alice wants to collaborate with Bob, she can send him an invitation. When Bob accepts the invitation, they both are in a Collaborative Session. Alice, as the initiator of the session, is the session's host[85]. When Alice closes the session, the session ends for all participants. Only she then has the privilege to assign or strike off editing rights for Bob and all other potential session participants.
- Replicated Objects: When Bob accepts the invitation of Alice, he can choose whether *Saros* should transmit a local copy of Alice's project files or whether his existing local copies should be synchronized with that of Alice, which would mean that his local changes would be overwritten. In any case, the bottom line is that both work on a consistent local copy of the shared artifacts.
- Decentralized Updates: To keep the local copies of Alice and Bob in sync, *Saros* updates the local copy of Alice with Bob's changes and vice versa. This is done via sending XMPP-messages[86] containing the update information (local changes) to all other participants in a session.
- Operational Transformation: To ensure consistency and thus to control the concurrent changes of Alice and Bob, *Saros* uses the *Jupiter* Algorithm of the *ACE* project[87]. In a simplified

[85] For a comic explaining the *Saros* host role, see `http://www.saros-project.org/host-comic` (accessed November 11, 2017)

[86] *"XMPP is the Extensible Messaging and Presence Protocol, a set of open technologies for instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data."* `http://xmpp.org/xmpp-protocols/xmpp-extensions/` (accessed November 11, 2017)

[87] `http://sourceforge.net/projects/ace/` (accessed October 29, 2017)

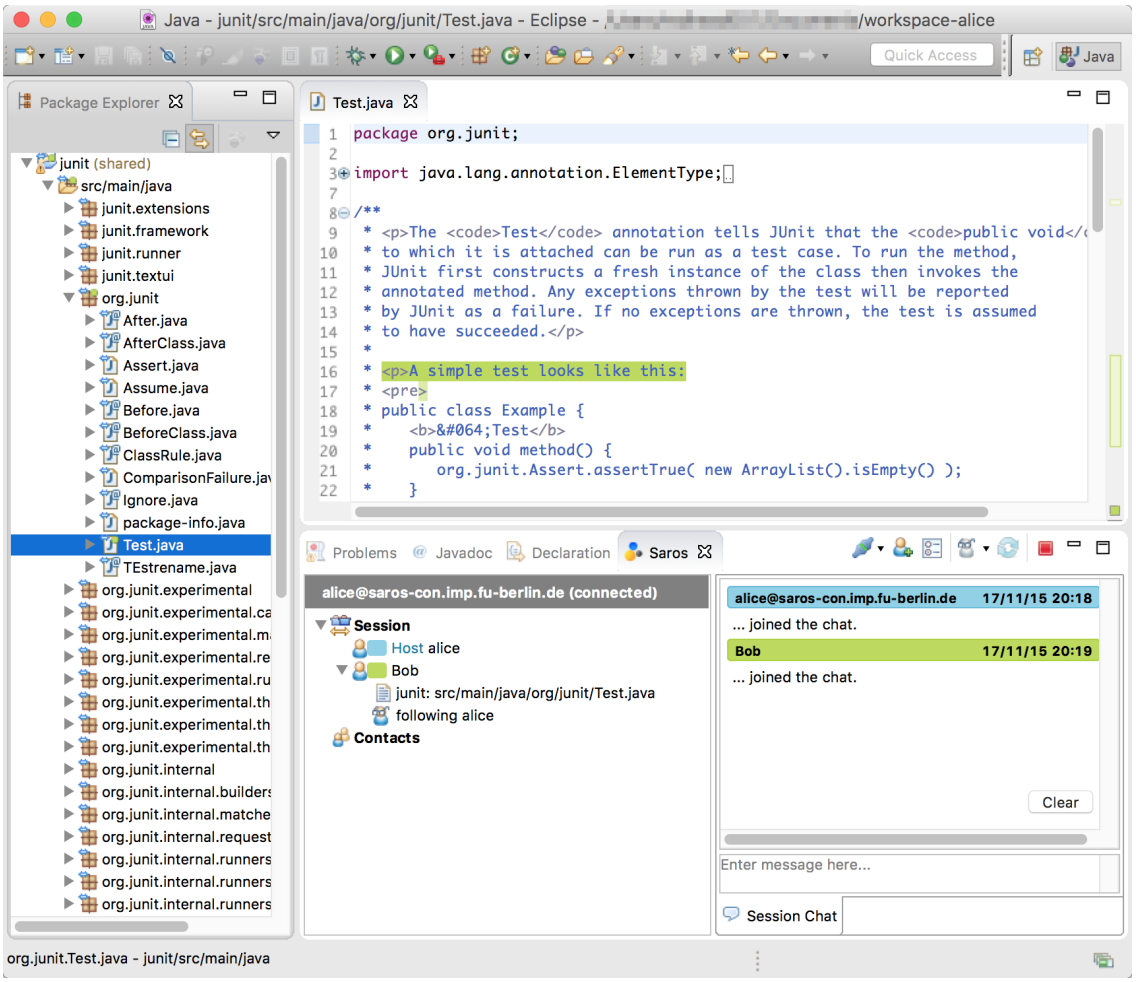

Figure 2.4: Screenshot of *Saros* (version 13.12.6)

scheme[88], with the Jupiter algorithm, local as well as remote operations (changes in the artifact) get a timestamp, so that they can be transformed in their correct chronological order before being applied locally.

- Conflict Detection: Every ten seconds *Saros* generates and compares checksums for all opened files in a session. If these checksums differ, the files are assumed to be in an inconsistent state. If an inconsistency is detected, Alice and Bob are informed and can choose to recover from this inconsistent state. Since Alice is the host of the session, the state of her file is transferred to Bob.

2.3.1 Saros History

This section outlines the history of *Saros* and illustrates the course of the development focus since its birth. As will be seen, the development of *Saros* was not straightforward, but after more than ten years of active research and development, *Saros* has finally reached industrial maturity.

The first version of *Saros* was developed by R. Djemili (Djemili, 2006) and provided the basic functionality for realizing distributed pair programming with a strict floor control according to the roles of driver and observer. The following years, *Saros*' feature scope was extended. After reaching a saturation of relevant eDPP features but while still having problems in finding users, the development rather concentrated on stability and maintainability issues (involving architectural consolidations) as well as usability aspects.

Table 2.2 provides an overview of all student theses that contributed to *Saros* – by July 2017, 60 theses had been submitted. For each thesis, the table briefly outlines the objective addressed, the main steps that had to be taken in that context, and the result. The table's first row shows the month and year of submission for each thesis. It also classifies each thesis with regard to the addressed product aspect and encodes it according to the following scheme:

[Category]-T[XX](NO)

- Category: The theses are categorized according to the following product aspects they focused on:
  - **Functionality (Fct)**: Theses in this category were concerned with adding features or functionality to *Saros*.
  - **Quality improvement (Qua)**: These theses aimed to improve the product quality. This involved product-related improvements like increasing the test coverage, fixing defects, refactoring modules, or betterments of the general architecture. It also involved constructive actions on project and process level like introducing test frameworks, optimizing processes, or work out and provide information to ease the settling-in for new developers.
  - **Usability and user experience (UX)**: This category comprises theses that explicitly aimed at improving *Saros*' usability and user experience. Usually they focused on a specific aspect or area and followed the user-centric design approach.
  - **Outreach (Outr)**: Theses in this category faced the challenge to introduce and supervise *Saros* in industrial or open source settings.
- XX is the sequential number of the thesis within the respective category.
- T is the type of thesis: **D**iploma (duration: 6 months), **M**aster (duration: 6 months), **B**achelor (duration: 3 months), **S**tudienarbeit (German word for a type of thesis similar to bachelor thesis, duration: 3 months).
- NO is the sequential number of that thesis.

[88] for details see `http://www.saros-project.org/concurrency` (accessed October 29, 2017)

The **categories represent the predominant focus of the respective thesis**, but in fact each thesis affected more than one category. For example, nearly every thesis had to contribute to the *Saros* release process and had to write tests or fix defects. Theses focusing on usability aspects also had to implement, modify or correct some functionality. Theses that developed a new feature often had to do refactoring and fix defects before or during the implementation of the new feature.

Year, Type	Thesis
2006-08 Fct-D1(01)	Developing an *Eclipse* extension for realizing distributed pair programming (Djemili, 2006) • Objective: Have an initial *Saros* version. • Milestones: Requirements elicitation and initial implementation of *Saros*. • Result: Alpha version of *Saros* for supporting one driver (write access) and one observer (with read-only access).
2007-10 Fct-S2(02)	Further development of a plugin for distributed pair programming (Gustavs, 2007) • Objective: Support multiple drivers in a *Saros* session and improve *Saros*' network reliability. • Milestones: Improve existing features, implement support for multiple observers and accordingly adapt awareness annotation coloring. • Result: First version of *Saros* website, multiple observer support, and an improved handling of network errors.
2008-07 Fct-D3(03)	Further development of an *Eclipse* extension for distributed pair programming in view of collaboration and communication (Rieger, 2008) • Objective: Peer-to-peer project synchronization and support of concurrent writing. • Milestones: Realization of a peer-to-peer connection for data transfer during project synchronization, implementation of operational transformation and conflict detection to synchronize concurrent writing events. • Result: Multiple driver support, and project synchronization via peer-to-peer connection.
2009-04 Qua-D1(04)	Further development of a tool for distributed collaborative software development (Jacob, 2009) • Objective: Have a well-working concurrent editing feature. • Milestones: Rectify concurrent editing functionality. • Result: Significant issues were fixed but not all defects in the network layer could be addressed.

Year, Type	Thesis
2009-06 Fct-S4(05)	Further agile development of a software tool for distributed, collaborative real-time programming (Rintsch, 2009) • Objective: Improve the workspace awareness annotations in *Saros* and implement the synchronization of the editing activities for multiple editors. • Milestones: Conceptual development and implementation of the specified work packages. • Result: Improved awareness mechanisms and multiple writer support in *Saros*.
2009-07 Fct-B5(06)	Surveying user feedback from usage of a tool for distributed pair programming (Dohrmann, 2009) • Objective: Collect statistical usage data during *Saros* sessions and have a *Saros*-integrated survey displayed to users. • Milestones: Requirements elicitation and implementation of an integrated survey system. • Result: A survey is displayed to users after every few *Saros* sessions, and users can agree to send anonymous usage data.
2009-10 Qua-D2(07)	Handling concurrency issues of a tool for distributed pair programming (Ziller, 2009) • Objective: Have a well-working concurrent editing feature. • Milestones: Detailed analysis and handling of concurrency issues for the multiple-writer mode. • Result: Introduced operational transformation in *Saros* with the Jupiter algorithm (see page 107) to handle concurrency issues during collaborative writing.
2009-10 Qua-B3(08)	The invitation process in *Saros* (Sóti, 2009) • Objective: Have a technically improved invitation process. • Milestones: Identify technical flaws in the invitation process of *Saros* and improve it by fixing them. • Result: Non-modal, cancelable invitation process with progress indicators better indicating the actual progress.
2009-11 Outr-D1(09)	Distributed pair programming in an industrial environment (Rosen, 2009) • Objective: Identify problems in establishing and using *Saros* in an industrial setting. • Milestones: Establish *Saros* in an industrial environment, qualitatively analyze the process of introducing and using *Saros* in the industrial setting. • Result: *Saros* established and used in an industrial environment, insights about pitfalls as well as general lessons learned.

Year, Type	Thesis
2010-02 Outr-B2(10)	Supporting various programming languages and plug-ins in a tool for collaborative software development (Kiwitt, 2010) • Objective: Have an overview of *Eclipse* plugins that work with *Saros*. • Milestones: Identify compatibility requirements for *Saros* with other *Eclipse* plugins and test a set of plugins for the most common programming languages. • Result: List of *Saros*-compatible *Eclipse* plugins (for specific programming languages).
2010-03 Qua-D4(11)	Handling network and security aspects in a tool for distributed pair programming (Szücs, 2010) • Objective: Have a more reliable and better testable network module. • Milestones: Develop a test suite to test the *Saros* network layer and improve its robustness. • Result: *Saros*-proprietary test framework, an accelerated file transfer during project synchronization, defect removal in the network layer.
2010-03 Fct-B6(12)	Improving communication possibilities in *Saros* (Loga, 2010) • Objective: Have a text chat that is connected to a session, which is not public to all *Saros* users, and have a VoIP feature for *Saros*. • Milestones: Re-engineer the existing chat, evaluate and implement a VoIP integration in *Saros*. • Result: A multi-user chat for all participants of a session. Basic VoIP functionality integrated in *Saros* but with high latency issues that make it unusable.
2010-03 Outr-D4(13)	Technical project management in the open source project *Saros* (Rintsch, 2010) • Objective: Guide and structure the development process of *Saros*, increase *Saros*' level of awareness in industry and open source projects, and identify technical problems that occur in long-term use of *Saros* in an industrial setting. • Milestones: Temporarily be the technical lead of the *Saros* project (since technical lead left project). • Result: Attempts of introducing *Saros* in an industry environment as well as in open source projects failed: The long-term use in industrial setting could not be established and none of the contacted open source projects gave *Saros* a trial – a few had a look at *Saros* but only reported defects.
2010-03 Outr-D3(14)	Distributed pair programming in open source projects (Starkmann, 2010) • Objective: Validate the usefulness of *Saros* in open source projects. • Milestones: Introduce *Saros* into open source projects. • Result: The active introduction of *Saros* in open source projects was not successful.

Year, Type	Thesis
2010-04 Fct-B7(15)	Improved presence through screen sharing in a tool for distributed pair programming (Lau, 2010) • Objective: Have a screen sharing functionality integrated in the *Saros* user interface. • Milestones: Evaluate requirements for an integrated screen sharing feature, its technical feasibility, and integrate it in *Saros*. • Result: The screen sharing feature was not finished: the host can share his screen but the client can operate the host's screen without authentication.
2010-05 Qua-D5(16)	Improving an XMPP Library for use in distributed pair programming (Staib, 2010) • Objective: Have a more reliable network module with reduced code complexity. • Milestones: Adapting the *Smack* library[89] (used for XMPP message transfer) to the needs of distributed pair programming. • Result: A patch for the *Smack* library improving data exchange during project synchronization in *Saros*.
2010-07 Qua-B6(17)	Analysis and extension of the VoIP functionality in *Saros* (Pütz, 2010) • Objective: Make the VoIP implementation in *Saros* usable. • Milestones: Analyze and fix latency problems in the VoIP functionality in *Saros*. • Result: Latency problems could be improved but not be fixed; the feature is still not usable.
2010-06 Fct-B8(18)	Statistical evaluation of distributed pair programming sessions (von Hoffen, 2010) • Objective: Understand the actual usage of *Saros* to get indications what to focus on in the *Saros* development. • Milestones: Extend the existing statistics module of *Saros* and analyze existing statistical data. • Result: Due to technical issues, no sufficient data for analyzing the usage of *Saros* was available. The statistics module, however, was extended and can now collect more detailed data.
2010-11 Fct-B9(19)	Integration of version control systems in a tool for distributed pair programming (Haferburg, 2010) • Objective: Allow users to choose to get their project files from a VCS repository instead of using the *Saros* synchronization mechanism. • Milestones: Requirements analysis and integration of a version control system in *Saros*. • Result: The basic functionality for using a VCS from within *Saros* was implemented but no fallback mechanism in case of problems with the VCS repository. The user interface for using the VCS integration is complex.

[89] `http://www.igniterealtime.org/projects/smack/` (accessed November 11, 2017)

Year, Type	Thesis
2010-11 Fct-D10(20)	Distributed debugging (Erdogan, 2010) • Objective: Support distributed multi-user debugging in *Saros*. • Milestones: Feasibility analysis of distributed debugging based on the Java debugger • Result: A lot of tricky aspects of distributed debugging were identified but could not be solved satisfactorily. Therefore, no solution for distributed debugging was implemented.
2010-11 Qua-B7(21)	Improving *Saros* in unreliable networks with high latency (Bauch, 2010) • Objective: Improve the stability and performance of *Saros*' network connections. • Milestones: Analyze the adequacy of the selection of the different automatically determined connection protocols in *Saros*. • Result: Defects in the *Saros* network layer were fixed, but there are still delays in *Saros*' event transmission.
2010-12 Fct-D11(22)	Iterative, prototype-driven development of a whiteboard feature (Jurke, 2010) • Objective: Develop a usable distributed whiteboard in *Saros* allowing freehand drawing and the use of predefined shapes. • Milestones: Requirement elicitation and prototype-driven development of a shared whiteboard. • Result: A basic shared whiteboard was implemented and related parts of the *Saros* network layer were refactored.
2011-02 UX-B1(23)	Design and implementation of the new *Saros* user interface (Bitterling, 2011) • Objective: Review and adapt the *Saros* user interface with regard to user interface principles. • Milestones: Integrate the different *Saros* views (session list, buddy list and chat view) into one view. • Result: The integrated *Saros* view is more clear for users but still has some confusing interface elements.
2011-04 Qua-D8(24)	Introduction of a testing process (Chen, 2011) • Objective: Introduce a strategic testing process for the *Saros* project and simplify *Saros*' proprietary test framework. • Milestones: Analyze status quo of the *Saros* testing process, develop a concept for a new adequate testing process, and make the existing *Saros*-proprietary test framework easier to use. • Result: A simplified and well documented test framework that helps developers to more easily write tests for *Saros*.

Year, Type	Thesis
2011-03 Fct-B12(26)	Supporting multiple projects in a *Saros* session (Dohnert, 2011) • Objective: Allow users to start a session with more than one project or to add multiple projects during a session. • Milestones: Develop an approach for adding a second project to a running session. Then extend this approach to more than one project. • Result: Sharing multiple projects is possible now. However, it makes the invitation scenario more complex from a technical as well as from a user perspective.
2011-04 Qua-M9(25)	Construction of a test framework for writing unit tests in the *Saros* project (Cordes, 2011) • Objective: Have a test framework with which in particular the network layer can be tested more easily. • Milestones: Provide useful abstractions of relevant *Saros* components and make them available in a test framework. • Result: The test framework provides some basic functionality but does not work reliably. Moreover, developers had problems understanding its concept and thus using it.
2011-05 UX-M2(27)	Improving the out-of-box experience in *Saros* using heuristic evaluation and usability testing (Kahlert, 2011) • Objective: Lower the hurdle for *Saros* users to getting started. • Milestones: Analyze and fix usability problems during the *Saros* installation and configuration process. • Result: 41 out of 61 identified usability issues were fixed. Some new became evident during the last iteration of usability testing.
2011-07 Qua-D10(28)	Improving the architecture of the eDPP software *Saros* by introducing a documented module perspective (Belousow, 2011) • Objective: Have a concept for an architecture that makes *Saros* internal structure easier to maintain and understand. • Milestones: Analyze the architecture of the *Saros* source code and develop a concept of how to improve it. • Result: A new architecture concept which requires a lot of changes in *Saros* that could be realized by an iterative transformation and several refactoring steps.

Year, Type	Thesis
2011-08 Qua-D11(29)	Stability and testability improvements of the network layer in *Saros* (Gustavs, 2011) • Objective: Make the network layer of *Saros* more reliable. • Milestones: Evaluation of the potential usefulness of the *Eclipse* communication framework for *Saros*, a structured analysis of the network layer, the removal of existing issues in the network layer, and an examination of the testability of the network layer resulting in suggestions for improvement. • Result: An adaptive network protocol switch: Depending on the user's network environment, *Saros* choses the best network protocol to use.
2011-09 Fct-M13(30)	Partial project synchronization and need-based file synchronization in *Saros* (Held, 2011) • Objective: Enable a more efficient resource handling in *Saros* by supporting an automatic inclusion of unshared files in a running session when they are needed. • Milestones: Support synchronization of single files or parts of a project instead of only whole projects and make this a usable feature in *Saros*. • Result: The feature was prototypically implemented. It involved changes in the technical realization of the synchronization process as well as sophisticated usability considerations with regard to the behavior for automatically adding files.
2011-09 UX-B3(31)	Evaluation of the mechanisms for displaying workspace awareness in *Saros* (Solovjev, 2011) • Objective: Improve the workspace awareness mechanisms in *Saros*. • Milestones: Observe and evaluate the usage of the workspace awareness features provided by *Saros* in an industrial setting. • Result: Users are satisfied with the awareness during concurrent editing, but awareness for other areas of the shared workspace is not sufficient.
2011-10 Qua-D12(32)	Improvement and maintenance of the documentation in the eDPP software *Saros* (Johannsen, 2011) • Objective: Have an actual and complete documentation of the *Saros* source code on the *Saros* website. • Milestones: Revise outdated documentation and develop a concept to keep the documentation up-to-date for the long term. • Result: The website was restructured. Documentation guidelines were introduced as well as the new role of a documentation manager who assures that released changes are documented.

Year, Type	Thesis
2011-11 UX-B4(33)	Analysis and evaluation of improvement possibilities of the *Saros* user experience (Narweleit, 2011) • Objective: Improve the user experience of *Saros*. • Milestones: Identify problems in the *Saros* user interface that negatively influence the user experience. Compare *Saros* to other related tools and evaluate inspiring ideas from which *Saros*' user experience could also benefit. • Result: Inconsistent or senseless interaction opportunities were removed from the user interface. Concepts to potentially improve the user experience of *Saros* were suggested.
2011-11 Qua-B13(34)	Introduction of a continuous integration environment and improvement of the test framework (Rossbach, 2011) • Objective: Have a continuous integration running automated tests. • Milestones: Develop and realize a concept for integrating a continuous integration server in the *Saros* development process. • Result: Regression tests run automatically and the status is shown on a corresponding webpage. Independent from that, there is still a low test coverage in Saros.
2012-04 UX-D5(35)	Evaluation and revision of the usability of *Saros* (Waldmann, 2012) • Objective: Improved efficiency and effectiveness of *Saros* during an eDPP session • Milestones: Find and fix usability issues by an iterative approach of empirical and analytical usability evaluation. • Result: 128 usability issues have been identified and categorized of which 54 could be fixed. Those that could not been fixed in particular referred to experimental features.
2012-04 Qua-D14(36)	Improving the algorithmic core: concurrent editing (Warnatsch, 2012) • Objective: Eliminate deadlocks and undesired behavior of *Saros* during concurrent editing. • Milestones: Extensively analyze the program flow during concurrent editing by code reviews and debugging. • Result: A conceptual solution was developed and implemented in some parts, but some problems with regard to concurrent editing could not be solved.

Year, Type	Thesis
2012-05 UX-M6(37)	User-centered design of a usage concept for the *Saros* whiteboard considering the support of drawing tablets (Degener, 2012) • Objective: Have an improved multi-user whiteboard in *Saros* which can be operated using a mouse or a drawing tablet. • Milestones: Develop a usage concept based on personas and implement this in an iterative approach with prototyping and user testing. • Result: A multi-user whiteboard supporting mouse and drawing tablet interaction was developed. Due to technical shortcomings, it was not integrated in the *Saros* production code.
2012-07 Qua-B15(38)	Removal of stumbling blocks in the *Saros* development process (Freyther, 2012) • Objective: Facilitate getting started with *Saros*' development for new developers. • Milestones: Identify potential improvements in the development process and in the *Saros* architecture. Improve the on boarding process and the documentation for new developers. • Result: The *Saros* source code was migrated from *SVN* to *Git* and a comprehensive documentation for new developers was created.
2012-08 UX-B7(39)	Improvement and rework of the *Saros* website using a user-centered design approach (Kretzschmann, 2012) • Objective: Identify the main target groups of the *Saros* website and adapt the website to meet their needs. • Milestones: Identify the main target groups, their expectations, and needs. Restructure the website and improve its content and visual appearance. • Result: The contents of the website were migrated to a new technical platform along with a more appealing visual design.
2012-11 UX-M8(40)	Improvement of the usability of *Saros* using a user-centered design approach (Spiering, 2012) • Objective: Minimize the gap between the user's mental model of *Saros* and *Saros'* actual conceptual model. • Milestones: Capture the target user groups' mental model about *Saros* using methods from the user-centered design approach. Compare these models to *Saros'* actual conceptual model and develop solutions to bring both closer together. • Result: 34 usability issues were identified, 9 problems could be solved in the context of the thesis, and for the remaining problems solutions were suggested.

Year, Type	Thesis
2013-02 Qua-M16(41)	Improving the reliability of *Saros* using root cause analysis (Starroske, 2013) • Objective: Improve the general reliability of *Saros*. • Milestones: Identify defects in the *Saros* source code by applying the root cause analysis approach and fix them. • Result: Defects were found and fixed. Due to the limited time available for this thesis and the considerable effort of the root cause analysis, it could only be applied to parts of *Saros'* source code.
2013-07 Qua-D17(42)	Facilitation of getting started in the *Saros* development process by eliminating hurdles (Stejngardt, 2013) • Objective: Identify issues for new *Saros* developers with setting up the test and development environment for *Saros*. • Milestones: Identify confusing, abundant, or missing information by interviewing and observing new *Saros* developers. Solve these problems by providing an appropriate, compact guide tailored for new developers. • Result: 93 problems were identified and about 90 percent of them could be solved. Final user tests showed, that with the new getting started guide all participants were able to set up their development environment much quicker than before and without running into dead ends.
2013-10 Qua-M18(43)	Analysis and improvement of the architecture of a concurrent and distributed software system (Schlott, 2013) • Objective: Improve the architecture of *Saros* to be more coherent. • Milestones: Identify and fix architecture breakers. • Result: Problems concerning the coherency of the architecture were identified and solved. The lessons learned were synthesized in an architecture documentation.
2014-02 Fct-B14(44)	Development of a server prototype for *Saros* (Bussas, 2014) • Objective: Some *Saros* users expressed the wish to have a *Saros* server holding several running sessions that clients can join and leave, comparable to the Interaction Directory pattern by Schümmer and Lukosch (2007). • Milestones: Conceptually develop and implement a server prototype that initially can hold one running *Saros* session. • Result: A prototype was implemented which revealed some general technical issues regarding a *Saros* server. Moreover, a list of features for the further development of a *Saros* server was provided.

Year, Type	Thesis
2014-04 UX-B9(45)	Integrating end-user feedback in the *Saros* development process (Szaffranietz, 2014) • Objective: Implement a mechanism to systematically integrate user feedback from different channels in the *Saros* development process. • Milestones: Evaluate potential and existing feedback sources (like the integrated *Saros* survey and mailing lists) with regard to their provided feedback. Develop and implement a concept of how to systematically process and use this feedback in the *Saros* development process. • Result: A conceptional idea of the different feedback types were provided as well as some e-mail templates to respond to specific types of user feedback.
2014-06 Fct-B15(46)	Implementing a mechanism for transmitting information from the *Eclipse* console in *Saros* (Bieber, 2014) • Objective: Some *Saros* users expressed the wish to share the built-in *Eclipse* console with all participants of a *Saros* session. • Milestones: Develop a proof of concept based on a prototype implementation. • Result: A prototype for sharing the *Eclipse* console was implemented. In addition, a list of features and improvements for the further development of this feature was provided.
2014-08 Fct-B16(47)	A need-based status input for the *Eclipse* plugin *Saros* (Formisano, 2014) • Objective: In the context of a *Saros* server or in an Interaction Directory, users should be motivated to collaborate. • Milestones: Develop ideas to push collaborations of developers. Provide a prototype which illustrates some of these ideas. • Result: A prototypical implementation of a 'Help Request' feature for the *Saros* server was provided along with considerations about a gamification-based approach to encourage its usage.
2014-10 Fct-M17(48)	Improving the action awareness in the open source plugin *Saros* (Durmaz, 2014) • Objective: *Saros* only shares editing activities. Actions like using dialogs or wizards are not shared. This situation should be improved by extending the action awareness mechanisms of *Saros*. • Milestones: Evaluate possible technical approaches and an adequate granularity for the action awareness information. Provide a prototypical implementation. • Result: An Activity Log was prototypically implemented, showing a rough overview of the remote partner's activities other than editing.

Year, Type	Thesis
2014-12 Qua-M19(49)	Operationalizing architecture in an agile software project (Solovjev, 2014) • Objective: Improve how *Saros*' architecture is visible in the development process in order to avoid architecture erosion[90] and to properly maintain and improve it. • Milestones: Evaluate tools and practices for avoiding architecture erosion in *Saros* and integrate them in the *Saros* infrastructure. • Result: A compliance check mechanism adapted to the *Saros* architecture was provided which can be applied to *Saros* source code contributions.
2015-04 Qua-B20(50)	Refactoring of the *Eclipse* plugin *Saros* for a porting to other IDEs (Lasarzik, 2015) • Objective: To avoid source code redundancy when porting *Saros* to the *IntelliJ IDEA* IDE, *Saros'* source code should be divided in IDE-independent core functionality and IDE-specific classes. • Milestones: Analyze IDE dependencies in the *Saros* source code and uncouple it in an IDE-independent core and IDE-specific classes. • Result: The *Saros* core now contains basic functionality detached from IDE-specific classes.
2015-04 Fct-M18(51)	Evaluating the use of a web GUI to unify GUI development for IDE plugins (Cikryt, 2015) • Objective: *Eclipse* and the *IntelliJ IDEA* IDE use different graphical widget toolkits. In the context of porting *Saros* to the *IntelliJ IDEA* IDE, this means that GUI code must be replicated in the graphical widget toolkit used by *IntelliJ IDEA* IDE. To avoid this GUI code duplication, it should be evaluated whether a HTML-based GUI implementation, which could be used for both IDEs, is a feasible approach. • Milestones: Evaluate possible HTML-based GUI technologies and provide a prototypical implementation of a HTML-based *Saros* interface. • Result: A HTML-based prototype of the *Saros* interface.

[90] *"Software architecture erosion designates the progressive gap normally observed between the planned and the actual architecture of a software system as implemented by its source code"* (Terra et al., 2012). In the development process, it creeps in *"due to violations of the architecture"* (Perry and Wolf, 1992).

Year, Type	Thesis
2015-04 Qua-B21(52)	Automated configuration of the build server in the *Saros* project with Salt and *Git* (Hobusch, 2015) • Objective: The build system of *Saros* involves configuration management, test suites, version control, as well as a review system. Each tool has to be separately configured, and this complex infrastructure is hard to understand and maintain. To ease the administration of the build infrastructure, it should be declaratively described in one place. • Milestones: Evaluate approaches of how to declaratively manage the review and the continuous integration tools considering the technical constraints of the actual build system. • Result: A solution for an eased configuration management of the build system was provided.
2015-07 UX-B10(53)	Faster session start-up in *Saros* (Damm, 2015) • Objective: When users start a *Saros* session, their files are synchronized. This is done via data archives from the host, which are transferred to and unpacked on the remote site. The users have to await completion before they can start to edit the artifacts. This waiting period should be shortened for the users. • Milestones: Evaluate alternative/faster mechanisms for the initial data exchange of a session so that the users can proceed their session establishment without having to wait for the data transfer. • Result: When trying to eliminate the transmission of the whole project at the beginning of a session, two central issues were identified. A provisional concept was presented.
2015-09 Fct-M19(54)	User-centered development of a JavaScript and HTML-based GUI for *Saros* (Sieker, 2015) • Objective: Start implementing a HTML-based *Saros* interface. Use the user-centered design approach to address usability issues of that interface. • Milestones: Develop a concept to migrate the *Saros* interface to a HTML-based technology, start implementing it and validate its usability. • Result: A basic implementation of the HTML- and JavaScript-based *Saros* GUI.

Year, Type	Thesis
2015-11 Fct-B20(55)	Development of an infrastructure for exchangeable variants of project transmission in *Saros* and further development of existing ones (Theus, 2015) • Objective: One variant for project transmission is available in *Saros* which transfers a project as a single zip file in the beginning of a session. Another approach that transmits a project file by file in the background is prototypically implemented. Both variants should be available in *Saros*. • Milestones: Integrate the prototype in *Saros* in addition to the exiting transmission approach. Allow to select which approach to use and extend the prototype's features. • Result: Both transmission variants are available in Saros, based on a structure that is open for the integration of further variants. Some problems of the prototype could not be fixed.
2016-01 Fct-M21(56)	Development and evaluation of a location-independent session server for the *Saros* project (Washington, 2016) • Objective: Currently, *Saros* requires one participant to be the host of a session. When the host closes the session, the session ends for all participants. The aim is to have a server that plays the host role for *Saros* sessions and thus enables long-lasting sessions that can be arbitrarily left and entered by participants. • Milestones: Development and implementation of a concept that allows non-host participants to share projects. Preparation of the *Saros* core for the independent session server. Implementation of the session server. Evaluation of the server with regard to resource management. • Result: In preparation to the *Saros* version for *IntelliJ*, a first version of an Eclipse-independent session server was developed. Projects can be shared from within sessions on the server.
2016-04 Qua-B22(57)	Improvement and extension of parts of the *Saros* core (Sungaila, 2016) • Objective: Remove code duplicates that exist for the *Saros Eclipse* and *IntelliJ* version but that can be moved to the IDE-independent core. Moreover, add functionality to the new *Saros* HTML GUI. • Milestones: Analyze which code duplicates can be joined to a single version in the core. Move such identified code parts. Extend the *Saros* core (written in Java) so that it can communicate with the new HTML/JavaScript GUI. • Result: Eight code duplicates were refactored in the *Saros* core, others were identified but could not be moved in the context of this work.

Year, Type	Thesis
2016-07 Qua-B23(58)	Facilitating further development and extension of the JavaScript and HTML GUI of *Saros* (Weber, 2016) • Objective: Ease the process of getting started for new developers who want to work on the *Saros* HTML GUI. • Milestones: Analyze the HTML GUI's source file structure and evaluate the hurdles for adding or changing functionality in the GUI. Restructure the folder structure of the HTML GUI source code so that new developers can get going more easily. • Result: Several changes in the HTML GUI's folder structure which facilitate a better understanding of the code.
2016-10 Qua-B24(59)	Simulation of a network environment for distributed pair programming (Giesler, 2016) • Objective: Develop a program that simulates network effects for *Saros*, to be able to test *Saros* in problematic network environments. • Milestones: Develop an approach to integrate the network simulation in *Saros*' test framework. Implement this network simulation and make it configurable via a configuration file. • Result: A configurable network simulation program for testing *Saros* in suboptimal network environments is available.
2017-05 Fct-B22(60)	Preparing and Releasing a functioning alpha of *Saros/I* (Bouschen, 2017) • Objective: Convert the prototypical version of *Saros* for *IntelliJ* (*Saros/I*) into a version ready for release. • Milestones: Analyze and improve existing code. Add new features that are necessary to have a basic, useful version of *Saros* for *IntelliJ*. • Result: Due to very time-consuming, unforeseen complications, only the first aspect of analyzing and improving existing code could be met to some degree.

Table 2.2: Overview of theses which contributed to *Saros*' development.

Figure 2.5 summarizes the achievements of all theses in the *Saros* project. The effort is expressed in person-months: For a master or diploma thesis, six person-months are taken as a basis, for a bachelor thesis or Studienarbeit it is three person-months. **The unadjusted sum of all student theses is 270 person-months, which equals 22,5 years of development**.

This big number does not, however, take into account the fact that new project members need time to familiarize with the project, which cannot be considered productive development time. Since many theses contributed to *Saros*, there are proportionally many settling-in periods. Time for research, evaluation, and concept development is also part of each thesis and thus included in the listed person-months. On the other hand, the time invested by the thesis supervisors, the project lead, as well as the effort of other developers who contributed to the open source project, is not included in this number. Over the years, about 5 working group members have contributed on the *Saros* project, each for several years, as well as another very skilled voluntary open source developer. Including only 1,5 years for each of them, adds 9 years to the development effort of *Saros*. In summary, ***Saros* includes a development effort**

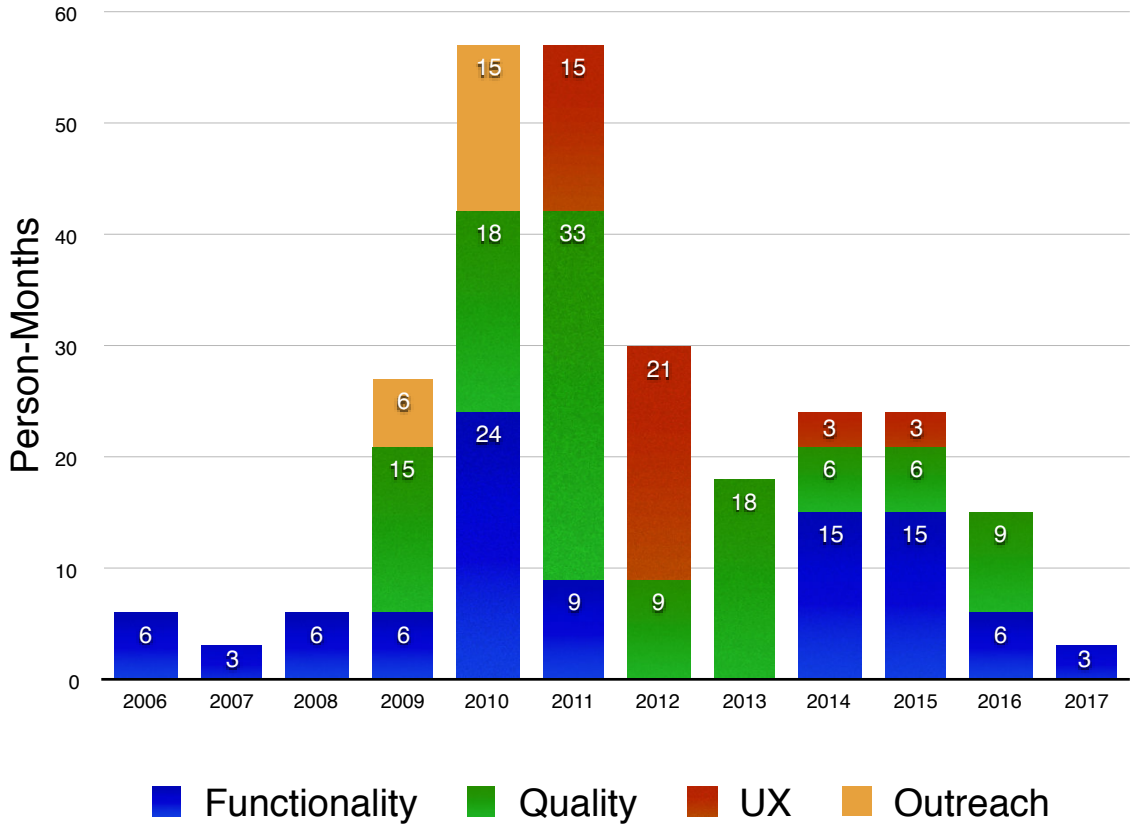

Figure 2.5: Person-months of theses that contributed to *Saros*

of more than 31 years.

2.3.2 Phases in Saros' Development

The last section summarized the amount of effort that has been spent on the *Saros* development. This section evaluates the work effort per category or product aspects of the respective theses, taking into account the first column in table 2.2.

Figure 2.5 shows the amount of work effort per work product aspect and reveals the following course of development focus:

1. The first three years of development mainly focused on functionality and had low project participation.
2. After about three years the participation in the project increased.
3. In 2009 and 2010, attempts were made to increase *Saros*' outreach.
4. In 2009, an increased focus on quality improvements started.
5. Functionality-focused theses decreased in 2011.
6. Theses aiming at a better user experience of *Saros* considerably increased in 2011.
7. In 2011, the focus shifted to quality improvements.
8. In 2014, the number of feature-related theses increased again.

These trends reflect four characteristic phases in the development history of *Saros*, which are presented in the following paragraphs. For each phase, the stage of development and actual problems will be described as well as what was thought to happen, what actually (did not) happen and what was learned from this.

2.3.2.1 1 – Phase of initial development

The idea of developing a tool for distributed pair programming was born, and in the first three years of development, the essential features for concurrent editing along with awareness annotations were implemented. In these years, *Saros* struggled with defects as well as with stability, robustness, and performance issues. The **basic functions had to be fixed and improved repeatedly**.

2.3.2.2 2 – Phase of attempts of establishment

After the start-up period, the essential features for realizing distributed pair programming more or less worked, and people gave positive feedback concerning the idea of a tool like *Saros*. We thought that people, in particular open source developers, look forward to a tool like *Saros* and would willingly adopt it. This was a fallacy. **Outreach efforts (starting in 2009) of introducing *Saros* into several open source projects failed**. Reasons for the non-acceptance of *Saros* were either rejection of the tool itself or of pair programming in general, or that *Saros* was considered inappropriate for the working context in question (for example due to asynchronous development, the usage of a programming language not supported by *Saros*, or the time needed to familiarize with *Saros*). Others stated that they already knew another tool providing the functionality of *Saros*, but without mentioning its name and whether they actually use it. Another very fundamental issue was that developers did not want to use *Eclipse* for their development work. Even people who were willing to try *Saros* struggled with fundamental problems and lost further interest in the tool. (Starkmann, 2010)

In essence, **industrial outreach efforts failed**. Since 2010, 73 companies have been contacted to which *Saros* and the eDPP approach were presented. Some of them showed interest, but none of them really used *Saros*. Some rare team members gave *Saros* a try, but often struggled with usability as well as technical hurdles in setting up *Saros* and synchronizing their projects, as well as with unreliable and faulty behavior of *Saros*. In particular, the poor out-of-box experience and the clumsy invitation process put off many potential users. All in all, **Saros did not pass the test of productive fitness** from a technical as well as from a user-friendliness perspective.

Kent Beck, one of the originators of XP who was fond of the idea of *Saros*, gave it a try in 2010. His opinion sums it up nicely[91]:

"Saros (pronounced 'zar-ose', btw) is a set of extensions to Eclipse *to support real-time collaboration. It is a research prototype at the moment, and as such has some rough edges. In 15 or 20 years, most programs will be written through real-time collaboration, so for me it's worth a bit of pain today to experience the future."*

One comment of this blog entry declares *Saros* as immature but the best option at the moment: *"I've been following* Saros *and alternatives since 2008. I agree with Kent that it is the future – once an implementation is completed with the sync bugs worked out.* Saros *remains the best implementation of all, though, still today."*

[91] Unfortunately, Kent Beck's blog `http://www.threeriversinstitute.org/blog/?p=584` is not available any more, but his quote can still be found on `https://web.archive.org/web/20130112231612/http://www.threeriversinstitute.org/blog/?p=584` (accessed October 29, 2017).

2.3.2.3 3 – Phase of consolidation and UX work

Facing the fact that *Saros* fails in practice, in 2009, the project started to focus on *Saros*' internal quality improvements as well as on usability work, keeping functional enhancements in the background.

Saros was mainly developed by student theses, which implied a high fluctuation rate of the project members, each being active in the project only for about three to six months. Accordingly, the **individual feature contributions were more or less isolated contributions**, flanged to the system, often without having a a deep understanding of *Saros*' architecture and concurrency demands. Usually, **usability was not considered to be a serious matter**. To tackle these technical and user-related debts, the years 2011, 2012, and 2013 were characterized by technical consolidation and usability improvements.

2.3.2.4 4 – Phase of industrial acceptance

In 2013, when *Saros* had improved so much to be actually used, users began to appear from industry as well as from freelance and private contexts. Is was unclear how long they had been using *Saros* and where they had hidden until then. Each week, however, the project got user feedback and requests via the built-in *Saros* survey or the project's mailing list. Finding users who would agree to be recorded for research purposes was still not successful. Although some users showed interest and considered the research aim to be valuable, the common response 'I will talk with my colleague/boss/...' was not followed by anything concrete. **One pair of users**, however, who sent a support request and **agreed to be recorded for research purposes**, kept their word. **They finally provided the data this inquiry is based on**.

As mentioned above, in this phase, the *Saros* project got a lot of user feedback, such as the following:

E-mail from a *Saros* user, August 2013: *"I had just finished my first session with Saros and was extremely happy. We have been talking about this kind of collaboration since about the time I started at this place last October. [...] I saw* Saros *and was extremely excited. Needless to say, one awesome session later, and I'm literally working with my team lead who is in India (its friggin late there lol) and* Saros *is amazing."*

Responses from the built-in *Saros* survey (January 2014):

- How did your latest *Saros* session work out? *"Great."*
- Which goals did you want to accomplish in your latest *Saros* session? *"Peer developer got stuck writing a function. Used saros to assist him."*
- What didn't go well (if anything) and what happened? *"Everything worked fine."*
- Do you have any other comments to share with us (e.g about the interface, specific features or possible improvements)? *"We use share project from the saros menu because it allows us to select only the sources otherwise its too slow to sync."*

Response from the built-in *Saros* survey (April 2014):

- How did your latest *Saros* session work out? *"Quite good."*
- Which goals did you want to accomplish in your latest *Saros* session? *"Pair programming with homeworking colleague."*
- What didn't go well (if anything) and what happened? *"Quite difficult to debug together. The pair doesn't see my breakpoints, and the values of what variables."*

Response from the built-in *Saros* survey (March 2015):

- How did your latest *Saros* session work out? *"Productive and comfort"*

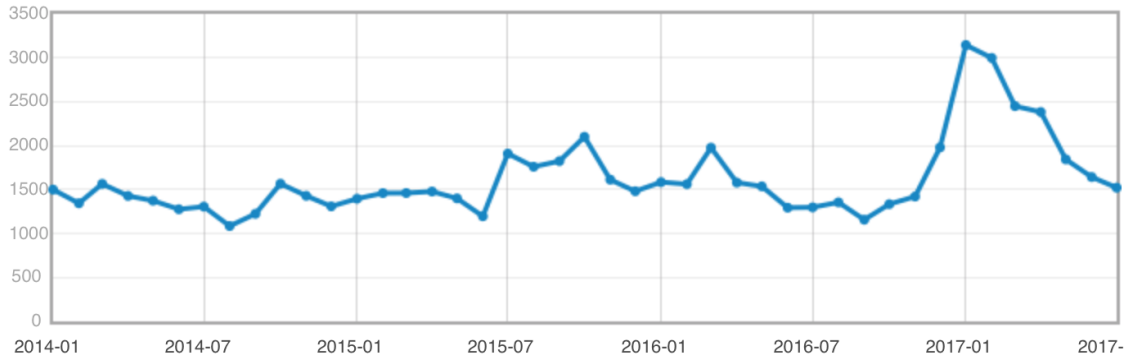

Figure 2.6: Monthly download rate of *Saros* on SourceForge

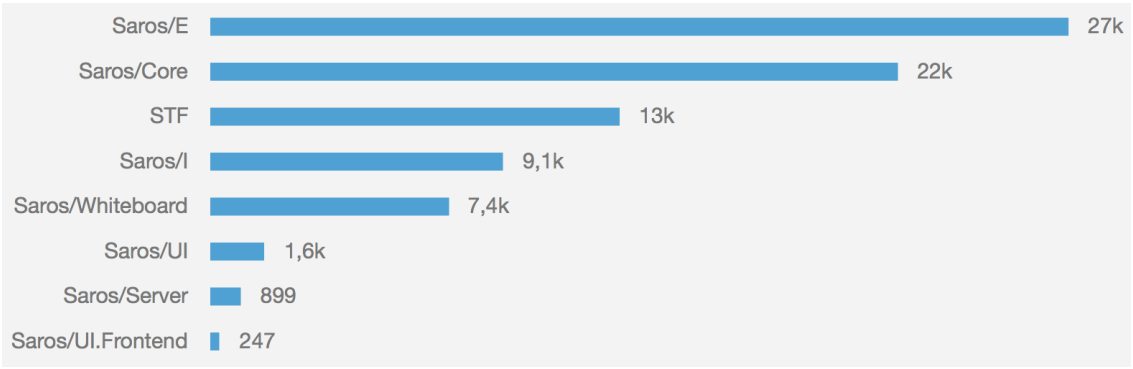

Figure 2.7: Current number of lines of code (LOC) of the components of *Saros*

Source `http://saros-build.imp.fu-berlin.de/sonarqube/overview/structure?id=saros-complete` (accessed July 17, 2017)

- Which goals did you want to accomplish in your latest *Saros* session? *"To use my own environment and hardware (keyboard and mouse) in pair programming."*
- What didn't go well (if anything) and what happened? *"Accepting sharing often hangs for a while before open dialog with project destination."*
- Do you have any other comments to share with us (e.g about the interface, specific features or possible improvements)? *"Just-in-time syncing works perfect, peer's changes highlighting is also awesome."*

And finally some statistics: From January 2014 until July 2017, *Saros* had more than 69.000 downloads[92] with often more than 1500 downloads per month, as shown in Figure 2.6.

Currently, *Saros* has 80.583 lines of code (LOC). Figure 2.7 shows their distribution for the main components of *Saros*.

Figure 2.8 shows the LOC trend in *Saros* since 2013. The curve also reflects the course of development focus of *Saros*:

- In 2013, the development focus was still on improving *Saros*' code quality. The decrease of LOC starting at the end of 2013 is mainly due to the removal of the integrated screen sharing and VoIP functionality. Both features never reached a state where they were fit for purpose. Other tools like *Skype* or *Google Hangouts* specialize in these purposes and can be used during a *Saros* session.

[92] `https://sourceforge.net/projects/dpp/files/stats/timeline?dates=2014-01-01+to+2017-07-28` (accessed November 11, 2017)

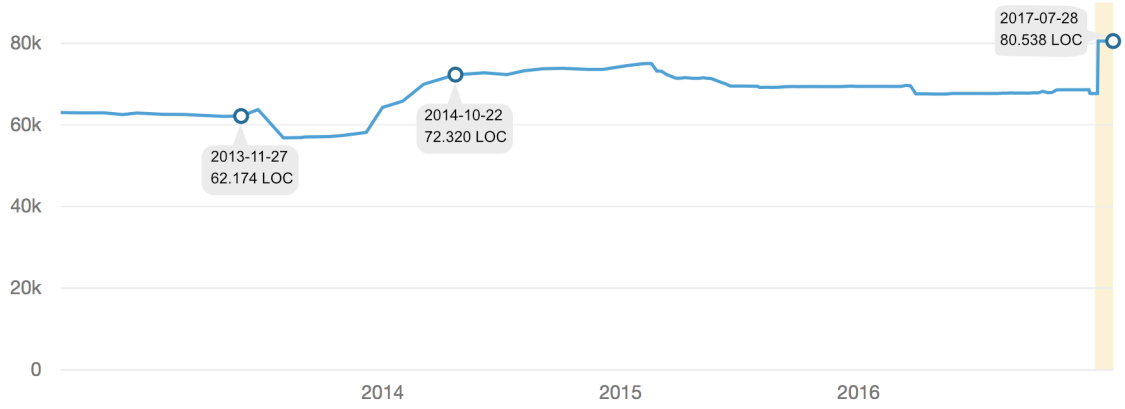

Figure 2.8: *Saros* LOC since 2013

Source: `http://saros-build.imp.fu-berlin.de/sonarqube/overview/structure?id=saros-complete` (accessed July 17, 2017)

Other code optimizations involved rather few code removals but led to a reduced complexity of *Saros*' source code.

- In 2014, the focus turned back to adding functionality and to the development of *Saros* for *IntelliJ* (*Saros/I*). In Figure 2.8, this is reflected by the increase in the lines of codes.
- Since 2015, there has been a moderate decrease in LOC, mainly due to the removal of redundant code in *Saros/I*. For some time, there was redundant code which provided basic functionality once for *Saros* for *IntelliJ* and once for *Saros* for *Eclipse*. During *Saros* server development such basic IDE-independent functionality was moved to the so-called *Saros* core. Successively, the redundant IDE-independent code in *Saros/I* was replaced by functionality from the core.
- The yellow bar at the end of the chart indicates the last month of the chart, which is July 2017. In this month, there is a strong rise of about 13.000 LOC because the *Saros* test framework was included in the LOC count. Until then, the Saros test cases were part of the LOC measurement, but not the code of the Saros-specific test framework.

The efforts on stability and user friendliness started to bear fruits in 2013: After several years of slack with regard to user acceptance, *Saros* recored more and more users. In the course of this research project, **I was the motor behind all UX-related work on *Saros*** and thus considerably contributed to *Saros'* success.

Chapter 3

Research Framework and Limitations of the Study

Chapter Structure – *What to expect in this chapter and why*

This chapter addresses the methodical aspects of this inquiry, the research framework. It starts by specifying the overall research question with refined research questions because their nature is relevant for methodical decisions. Subsequently, Section 3.2 "Research Framework" presents the research strategy employed, the unit of analysis (the pair), the data collection instrumentation (non-participatory audio, video, and screen recordings) as well as the data analysis method employed (elements of the Grounded Theory methodology). The chapter closes with a discussion of the study's limitations and threads to validity.

As discussed in Section 1.4.5.3 "DPP Research – Outstanding Issues", this thesis is an exploration of a relatively unknown area. The research question is of empirical nature, aiming at an initial understanding of behavioral phenomena during eDPP, and focuses on the following specific aspects in eDPP (see also Section 1.3 "Research Question"):

How do

- reduced workspace awareness and
- independent editing capabilities

impact the eDPP process?

These aspects are further detailed in Section 3.1 "Refined Research Questions". Due to the lack of knowledge about the eDPP process and the impact of these eDPP-inherent aspects, **this study is of exploratory character**.

Initially, the study was intended to characterize the process differences between PP and eDPP. This has proven to be too difficult: When studying the actual problem-solving of industrial software developers, no two observations are ever exactly comparable. Thus, an observed behavioral difference between two sessions cannot clearly be ascribed to originate from characteristics of the pair, from characteristics of the tasks, or from the fact that one pair is working locally and the other remotely.

The data for this study has been obtained from eDPP sessions of a single pair of quite eDPP-experienced professional developers doing their regular work. In total, the eDPP sessions were recorded over a period

of one week and resulted in about 16 hours of data (see details in Sections 3.2.2 "Unit of Analysis: Pair of Experienced eDPP Developers" and 3.2.3 "Data Collection: Non-Participatory Observation").

Due to the characteristics of the data and the research question, this inquiry is a single-case study according to Yin (2014) (see Section 3.2.1 "Research Strategy: Single-Case Study"). In contrast to the definition by Yin (2014), however, the intended understanding of this study does not claim to be holistic (no session-wide assessment of phenomena), nor does it analyze the contextual conditions of the pair. Nevertheless, the research situation is well suited to be realized as a case study: **the research situation is to understand what happens in a specific, contemporary real-world situation without control over the behavioral events** (Yin, 2014).

Originally, several pairs doing eDPP have been studied. These are data sets of student as well as professional novices though, who apparently coped with the new style of working in eDPP and technical issues. It was decided to only use the most competent and experienced professional pair and thus turn the study from a multiple-case study delivering a complicated, mixed message into a revelatory, single-case study providing the proof of existence for something that has not been examined before: Industrial distributed pair programming which appears to be as effective as if it had been local pair programming.

As described in Section 1.3.2 "Tool-Related Contributions", finding professional study participants was extremely difficult. Only one pair of developers remained as a sound data base. In light of a case study, this is unproblematic because a case study does not aim at statistical generalizability in terms of replicable results from a broad sample. Instead, a sample is *"the opportunity to shed empirical light about some theoretical concepts or principles"* (Yin, 2014). As will be discussed in Section 3.3 "Limitations and Threats to Validity", for the results of this study this means that they do not claim to be complete with regard to the research question. They provide valid insights about a specific situation, though. Since the specific case is carefully described, the result's validity for other situations can be assessed.

Due to the explorative character of this work, Grounded Theory methodology (GTM) (or parts of it) were chosen as a data analysis method. GTM is well suited for investigatively breaking up complex data by conceptualizing observed events/phenomena (see Section 3.2.4 "Data Analysis: Grounded Theory Methodology").

Finally, the research framework can be outlined as follows:

- Knowledge acquisition: empirical
- Epistemological interest: analytical, practical-oriented
- Research approach: qualitative
  - based on video recordings
  - results are theoretical concepts of qualitative character
- Research strategy: single-case study
- Data generation method: non-participatory observation of natural behavior
- Data analysis method: parts of the Grounded Theory methodology according to the variant by Strauss and Corbin (Corbin and Strauss, 1996)

3.1 Refined Research Questions

In GTM, the research question is considered to lead the researcher to a specific field of interest for his analysis. During the analysis, this leading question is refined whenever it seems reasonable. (Corbin and Strauss, 1996)

During the data analysis of this study, the following six aspects of interest evolved. Two of these concern the effects of the awareness issues, the other four concern the use and effects of editing freedom in eDPP:

1. *Reduced-Physical-Awareness*: How and when does the reduced physical awareness influence the eDPP process? This refers in particular to the reduced non-verbal communication capabilities like facial expressions, gesture and posture.
2. *Reduced-Workspace-Awareness*: How and when does the eDPP-specific reduced workspace awareness influence the eDPP process? This refers to the limited sharing and the limited workspace awareness information in eDPP.
3. *Viewing-Freedom*: When and how does the pair make use of independent viewing capabilities? This refers to the capability of concurrently viewing artifacts in the workspace.
4. *Editing-Freedom*: When and how does the pair make use of independent editing capabilities? This refers to the capability of concurrently editing artifacts in the workspace.
5. *Freedom-Positive-Effect*: When and how does making use of the viewing and editing freedom appear to help the process? This refers to the potential positive impacts when the programmers concurrently view or edit artifacts.
6. *Freedom-Negative-Effect*: When and how does making use of the viewing and editing freedom appear to harm the process? This refers to the potential negative impacts when the programmers concurrently view or edit artifacts.

3.2 Research Framework

"Software engineering involves real people in real environments. People create software, people maintain software, people evolve software. Accordingly, to truly understand software engineering, it is imperative to study people – software practitioners as they solve real software engineering problems in real environments." (Lethbridge et al., 2005)

The groundwork nature of this inquiry was discussed and illustrated in Figure 1.6, leading to the need of developing an initial understanding of the eDPP process, which is tackled by a single-case study. As has been explained in Section 1.4.5 "Distributed Pair Programming (DPP)", currently eDPP research mainly is of comparative character and still fails to explain its mixed results. This is why a research approach to *"explain 'how' or 'why' a given treatment or intervention necessarily worked (or not)"* (Yin, 2014) is needed, which is the focus of case studies.

3.2.1 Research Strategy: Single-Case Study

"Many of the field studies in software engineering tend to be exploratory in nature, because we are still gathering basic knowledge about the human factors surrounding software development and maintenance. As a result, a case study design is commonly used and the study results in a theory or model that can be tested later. As our knowledge base grows, we can employ designs that test these theories or models." (Lethbridge et al., 2005)

The groundwork nature of this work was discussed Section 1.3.2. Associated with that is the problem of *"not knowing what to look for"* and *"how to grasp"* something of unknown manifestation. Consequently, it was a conscious decision not to formulate aspiring goals like finding classifications, making comparisons, determining causal relationships, or measuring something in a not yet understood process. Instead, **this inquiry explores the eDPP process to develop an initial understanding of eDPP-specific human behavior**.

The innovative but green-field nature of this inquiry also means that relevant **eDPP phenomena, which are part of a complex social situation, cannot be determined in advance. They have to be gradually experienced by the researcher**. To get a feeling for relevant eDPP events and their context, they have to be experienced in their natural entirety. And this is exactly where case studies have their key strength – they focus on a holistic view of the examined instance. Their priority is on the deep (not broad) understanding of one case by examining it in its natural setting, including all its complex attributes and relations.

Apart from the fact that relevant factors or *"variables"* cannot be determined in advance, survey research as well as experiments would not satisfy the requirements of this inquiry. Even though, in different ways, both reduce the complexity of reality to certain aspects. Survey research is focused on systematically collecting specific, similar information from a large group of participants. It provides a broad overview of aspects that have been anticipated to be relevant by collecting subjective reflections of many participants. The questions are designed to uncover the aspects, but it is rather unlikely to discover deep or unforeseen context information. In general, survey research is appropriate for *"general information (including opinions) about process, product, personal knowledge etc."* (Lethbridge et al., 2005). Experiments aim at understanding cause and effect by controlling disturbing factors, usually not happening in a natural context. Here too, relevant aspects must be known and controlled, which for the goal set out for this inquiry is neither possible nor would it lead to the desired results. For research projects such as the present, Yin (2014) suggests case studies as an adequate research strategy. Although not necessarily limited to, they are suitable for inquiries in an exploratory research phase *"where the questions are broad, there is little background knowledge, and little data to comparatively analyze"* (Lethbridge et al., 2005). They are reasonably used *"in situations when (1) the main research questions are 'how' or 'why' questions; (2) a researcher has little or no control over behavioral events; and (3) the focus of study is a contemporary (as opposed to entirely historical) phenomenon"* (Yin, 2014). For this inquiry, all three conditions apply and will be discussed in the following.

3.2.1.1 (1) Type of research question:

The overall research question of this inquiry is 'How do the reduced workspace awareness and the independent editing capabilities impact the eDPP process?'(see section 1.3). There is no notable knowledge available in that regard as well as no sound data for comparison.

3.2.1.2 (2) Extent of control over behavioral events, and (3) degree of focus on contemporary as opposed to entirely historical events:

First of all, eDPP is a current phenomenon, not *"dealing with the 'dead' past"* (Yin, 2014), and there is no rationale to use archival artifacts such as documents or log files.

A possible approach concerning the task control would have been to specify the task the participants work on. In case of having multiple pairs, this would have led to a better comparability of the pairs. However, it was not clear what would be an appropriate task that leads to desired behavior, or what would be 'desired behavior'. Also, as mentioned in Section 1.3.2 "Tool-Related Contributions", participants were rare and there was no pool of participants available from which to chose a sample. Actually, there was only one pair of skilled participants. They contacted the *Saros* project and due to their skills and experience they were asked whether they would agree to be recorded while doing their work. No need was seen to further constrain their natural behavior, they were simply asked to work on a real programming task. As a result, **natural behavior of programmers doing their regular work in their natural environment was observed**. The observation spanned one week of work and, since the pair worked on a re-implementation of a module, it comprised all typical phases of development – requirements elicitation, design, development and testing.

Finally, **the observed work is multifaceted and representative for industrial eDPP**. With respect to the research objective, the missing task control is considered unproblematic, and even beneficial since

it is *"essential to conduct field studies, i.e., to study real practitioners as they solve real problems"* to enhance software engineering tools and practices (Lethbridge et al., 2005).

3.2.2 Unit of Analysis: Pair of Experienced eDPP Developers

The observed pair consists of two male German software developers. One of them worked in an application systems department of a public service broadcaster. He was the domain expert and is referred to as Dom in the following. The other pair member was an external senior developer and software architect, called *Arc* in the following, who was employed at a service contractor. Dom and *Arc* knew each other in person because *Arc* has worked on-site for the public service broadcaster and both have been working in the same office for about three months. Then, Dom moved to another place and started working in another location. Their joint work on a workflow automation application, which originally had been developed by Dom, began several months before the observed work. When they had worked on different code areas in this project, they afterwards kept each other in the loop about their changes. When they wanted to do pair programming or joint code walkthroughs, they used *Saros*.

They got in touch with the *Saros* project because they wanted to present *Saros* to other developers and to be prepared for the question about the encryption of the transferred data in *Saros*. This inquiry resulted in a continued dialog and, after non-disclosure agreements had been signed, the pair volunteered to be recorded for research purposes.

Topic of the actual recorded work is a module in the radio-data-operation project, which over time became more and more complicated and hard to maintain. This module is responsible for the automated preparation and publication of radio data (reports and such) on a server. For an overall review and design optimization of the module, *Dom* was supported by *Arc*. The concrete observed task was a refactoring or re-design of that module. The pair worked together on this task for several days. The recordings cover three days of this collaboration; resulting in about 16 hours of video material, split into 7 recordings between 0:42 hours and 2:01 hours length and one of 5:26 hours. The first sessions focus on transferring domain knowledge from Dom to *Arc*, performing a joint design review and discussion. After these sessions, the pair decided it would be more efficient to re-design and re-implement the whole module. Most of the observations stem from these sessions covering design, implementation, and testing episodes.

3.2.3 Data Collection: Non-Participatory Observation

The chosen data collection approach was the **non-participatory observation by means of video recording**. Audio, video, and screen were recorded for each of the participants. Afterwards, both were joined in one video, resulting in an overview of both participants as shown in Figure 3.1.

This was technically realized by

1. inviting each participant into a separate(!) *Adobe Connect*[93] web meeting with the researcher at the other end
2. requesting each participant to share his screen, webcam, and audio
3. arranging the two web browsers showing these web meetings one above the other on a single large wide quad high definition (WQHD) (2560 $\times$ 1440 pixels) portrait-view monitor
4. locally recording the whole WQHD screen with *Camtasia Studio*[94]

The participants verbally communicated via *Skype*, saw their own desktop and *Eclipse* IDE enriched with the awareness information provided by *Saros*. They **chose not to use *Skype*'s integrated video** because they found it more distracting than useful.

[93] `http://www.adobe.com/products/adobeconnect.html` (accessed November 11, 2017)
[94] `http://www.techsmith.com/` (accessed November 11, 2017)

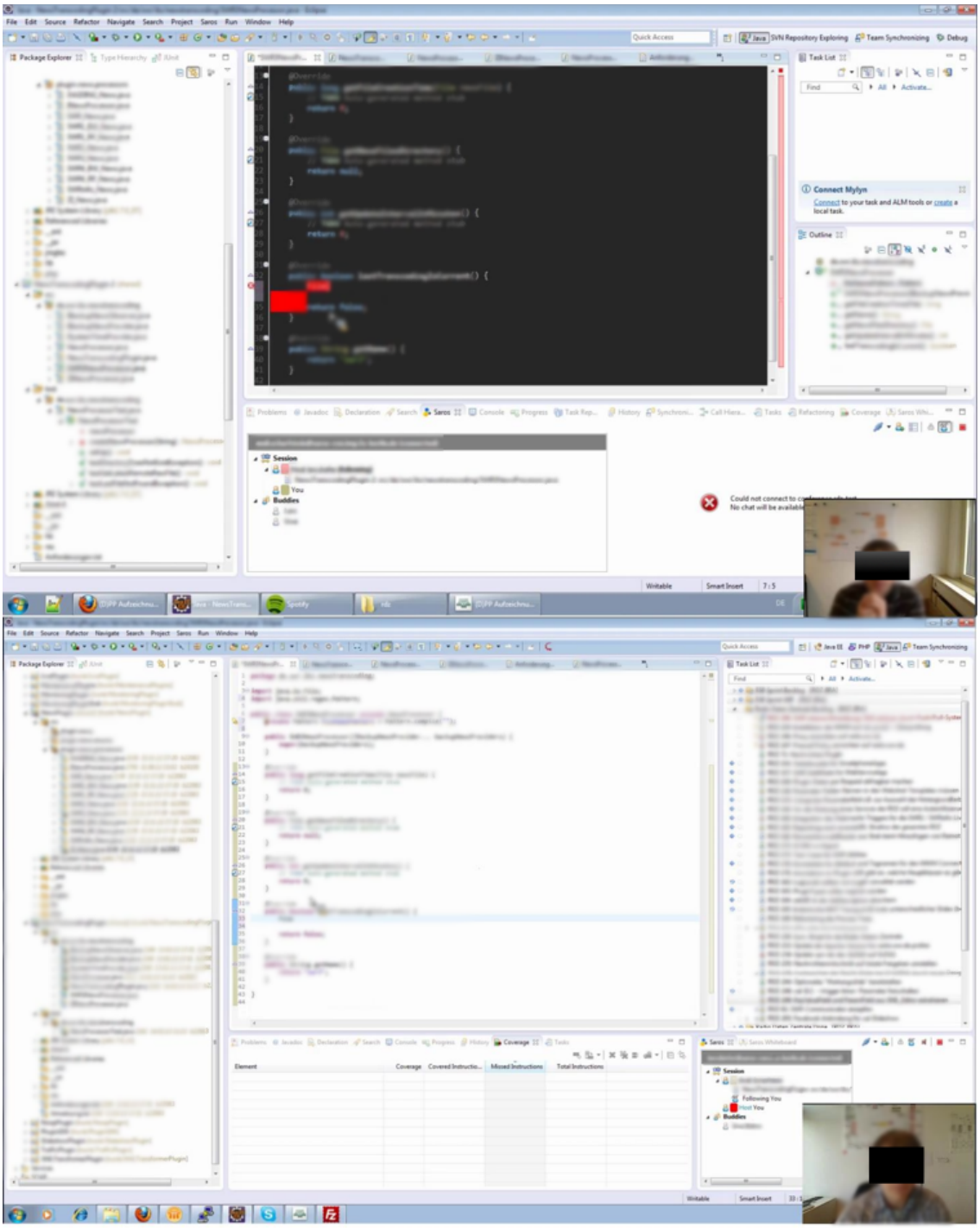

Figure 3.1: Consolidated video of both participants' screen and video.

The researcher had a more comprehensive view of the situation: She heard the participants' conversation, saw everything happening on both recorded screens of the pair members, and saw a small webcam view from atop the participants' monitors.

The **non-participatory observation was combined with a retrospective** (via *Skype* conference) after the recordings and post interviews (via e-mail) after some part of data analysis. As a kind of *"communicative validation strategy"* (Strübing, 2008) (translation from German by the author), a **member check** was performed when the participants reviewed a paper with initial research results.

3.2.4 Data Analysis: Grounded Theory Methodology

For the data analysis, parts of GTM were used. Stemming from the social sciences, GTM has become a practice in software engineering research, too:

"we use a qualitative research approach called grounded theory [20] to preserve the complexity of our case study data. The intent is to generate or discover a theory that is 'grounded' in data from the field. This approach is suitable for our study since we were not investigating any preconceived theories of what might make a GSD XP team successful or unsuccessful. In this research, we do not provide formal hypothesis testing or draw any general conclusions. However, we conjecture about some communication practices for GSD XP teams based upon evidence gathered from this case study." (Layman et al., 2006)

The goal of this inquiry is of the same nature: conjecture about the potential and issues of eDPP based on actual observed phenomena in the data.

This section will first explain the necessary concepts of GTM and when it is possible to use only parts of it, and then discuss why GTM is considered to be appropriate for this inquiry and how it has been applied.

The historical evolution of GTM was not direct and uniform. Due to the different epistemological backgrounds of its forefathers Anselm L. Strauss and Barney G. Glaser, in the late seventies two main versions of GTM have emerged. Both Glaser and Strauss share the belief that the main focus of the research products is their applicability to reality and that this can only be achieved by a systematic, data-grounded procedure (Strübing, 2008). Strauss and Corbin concretize many process aspects that are only vaguely described by Glaser (Salinger, 2013). One main aspect in this regard is that, for Strauss and Corbin as opposed to Glaser, the verification is an integral part of the theory generation (the research process). They also call for an intense description of the observed phenomena and explicitly address dealing with prior knowledge of the researcher. For a more differentiated discussion of the two GTM variants Salinger (2013, Chapter 3) is recommended.

Due to these central aspects and the more comprehensive and consistent character of GTM by Strauss and Corbin (Salinger, 2013), this variant of GTM was applied in this study. The following is a simplified depiction of this GTM version, focusing on on the central aspects that are necessary to understand its usage in the context of this work.

The GTM is not only a research method, it also specifies the kind of research product, namely a grounded theory (Strübing, 2008). It can be considered as a *"style of research to develop theories grounded in empirical data"* and it *"became the most widely used method in qualitative-interpretive social research"* (Strübing, 2008).

The methodology does not dictate strict analysis steps with detailed instructions. Rather, it is a *"systematic set of techniques"* (Corbin and Strauss, 1996) which have proven to be successful for investigating certain sociological issues. They expedite to get an explorative and open-minded access to the data in a structured way.

Applying GTM implies **accessing the material by the conceptualization of data segments**, this means in effect that relevant attributes are extracted and subsumed in a named concept. In this way, the raw data is conceptualized and relevant information are preserved. Instances of these concepts

are constantly compared where their similarity and dissimilarity is permanently challenged. Another comparison heuristic are far-fetched comparisons, where comparisons are made which at a first sight may seem absurd, like for example the question *"How is a priest like a prostitute?"* (Gerson, 1991). Through the constant comparison concepts come to maturity – their facets are discovered and adjusted, and concepts are split up or classified to 'higher level' concepts, so-called categories. (Salinger, 2013; Strübing, 2008)

The goal of this procedure is to gain **conceptual representativity**, that means having an adequate comprehensive and detailed characterization of these concepts (Strübing, 2008). Throughout the research process, all (preliminary) insights and thoughts of the researcher should be recorded in so-called memos so that they last and are available during the whole process and afterwards (Salinger, 2013).

The conceptualization of data is approached in a three-step iterative (non linear!) coding process where each coding phase focuses on different aspects. One must alternate between these three perspectives when looking at the data.

1. **Open coding** breaks up the data into **unconnected concepts and categories**. It is an analytical extraction of isolated phenomena in order to develop a broad access to the data. For the researcher, it is a kind of getting into the spirit of the data. Interesting segments are identified and conceptually named (Salinger, 2013). Identified segments are compared to each other in order to properly subsume them in concepts (Salinger, 2013). The elaboration of a concept also involves the elaboration of its properties or attributes. An attribute in turn can have different values or manifestations. For example, the attribute 'duration' can range from short to long and each observed phenomenon has a specific duration (value/manifestation) on this continuum (Strübing, 2008). Since the attribute's possible values are discovered and arranged on a continuum, this activity is referred to as 'to dimension' (Corbin and Strauss, 1996). This keeps the phenomena's substance and helps to develop an analytical variety instead of reducing it through consolidation (Strübing, 2008).

2. **Axial coding** aims at **understanding the relationships between the concepts**. Subject of this analysis step is still the evolution of concepts and categories, but with a different, more integrative focus. A phenomenon is treated along the axis of a general action model (Salinger, 2013), called the *"coding paradigm"* (Corbin and Strauss, 1996). Here the interactional and contextual conditions of a phenomenon are worked out by considering the five aspects of the coding paradigm:

   (1) The actual observed phenomenon/event of interest.

   (2) Conditions/events causing this phenomenon.

   (3) Strategies/actions that the participants use to deal with the phenomenon and the results of these strategies/actions (Corbin and Strauss, 1996).

   (4) The context of a phenomenon, the concrete situative conditions which help to understand the phenomenon (Strübing, 2008).

   (5) Intervening conditions like socio-structural, economic, historic, and other factors which shape the general, wider context of the phenomenon. These factors affect the strategies/actions of the participants for handling the phenomenon but they are no direct properties of the considered phenomenon (Salinger, 2013; Strübing, 2008).

3. **Selective coding**: Based on the open and axial coding, one category evolves to be the most remarkable and most promising with regard to the research question. In this step, this central phenomenon is selected and interrelated with other categories. The development of the phenomenon and its relationships are refined and validated in the data. Based on this, a story about the central phenomenon is built that is a detailed, grounded and conceptually comprehensible reflection of the real world – it is the theory. (Corbin and Strauss, 1996)

The GTM procedure described above is an **inductive-deductive alternating mental approach**: Concrete phenomena are discovered, compared, and conceptualized (inductive step). These conceptualizations (generalizations) and claims concerning their interrelationships in turn are tested in the concrete data (deductive reflection). (Salinger, 2013)

This interplay of building concepts and verifying them in the concrete data ensures that the interpretations and thus the produced theory are grounded in the data (Corbin and Strauss, 1996).

The goal of GTM is **neither to verify a theory nor to treat a theory as final knowledge** (Glaser and Strauss, 1967). The key aspect is that **a grounded theory is considered to be a temporary product** matured from the research process that in the moment of its formulation is the base for new theorization (Strübing, 2008): *"The published word is not the final one, but only a pause in the never-ending process of generating theory"* (Glaser and Strauss, 1967).

Accordingly, the fundamental attitude of this research style is not to prove a pre-formulated theory but to provide explanations by having an understanding of the actual facts. By refraining from a pre-defined theory, **it allows whatever is relevant for a research subject to come to the surface**. (Strübing, 2008)

This explicit desire of discovery also affects the way of sampling in GTM research. The so-called **theoretical sampling** satisfies the underlying idea that only the emerging theory, the gradual understanding and therefore the data itself, can reasonably direct what kind of data is useful for further theory generation. More precisely, the sampling in GTM research is perpetual and stepwise. (Corbin and Strauss, 1996; Salinger, 2013) Additional data is collected as material for further comparison and to encourage the (further) development of concepts. New insights and growing awareness of the researcher for certain aspects do not necessarily mean that new data is needed. Existing data can be re-coded with this enhanced awareness (Corbin and Strauss, 1996). For the initial data collection, the researcher has to count on theoretical considerations or practical experience (Strübing, 2008).

When sampling is not an initial, finalized step but an ongoing activity, the outstanding question is where to stop with data elicitation. This is addressed by the so-called **theoretical saturation**. It means that a category with its variations and relations is well elaborated and validated so that additional data is not expected to generate new insights and further development of the category (Corbin and Strauss, 1996). A theoretically saturated category provides an adequate (precise enough) answer to the research question and can be used by other researchers (Salinger, 2013).

As mentioned at the beginning of this section, the above merely is a rough overview about GTM. It does not address how GTM deals with existing knowledge, bias, and needed creativity of the researcher, nor does it provide a detailed description of the procedure and its potential hurdles. A more detailed discussion and application guide can be found in Corbin and Strauss (1996) and Strübing (2008). Salinger (2013, Chapter 3) is highly recommended as a summary of the historical development and the varieties of GTM, its central aspects, as well as its practical application.

Due to the systematic and explorative character of GTM, it is particularly suitable for unexplored research areas where concepts with their properties and relations do not yet exist but have to be developed by an open but structured and generative approach (Salinger, 2013; Strübing, 2008).

Since this work aims at developing an initial understanding of eDPP phenomena, and neither the eDPP process nor relevant eDPP phenomena are understood, it is greenfield research of explorative character. This is why elements of GTM were used in this inquiry to explore the videos in a data-driven, structured way. Before the application of GTM in this study is detailed in Section 3.2.4.2 "Grounded Theory methodology: Usage in this Inquiry", the next Section 3.2.4.1 "Methodological Fitting" roughly discusses the adequacy of the other prevalent qualitative data analysis methods.

3.2.4.1 Methodological Fitting

This section discusses the appropriateness of the most common qualitative research strategies with regard to the objectives of this inquiry. The presented selection does not claim to be comprehensive, but it gives an actual overview of the central methods used in social science. Since the suitability of a method for a research endeavor is determined by the method's purpose and area of application and not by their procedure, the following overview discusses the suitability of the methods but does not provide a description of their implementation. A more comprehensive delineation of the methods and their implementations can be found in Mayring (2002).

- **Phenomenological analysis**: Like GTM, the phenomenological analysis is a way to discover phenomena and their relationships. The phenomenological analysis focuses on the perspective of the involved individuals, their essential interpretations and intentions in a situation. (Mayring, 2002)
- **Social-scientific hermeneutic paraphrase**: Aims at interpreting a situation from the position of the involved individuals. It is mainly suited for a detailed understanding of text material, i.e. transcribed interviews. (Mayring, 2002)
- **Psychoanalytic text interpretation**: This method aims at understanding human interaction from a psychological perspective. Psychoanalytical means are used to analyze situations and to reveal unintended hidden assumptions and the suppressed sense of individuals. (Mayring, 2002)
- **Qualitative content analysis**: Chunks material by applying pre-developed categories on data segments. Qualitative content analysis is comparable with open coding in GTM, but the development of the categories is considerably different: with regard to the research question, the level of abstraction of the categories and the criteria that qualify a text segment to be classified in a category are determined in advance. (Mayring, 2002)
- **Typological analysis**: Data segments which significantly represent the material are picked out and typed. The typing is based on pre-determined selection criteria that determine which fact qualifies a segment to be typed and which aspects of it are considered in detail. The remarkable types are described in detail, whereas the typecast is validated in the data. This approach enables to reduce a large amount of data to have individual detailed case descriptions. (Mayring, 2002)

According to the research question and goals, this inquiry does not aim at analyzing individuals' (explicit or latent) minds or motivations, nor should external influencing factors be uncovered. Thus, the first three research strategies do not constitute adequate candidates for this research project.

Both qualitative content analysis as well as typological analysis appear closely related to GTM. The above-mentioned greenfield character of this inquiry with the desired open-mindedness are contrary to the predefinition of a framework defining the categories or type selection criteria and dimensions in advance.

Due to its openness and adequacy for explorative research approaches, GTM was considered to be an especially suitable data analysis method for the present project.

3.2.4.2 Grounded Theory methodology: Usage in this Inquiry

This work does not aim at producing an actual grounded theory and thus did not use GTM in full.

Shortened versions of GTM are common and envisaged by Corbin and Strauss (1996). They are aware of the fact that GTM is an extremely time-consuming procedure. Its application is not trivial, and, according to (Glass et al., 2009), less than one percent of research in computer science is GTM-based. Depending on the desired results, it is absolutely legitimate to reduce the effort by cutting the extent of the research (Corbin and Strauss, 1996). This applies in particular when research does not aim at encouraging scientific theory progress but wants to tackle practical issues while benefitting from

the sensing, openness, and systematics of GTM (Strübing, 2008). **For the pure understanding of an issue, which involves analyzing and conceptualizing the data, open and axial coding are sufficient** (Salinger, 2013). The systematic usage of constant comparison and theoretical sampling is required to some degree, but the properties and the relationships of the categories and subcategories need not to be worked out down to the last detail (Corbin and Strauss, 1996).

This dissertation wants to understand a practical issue and so to unravel the impact of the reduced awareness and increased flexibility on the eDPP process. The videos were explored by applying the following elements of GTM:

- The theoretical sensitivity about the evident eDPP specifics of reduced awareness and increased flexibility and their potential risks was used to find plausible starting points for the initial focus of the data analysis.
- Open coding, constant comparison, memo-writing, and developing the concepts' properties and dimensions was done. Open coding represents the main concern of this research, which is to gain a feeling for relevant phenomena in the eDPP process.
- Axial coding with an adapted version of the paradigmatic model was used. Relationships between concepts were not examined.
- Selective coding was not applied: The final step of integrating all categories, selecting one central phenomenon, and building a story around it which at the end represents the theory has not been done.
- The results are presented in a story telling style.

The central concern of this inquiry, to conceptualize observable phenomena which are specific to eDPP, was done via open coding. Phenomenon types that relate to awareness or to concurrent editing were described in detail and their characteristic properties with their possible values were identified. As described in Section 1.3.1 "Overall Research Goal and My Contributions", this was in no way straightforward: it required to get a feeling for relevant phenomena and their relevant characteristics. It also implied to develop a sense of how missing awareness manifests or what its indicators are.

Axial coding or the paradigmatic model was adapted to the research goal and to the characteristics of eDPP phenomena: evident (empirically observable) and relevant context information was gathered. A comprehensive context model for eDPP phenomena has been developed, which is presented in Section 4.1 "Classification of Observed eDPP Phenomena". It describes which context information are relevant for understanding a phenomenon – its emergence, course, and local impact.

As described in Section 1.3.2 "Tool-Related Contributions", gaining tool maturity and consequently user/participant recruitment were quite demanding and time consuming and for a long time not crowned with success. Thus, theoretical sampling as described to happen in an ideal world with nearly unrestricted access to the field was not possible. Instead, it was restricted to the search for corresponding situations in the available material. Strauss and Corbin (Corbin and Strauss, 1996) are aware of this participant-availability problem: *"If theoretical sampling could only be done with field data, most researchers could not use this method"* (Corbin and Strauss, 1996, translation from German by the author). They consider it as absolutely legitimate to conduct theoretical sampling within existing data because it can be seen as a theoretical selection of parts that are realized to be important for the inquiry. This is also in line with the aspect of re-coding data due to an enhanced awareness of the researcher, which has been described for theoretical sampling on page 139. Theoretical sampling within the data was done intensively. The data was reviewed over and over again, and codings were adjusted according to the increasing understanding and awareness of eDPP phenomena. There was never a point where additional data appeared to be missing in order to further develop a concept.

The integrative treatment of the concepts happens in the selective coding phase. Accordingly, the results of an inquiry that does not fully perform this step cannot claim to satisfy a grounded theory (Corbin and Strauss, 1996). Such results are rather unconnected interpretations and conceptualizations of striking phenomena and their context. Their grounded analysis and detailed description enables

substantial practical understanding, and the GTM-based procedure makes it more likely to get founded, comprehensible results (Corbin and Strauss, 1996).

This applies to the results of this inquiry. As described in Section 1.3.1 "Overall Research Goal and My Contributions", the identified phenomena reflect local events and their context. The result's description was inspired by the selective coding in so far as that for the most striking phenomena types/concepts an image of the real-world is narratively described, which is conceptual, comprehensible and grounded, as demanded by Corbin and Strauss (1996).

As noted earlier, the research design was subject to time, tool, and participants restrictions: Initially, the required tool did not work reliably, and (professional) users were hard to find. Study participants were quite limited, and tasks were not freely selectable. These aspects are addressed in the next section about the study's limitations and threats to validity.

3.3 Limitations and Threats to Validity

For GTM-based inquiries *"the usual canons of 'good science' should be retained, but require redefinition in order to fit the realities of qualitative research and the complexities of social phenomena"* (Corbin and Strauss, 1990). This redefinition implies that the research results must be discussed with regard to the purpose of GTM research in general and the concrete goals of a study (Strübing, 2008).

First, the purpose of GTM research in general and the concrete goals of the actual empirical study are recalled:

- **Characterization of GTM**: GTM in particular is *"designed to develop a well integrated set of concepts that provide a thorough theoretical explanation of social phenomena under study. A grounded theory should explain as well as describe. It may also implicitly give some degree of predictability, but only with regard to specific conditions"* (Corbin and Strauss, 1990).
- **Theoretical research goal**: As previously addressed, the goal of this inquiry is not to shape a theory but to develop an initial understanding of how the reduced workspace awareness and independent editing capabilities impact the eDPP process. This initial understanding should be achieved by grasping and characterizing evident eDPP-specific phenomena.
- **Practical utility of study**: From the practical point of view, this work is part of the goal of giving advice of how to make sensible use of eDPP. In general, CSCW research widely expects failures/big problems of eDPP, wondering whether applying an agile practice like PP to a distributed setting is possible at all (Schümmer and Lukosch, 2009), and in that sense this work serves as a feasibility study.

Generalizability in GTM is reached by the level of abstraction of the concepts as well as by the description of the conditions in which a phenomenon occurred. Both, the abstraction level as well as the completeness and integrity of the specified context, depend on the systematics and variety of the theoretical sampling. The more data was (systematically) sighted, the more phenomena and conditions have been observed which could be subsumed in concepts. In this way, the abstraction level, i.e. the range of applicability raises. The detailed context descriptions allow the assessment of the concepts' transferability to other situations, which is generalizability in the GTM sense. (Corbin and Strauss, 1990, 1996)

Based on the study's purpose and the general quality criteria for GTM research, the following paragraphs address the limitations of this study.

The non-existent variety of pairs in this study is a crucial factor regarding the completeness and generalizability of the found concepts. The observed participants were strong software developers with good communication skills being familiar with each other. Assumingly, these are necessary conditions for the success of eDPP. It is not clear what sufficient conditions might be or how common the above

conditions are, so **not much may be said about the situational generalizability of the observed phenomena**. The task, domain, participants' profiles, and further conditions are detailed in Section 3.2.2 "Unit of Analysis: Pair of Experienced eDPP Developers", so **the congruities and deviations to other settings can be assessed**.

The developed concepts represent an (incomplete) set of phenomena that *can* happen during professional eDPP. In light of the goal of developing an *initial* understanding of potential eDPP-specific phenomena, this is still satisfactory. As will be seen in Section 4 "Results", they can also be taken as evidence for the feasibility of professional eDPP.

"In grounded theory, representativeness of concepts, not of persons, is crucial" and not the generalizability of *"findings to a broader population per se"* (Corbin and Strauss, 1990). Theoretical sampling in GTM does not pertain to the representativity of the population under study but to conceptual representativity. This in turn denotes the completeness and developmental maturity of the properties and dimensions of the relevant concepts and categories. (Corbin and Strauss, 1996)

The recorded data comprises about one week of work of the pair, doing their regular tasks, passing typical software development phases like design, implementation, and testing by using a state-of-the-art eDPP tool. Accordingly, **apart from the aspect discussed above, that the data origins from only one pair, the data yielded a variety of phenomena, providing a rich base to develop the found conceptual abstractions**.

An exception are the concepts for real-parallel editing presented in Section 4.6.3 "eDPP Phenomenon: Parallelization", where both participants typed simultaneously. Here, the phenomena could not consistently be found. Together they cover only quite a small fraction of the time of the overall eDPP session. For these concepts, theoretical saturation (see page 139) could not be reached and their overall importance or process influence should not be overrated. Here the absence of problems during these parallel typing episodes is considered as an unexpected, positive insight. It provides a most interesting basis for further research, for example for examining optimization potential of the PP/eDPP process by real parallel work for trivial tasks.

The conceptualizations are so far fairly local. The described phenomena are short episodes and do not yet describe the whole solution process. Accordingly, the overall role that the phenomena play in the pair's process as a whole is hardly understood so far.

Moreover, it is conceivable that differences exist between local PP and eDPP outside the realms of awareness and editing freedom. The study was not designed to detect such differences.

The limitations of the *Saros* tool do not add to the limitations of the study. Certainly, a tool's functioning determines its usage. *Saros* can be considered as state-of-the-art for eDPP and is non-invasive in a sense that it does not come to the fore interrupting the users' work. From a theoretical perspective, additional functionality to better support nonverbal communication like facial expressions (through video) and gesture (through tele-pointer) is conceivable, so from that perspective *Saros* is still imperfect. The implementation of tele-pointers, however, is non-trivial and has a big potential of causing confusion. The observed users intuitively used *Saros'* Remote Caret and Remote Selection features for these purposes.

Another conceivable improvement regarding nonverbal communication would be an integrated video chat. *Saros* does not have this functionality but pairs could easily switch on the well working video chat of for example *Skype*[95] or *Google Hangouts*[96] when using it for their verbal communication anyway. The observed pair decided not to do so (see Section 3.2.3 "Data Collection: Non-Participatory Observation") and, as discussed on page 75, research shows that pairs tend to prefer working without a video connection.

[95] `http://www.skype.com/` (accessed October 29, 2017)
[96] `https://plus.google.com/hangouts` (accessed October 29, 2017)

Chapter 4

Results

"A good theory will incorporate the social and the technical. Either alone is 'one hand clapping'."[97]

The previous chapters presented the context and the tool-related work that needed to be done for this research project. The development of *Saros* was necessary spadework, but **actually this inquiry is process-centric and not tool-centric**. The research goal is to develop an initial understanding of the actual usage of eDPP technology and thus the results are about social aspects: eDPP-specific behavioral phenomena of the developers.

Chapter Structure – *What to expect in this chapter and why*

Chapter 4 presents the results of this inquiry in terms of behavioral phenomena observed during the eDPP process. Before the types of behavioral phenomena are presented, their attributes are introduced. Attributes are concepts which emerged as being relevant context information of a phenomenon. In the subsequent presentation of the results, each phenomenon type is discussed regarding its potential positive and problematic effect for the eDPP process, in particular with regard to the ideas of PP. The chapter closes with a general discussion of the results and their significance for eDPP.

[97] Jim D. Herbsleb, Keynote: *"Socio-technical coordination."*at the 36th International Conference on Software Engineering (ICSE 2014) `http://herbsleb.org/web-pres/slides/ICSE-keynote-2014-v18-dist.pdf` (accessed November 11, 2017)

4.1 Classification of Observed eDPP Phenomena

As shown in Figure 4.1, the observed eDPP phenomena either relate to the work mode of the participants or to the participants' roles:

- **Role**: The possibility of equal control of input devices in eDPP changes the definition of the classical PP roles.
- **Interleaved work mode**: When the participants make use of their increased flexibility by undertaking tightly interleaved viewing or editing operations.
- **Separated work mode**: When the participants execute activities or view things separately from each other. This can be tool-induced (things like dialogs that are technically not shared) or by choice of the participants. This work mode refers to the execution of activities, and does not necessarily mean that the participants also mentally split.

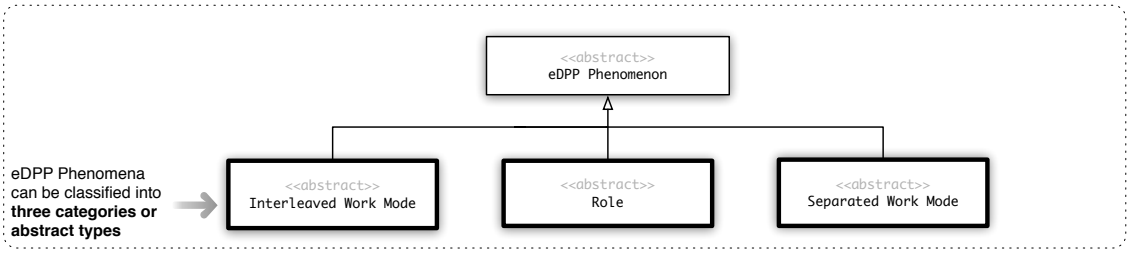

Figure 4.1: Result categories

The results relate to one of the following categories: role, interleaved work mode, or separated work mode

Results relating to the participants' roles are descriptions of the eDPP-specific variations of the PP roles that have been observed.

Results relating to the participants' work mode represent behavioral/interaction phenomena during the eDPP process and are further characterized by their attributes which are relevant to understand the process context of the phenomenon – its emergence, course, and impact.

An attribute (specifying concept) can either be of simple or complex nature. A simple attribute is context information that can be adequately expressed with one term, like for example the phase of development in which a phenomenon occurs (analysis, design, implementation, testing, or maintenance). In this context, adequacy means that it provides a sufficiently detailed understanding of the contextual aspect, but does not mean that the concept in general could not be further specified.

Complex attributes are compound aspects of a phenomenon's context which cannot be adequately expressed in one dimension but have to to be inferred from a composition of various aspects. An example of such a complex attribute is Coupling of Participants^{CAttr}, which includes the aspects *Visual Closeness*$^{CAttr\text{-}A}$, *Subject Awareness*$^{CAttr\text{-}A}$, and *Cognitive Affinity*$^{CAttr\text{-}A}$ (see Section 4.4.13 "Complex Attribute: Coupling of Participants").

Following the UML[98] notation, Figure 4.2 provides a grouped overview of all phenomena types and their attributes:

- All phenomena types are subsumed in the the abstract concept *eDPP Phenomenon*.
- They subdivide into three abstract types of results.
- Within these abstract types, the grey-shadowed boxes in the bottom layer of 4.2 show the concrete phenomena types. Such a concrete phenomenon type subsumes instances of observed phenomena with shared characteristics.

[98] unified modeling language

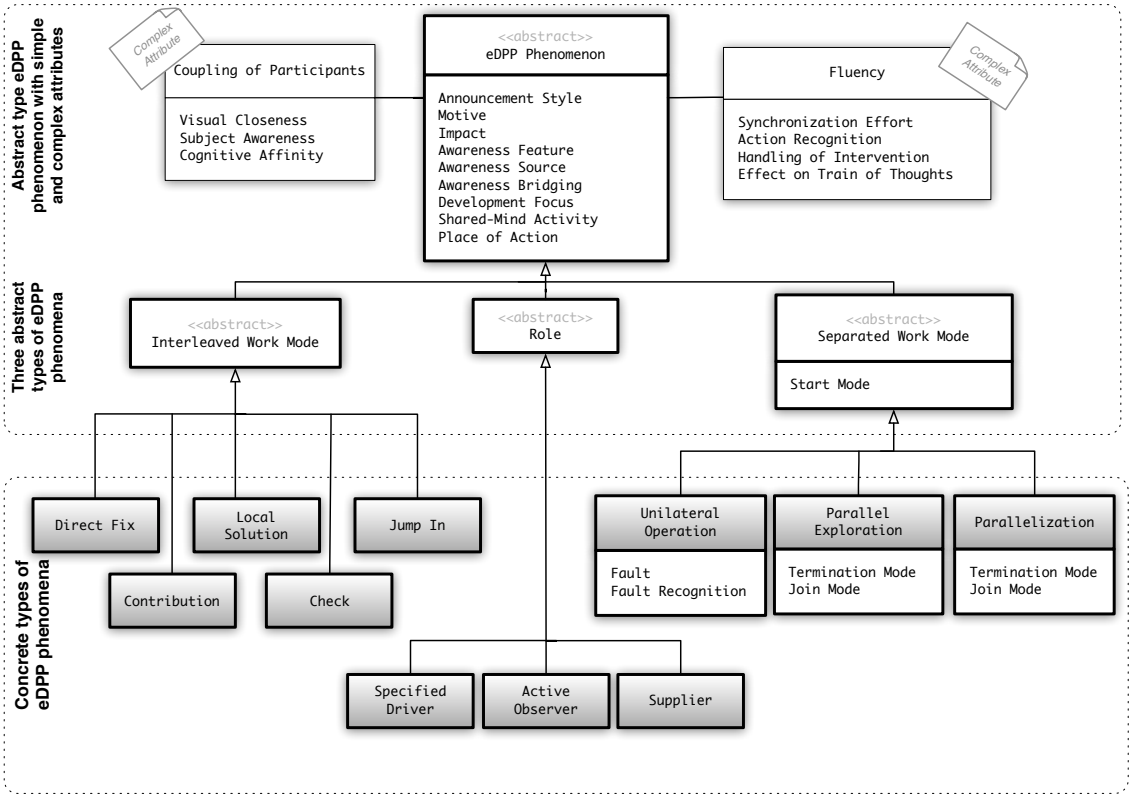

Figure 4.2: Abstract presentation of eDPP phenomena and their attributes

- Attributes applying to all phenomena types are noted in the abstract concept *eDPP Phenomenon*. Simple attributes are noted as attributes in the concept. Complex attributes are represented as associated concepts.
- Specific attributes only applying to specific phenomena types are noted in the concrete phenomenon type.

Each specialization of a phenomenon inherits the attributes of its generalization. If they do not apply, attributes must not be used for a concrete phenomenon type.

4.2 Presentation of Results

To provide a holistic picture of the observed phenomena types, the results are presented in a **narrative style which includes their discussion and interpretation**. For human behavior, its context is an integral, meaningful element, and for GTM research findings, the completeness of the context's description is an integral aspect of their generalizability (see Section 3.3 "Limitations and Threats to Validity" on page 142). Accordingly, a contextless, mere mentioning of the results was not considered to be an adequate form of presentation. At the end of this chapter, Section 4.7 "Discussion" reflects the research results on a more general level, addressing the potential benefits and drawbacks of eDPP.

To identify and distinguish concepts and attributes in the text, the notation is as follows:

- The phenomena types are typeset in small caps, for example Direct Fix.
- A simple attribute is typeset in small caps with the superscript 'SAttr', for example Development Focus$^{\text{SAttr}}$.

- A complex attribute is typeset in small caps with the superscript 'CAttr', for example Coupling of Participants^{CAttr}.
- The aspects of a complex attribute are typeset in italic with the superscript 'CAttr-A', for example *Visual Closeness*$^{CAttr\text{-}A}$.
- An attribute of an observed phenomenon has a specific manifestation or value (as discussed on page 138 in Section 3.2.4 "Data Analysis: Grounded Theory Methodology"). The various observed manifestations of an attribute are listed using the logical OR operator (||).

The recommended reading for the following sections is to start with the next two sections 4.3.1 "Specified Role: Driver" and 4.3.2 "New Role: Active Observer". They describe the findings regarding the roles in eDPP and require no further base concepts to be acquainted with. The characteristics of the roles are described, in particular with regard to their difference to the corresponding PP role. **The awareness of the roles in eDPP is necessary to understand the attributes of the work-mode-related phenomena**, which are described subsequently in Section 4.4 "Results Specifying Attributes of eDPP Phenomena". Maybe a reasonable approach is to only skim that section with the description of the attributes and to look them up later as occasion demands.

The phenomena's presentation starts in Section 4.5.1 "eDPP Phenomenon: Direct Fix" with work-mode-related behavioral phenomena.

Each presentation of a phenomenon type starts with an abstract description, which is succeeded by a description of its occurrence in the data. Afterwards, its variants of manifestation are presented. A discussion of the phenomenon type with regard to the research questions follows thereafter (see Section 3.1 "Refined Research Questions"). The presentation of each phenomenon type ends with discussing and conjecturing its possible positive and problematic behavior variations in the PP and eDPP context.

For observations from the data, past tense is used. Discussions of phenomena are kept in present tense. For theoretical reflections, the subjunctive in present tense is used.

4.3 Results Concerning Roles in eDPP

4.3.1 Specified Role: Driver

In a nutshell: Driver

In eDPP, the Driver predominantly, but not solely and exclusively, has the editing sovereignty.

Due to the independent workspaces with equal control of the input devices, **the classical PP roles cannot be mapped one-to-one to eDPP**. Nevertheless, they remain helpful notions.

An obvious issue with adopting the classical definitions of PP roles to eDPP is to determine the driver by being the person which controls the keyboard and mouse, as for example described in Williams et al. (2000). Linking the driver role in eDPP to the ownership of the input devices would lead to extremely frequent role switches (every time one of the participants makes use of one of his input devices) with a very different nature than role switches in PP. Even in PP, this simple definition can run up against its limits due to keyboard/mouse splits or in dual-keyboard/dual-mouse settings. In the eDPP setting, the situation gets worse: There is not only a second keyboard, but also a second cursor and screen.

Beyond that, as discussed in Section 1.1.1 "Pair Programming – a Common Agile Practice", the classical definition of roles in PP is less useful for understanding the nature of the process contributions of the respective roles. It does not satisfy the various sophisticated facets of the roles.

In eDPP, both participants own a physical keyboard and mouse. In the observed sessions, the pair made interleaved use of them although the mental role allocation (based on the PP roles regarding the type of cognitive focus, editing authority, and process driving) did not disappear.

Accordingly, the classical driver and navigator roles are adequate notions for role attribution in eDPP. Their associated concepts, activities and role connotations like the increased process and decision influence of the driver (as discussed in Section 1.4.1.3 "Pair Interaction in Pair Programming") remain a helpful basis.

It could be observed in the data that the DRIVER predominantly drove the editing process to solve the overall task. The observer, however, contributed some slave activities. The process driving of the DRIVER did not only involve pure editing activities but included breaks, discussions and manipulations of artifacts by using keyboard shortcuts, menu commands, and dialogs.

Accordingly, **the Driver role in eDPP refers to all activities for achieving the session's goal and does not mean continuous exclusive editing**. It is a variation of the PP driver role, taking the bilateral interaction capabilities of eDPP into account.

In PP, the observer can only support the DRIVER mentally or by looking up something on a detached laptop next to the shared machine. In eDPP, however, he can become physically active in the shared workspace. The nature of the observer's activity changes significantly compared to his possibilities in PP. As a result, the reason for adapting the definition of the driver role are not the activities of the DRIVER, but the observer's extended possibilities of contribution. This is why the DRIVER role is retained and a new eDPP-specific role, the ACTIVE OBSERVER, is introduced.

4.3.2 New Role: Active Observer

In a nutshell: Active Observer

The observer uses his viewing and editing freedom to become active in the shared workspace.

The observer in eDPP disposes of extended capabilities of interaction with the shared workspace. Indeed, the data showed that he occasionally made use of these capabilities to support the DRIVER's work. However, these activities did not aim at taking the leadership of the editing process but had supportive or amending character with regard to the DRIVER's work. For example, the observer used his independent editing possibility to concurrently fix misspellings in the code on his own without having to prompt the DRIVER to it. He also looked up something in another artifact or in his web browser which was relevant for the DRIVER's current work.

As a result, the observer became an ACTIVE OBSERVER, defined as someone who concurrently (in the sense of interleaved) operates his computer to provide support for the DRIVER – he is an **observer with an increased activity level**.

Activity Level of the Active Observer

The classical driver/observer role description implies that the observer's physical, workspace-related activity level is usually low — being in the position of observing and reviewing the driver's work.

Activity level refers to physical activity insofar as it contributes to the joint development process (speaking, pointing, typing, checking something, etc.). In that view, the observer can increase his activity level in eDPP by making use of his extended viewing and editing capabilities.

The Active Observer used his freedom to perform activities that he could not, or not unobtrusively, carry out in local PP. Consequently, each occurrence of the Active Observer is a manifestation of *Viewing-Freedom* or *Editing-Freedom*.

A possible consequence of the Active Observer's independent, interleaved viewing and editing is that the pair visually or mentally decouples. With respect to *Freedom-Negative-Effect* they potentially lose the beneficial effects of PP like continuous reviews, knowledge transfer, or shared problem solving. For the studied pair, this negative effect could not be observed. This, as well as the potential *Freedom-Positive-Effect* is discussed in more detail in Section 4.7.2 "Impact of Viewing and Editing Freedom".

4.4 Results Specifying Attributes of eDPP Phenomena

The attributes of eDPP phenomena refer to a phenomenon's context including aspects like the congruence of the participant's mental focus or the alignment of their activities with regard to the shared work goal. For this, a conceptual framework allowing to describe the respective aspects on an appropriate level of detail had to be found. For the purpose of this work, the following terms appeared to be adequate abstractions; some are explained using the definitions from Cooper et al. (2012) and Martin (2008).

- **Goal**: *"A goal is an expectation of an end condition, whereas both activities and tasks are intermediate steps (at different levels of organization) that help someone to reach a goal or set of goals"* (Cooper et al., 2012). For the scope at hand, further basic human needs or goals, like love, safety, rewards, and alike, are not considered. The goal of the observed collaborative programming sessions is to have a neatly designed module with a specific set of functionality.
- **Task** (segment of a goal): A goal or result is a logical unit that can be decomposed into several logical segments that have a logical coherence. They can be on different levels of abstraction, as long as they can meaningfully be considered as an abstraction and not an *"essential concept or a detail"* (Martin, 2008). A desired feature, for example, is a logical unit which is decomposed into several functions (Martin, 2008) that in turn are logical units as well but on a different level of abstraction. Each segment is tackled by a task.
- **Aspect** (part of a task): An aspect is a facet of a task. A task involves several aspects to consider. For example, an aspect of a task is to have test cases for the production code.
- **Step** (part of an aspect): The individual aspects are tackled via concrete steps that are taken. For example, writing a test involves the steps of creating a test class, maybe generating test data, and implementing the individual test methods.
- **Activity** (action behind a step): Activities occur in the course of performing a work step. This involves for example discussing, decision making, writing code, looking up things.
- **Solution Process** (process of the overall solution development): The goal drives the individual development activities since it is in the problem domain and needs a solution, which in turn is realized in the process. In the solution process, the activities and decisions solve a given problem.

4.4.1 Simple Attribute: Announcement Style

When one participant started an activity in the workspace, for example when the Active Observer became active or the Driver operated a unilaterally visible dialog, he either somehow announced his subsequent activity or just started acting right away.

The way **how a participant announced his activity** in the workspace is represented in the simple attribute Announcement Style^{SAttr}.

? Which question does it answer?

How does a participant announce that he will do something in the workspace?

The following list shows the manifestations of ANNOUNCEMENT STYLE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.3:

- **Explicit**: The participant explicitly announced his subsequent activity, either by announcing that he would do something, e.g. *"Oh, wait, may I just [...]"*, or by describing what he was going to do, e.g. *"I will look this up on Wikipedia [...]"* or *"Wait, I quickly create the interface"*.
- **Implicit**: The participant expressed his concern with an utterance, e.g. *"Brackets are missing"*, and usually started acting right away. He though did not explicitly mention that he would do it.
- **Mumble**: The participant uttered something under his breath. Often these utterances were no complete sentences and rather difficult to understand. Thus, they indicated that something was going on, but did not reveal much information.
- **Silent**: The participant simply started acting right-away without giving any verbal hints.

In particular in the latter case, this might confuse the DRIVER because he may wonder about what is suddenly going on in the workspace. The same applies if he had not noticed the action but might later recognize a changed artifact without understanding the reason for this.

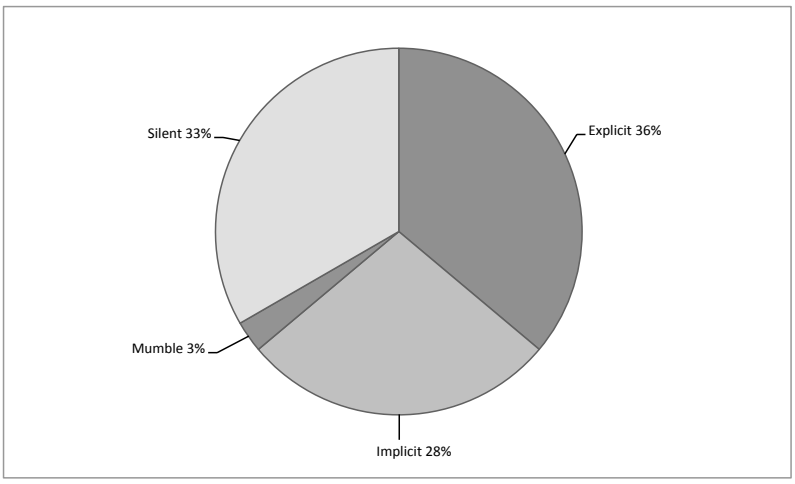

Figure 4.3: Proportional distribution of ANNOUNCEMENT STYLE^{SAttr} manifestations

4.4.2 Simple Attribute: Motive

In general, an eDPP-specific phenomenon occurred because of the only partially shared workspace or because of the independent workspace capabilities. In either case, there was a reason why one or both participants carried out a certain eDPP-specific activity: one or both participants saw something in the source code, or they recognized something in the shared solution process that motivated them to become active. Or it was a necessary activity in the current work step but could not be carried out in a joint manner, like for example operating a dialog.

MOTIVE^{SAttr} represents the cause for an eDPP phenomenon in the sense of the **underlying human motivation for performing a certain eDPP-specific activity**.

? Which question does it answer?

What was the product or process related trigger for one or both participants to undertake an eDPP-specific activity?

The following list shows the manifestations of MOTIVE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.4.

- **Clean Code Flaw**: The participant noticed some poorly written part of the code with regard to clean code (Martin, 2008) principles like comments, meaningful names of artifacts, formatting, artifact and function structure, or alike. He became active to improve the code in that regard.
- **Faced with Problem**: The DRIVER posed a question, or the pair was faced with a problem in their shared problem solving process. Then one or both participants used their independent exploring capabilities to search for a solution. In particular when the DRIVER formulated a problem, he often did not explicitly prompt the ACTIVE OBSERVER to become active, but the ACTIVE OBSERVER himself started to solve it. For example, the DRIVER raised the issue of how to write a test that circumvents the system-time dependency of the tested code, and the ACTIVE OBSERVER saw this as a call to look it up.
- **Create Artifact**: One participant created an artifact by using a dialog (unshared in *Saros*) when required.
- **Reassure**: The ACTIVE OBSERVER used his independent viewing freedom to look up something he was unsure about, for example because the DRIVER sought his approval and he wanted to reassure before giving his okay. Or the DRIVER considered a certain work step as finished, but the ACTIVE OBSERVER wanted to check that they had done the work correctly and completely. Thus, he looked up the required method declarations of an interface of a design pattern after the DRIVER declared the interface implementation as finished.
- **Realize Idea**: The ACTIVE OBSERVER had a thought relating to the current work thread or to the DRIVER's activity. He used his editing freedom to put that thought into practice in the source code. Such an intervention fitted in the currently accomplished work step of the pair. For example, the ACTIVE OBSERVER interrupted the DRIVER to note down a discussed insight in a comment before he let him continue his work. Or the ACTIVE OBSERVER had an idea regarding the next concrete implementation step which was to move a line of code from one block to another, and he did it right away.
- **Syntax Error**: The ACTIVE OBSERVER noticed a typo or syntax error in the source code and used his editing freedom to fix it.
- **End Session**: The participants were in a kind of end session mode and independently explored their artifacts to recapitulate their day's work.
- **Split Task**: The pair discussed the next tactical work step and split up to work in parallel on two trivial subtasks.

The underlying human MOTIVE^{SAttr} for an eDPP phenomenon is relevant to gain an understanding for what reasons eDPP-specificity is used and thus how it alters the work mode. In particular in combination with the process consequences of those actions, this allows insights about the problems and the potential of eDPP.

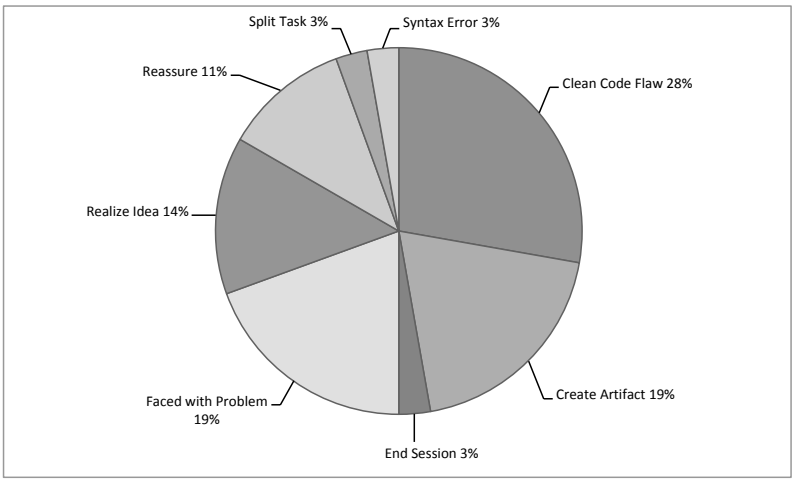

Figure 4.4: Proportional distribution of MOTIVE^{SAttr} manifestations

4.4.3 Simple Attribute: Impact

Since eDPP phenomena were part of the collaboration, they affected the process or product. Examples of such effects of eDPP phenomena were changes in the source code like a typo that has been fixed or episodes where the participants started a discussion to clarify an issue.

As illustrated in Figure 1.6, the process-wide, long-term analysis of eDPP phenomena is not part of this thesis. However, **the immediate observable impact or consequence of an eDPP phenomenon** is in the focus of this study and is expressed by the attribute IMPACT^{SAttr}. These consequences are described depending on the predominantly observable aspect.

? Which question does it answer?

What process or product related impact does an eDPP phenomenon have?

The following list shows the manifestations of IMPACT^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.5:

- **Clarification Episode**: An eDPP phenomenon triggered an episode where the participants sharpened their understanding regarding a requirement or an aspect of the problem domain.
- **Cleaner Code**: An eDPP phenomenon resulted in a code improvement with regard to clean code principles like meaningful comments or artifact names, or a consistent arrangement of methods in a class according to their access modifiers.
- **Step Forward**: An eDPP phenomenon contributed to the progress of the solution because it was the next work step. This contribution was achieved either through active editing or through the creation of artifacts.
- **Accelerate Next Step**: Such an eDPP phenomenon was of anticipating character and accelerated the collaboration process. While one participant was about to finish the current work step and this finishing only involved no-brainers, the other one already started working on a subsequent, directly connected activity. For example, while the DRIVER was about to finish a test case, the ACTIVE OBSERVER used his editing freedom to do the next step, which was to create files that were required by that test case.

- **Fixed Issue**: The result of such an eDPP-specific activity was that an issue in the code was fixed. This ranged from a simple fix of a typo up to solving content issues like identifying the reason for a failing test case.
- **Unilateral Verification**: The ACTIVE OBSERVER independently and without sharing his insights with the DRIVER, reassured himself regarding the correctness of the current activity of the DRIVER. This then was a unilateral verification of a work step.
- **Dialog**: An eDPP phenomenon triggered a dialog between the participants where they swapped ideas and thus developed or improved their awareness regarding a specific topic.
- **Reject**: The ACTIVE OBSERVER became active in the workspace and explicitly announced that, but was rejected by the DRIVER. As a consequence, the ACTIVE OBSERVER canceled his action.

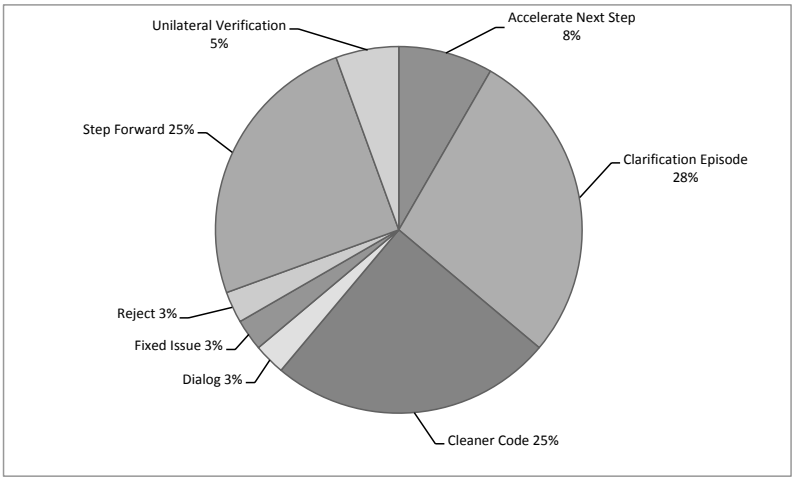

Figure 4.5: Proportional distribution of IMPACT^{SAttr} manifestations

IMPACT^{SAttr} helps to understand how eDPP-specific activities directly affect the collaboration.

4.4.4 Simple Attribute: Development Focus

During software development, the developers tackle different aspects of the overall goal like analysis, design, implementation, testing, etc. PP is a practice that can be applied to all aspects, and so far there is no evidence that this should be different for eDPP. The attribute DEVELOPMENT FOCUS^{SAttr} describes **which aspect the pair currently paid most attention to when an eDPP phenomenon occurred**.

? Which question does it answer?

Which aspect is the pair predominantly concerned with when a phenomenon happens?

In general, the following aspects are addressed during the software development lifecycle:

1. **Analysis**: The requirements for a software product are gathered from the customer or user. They are defined and analyzed for consistency. Outcome of this stage is that the team knows what the customer wants or what the user needs (which in many cases is not the same).
2. **Design**: The requirements of the analysis are transformed into a conceptual model of a system, reflecting how to realize the requirements. The team plans the overall structure of the system,

its components and how they interact. This can range from deciding on an macro (overall) architecture to specifying the system's micro architecture (its modules of different granularity).

3. **Implementation**: On this level, the actual coding takes place – modules are realized in source code. Discussions and decisions are made, like which design patterns to use, which classes to create, what operations and attributes each class should provide, as well as problems solved on program logic (algorithm) level.

4. **Testing**: The produced code is tested for defects. This includes performing quality assurance activities on different levels and with different approaches – ranging from writing unit tests to executing integration and system tests.

5. **Maintenance**: After finishing and operating the initial product, the software is improved, upgraded or extended. This again includes activities from the previous phases. At this level, aspects like software quality and code maintainability are relevant. In particular the latter is addressed by approaches like clean code software development[99].

The following list shows the manifestations of DEVELOPMENT FOCUS^{SAttr} that have been seen in the data. Their proportional distribution is visualized in Figure 4.6.

- **Implementation**: The pair actively worked on implementing the problem solution. They actually wrote production code and solved problems on program logic level: they discussed and implemented algorithms and data structures with regard to good code quality.
- **Local Design**: As part of the implementation, the pair discussed and decided on local design aspects: design patterns to use as well as objects, their attributes and relations.
- **Testing**: The pair created and ran test cases.

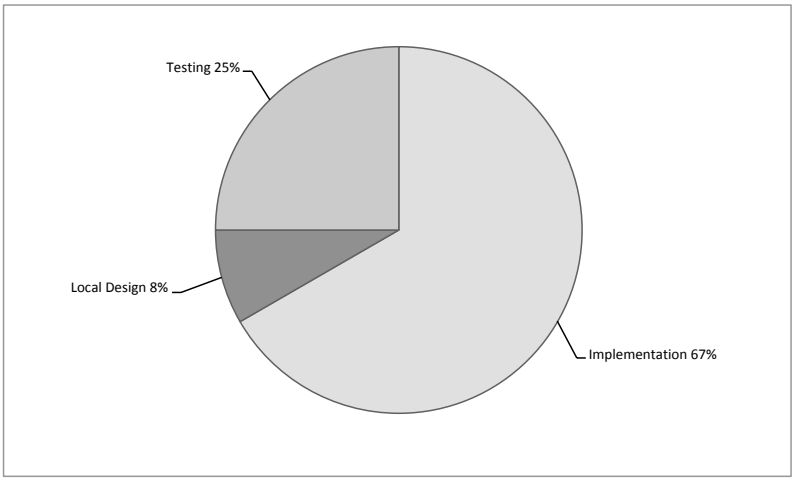

Figure 4.6: Proportional distribution of DEVELOPMENT FOCUS^{SAttr} manifestations

DEVELOPMENT FOCUS^{SAttr} helps to consider a phenomenon in the bigger context of the development life cycle. This enables a more meaningful understanding of the phenomenon and its criticality or potential during certain development foci. The goal of the observed pair was to re-implement a module. An interesting starting point for further research would be to observe other eDPP pairs with other goals that rather focus on other aspects of development. This could reveal interesting insights about the suitability of eDPP tools, which currently primarily focus on shared editing. Other types of activities that are currently rather poorly supported in eDPP tools are joint sketching, debugging, searching and understanding, refactoring, and alike.

[99] `http://cleancoders.com/` (accessed November 11, 2017)

4.4.5 Simple Attribute: Shared-Mind Activity

Pair work involves several mental activities. The attribute Shared-Mind Activity^{SAttr} describes the **type of mental activity the pair was currently concerned with** when an eDPP phenomenon occurred.

? Which question does it answer?

Which type of mental activity does the pair concentrate on?

The following list shows the manifestations of Shared-Mind Activity^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.7.

- **Solution Development**: The participants were concerned with developing the solution. They advanced the solution by writing code. This also involved arguing and challenging ideas or alternatives, and making decisions relating to the current work step. For example, this referred to details of how to implement a specific aspect as well as to which methods and classes to create. Solution Development does not involve fundamental discussions nor overall design decisions.
- **Write Test**: The participants created a test case. This comprised writing the actual test code as well as creating the test data.
- **Clarification**: A clarification episode started when the pair was faced with an issue or when they recognized that they lack knowledge regarding an aspect of the problem domain or the functionality of their IDE. This was done for example by visiting the requirements specification or by investigating an existing code excerpt regarding its applicability for the current problem.
- **Decide Artifact Structure**: The pair contemplated local design decisions like the usage of a design pattern or the hierarchy (inheritance structure) of artifacts.
- **Defect Localization**: When a failed test case indicated a defect in the code, this led to an episode where the pair localized that defect.
- **Reflect Result**: The pair summarized the state of their interim work result when ending their current work session to be continued the next day.

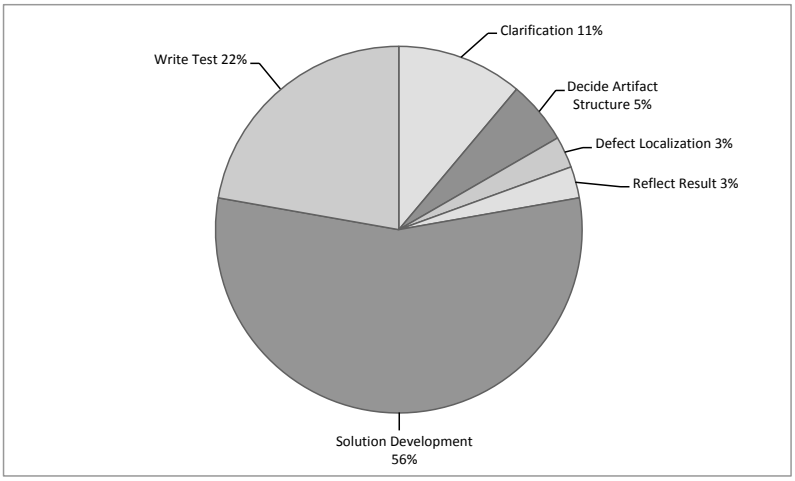

Figure 4.7: Proportional distribution of Shared-Mind Activity^{SAttr} manifestations

Shared-Mind Activity^{SAttr} breaks down the mental activities of the participants with a certain Development Focus^{SAttr}. This helps to complete the context of an eDPP phenomenon. Its potential

and the criticality of potential downsides like mental decoupling or loss of continuous review effect can be assessed more holistically.

4.4.6 Simple Attribute: Place of Action

eDPP phenomena occurred in a specific focused area in the workspace which was either shared or unshared. The participants also had their focus outside the IDE, for example in their web browser or terminal.

The attribute PLACE OF ACTION^{SAttr} describes **the workspace area where the phenomenon or the relating activities happened**.

? Which question does it answer?

Which workspace area is in focus during an eDPP phenomenon?

As could be seen in Figure 2.3, the *Eclipse* IDE is structured in different areas, each displaying a certain set of related data or information. These so-called views may be visible or not and can be freely arranged. For example, one can configure the workspace to see the editor containing the source code in the center of the IDE and arrange the file browser at the left side, a todo-list on the right side and below the output of the console. In *Eclipse*, shared and unshared code artifacts are shown in the editor view. Apart from the different (unshared) views in *Eclipse*, the developers may use IDE features via dialogs and wizards which are not shared in *Saros*, either.

The following list shows the manifestations of PLACE OF ACTION^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.8.

- **Editor**: The eDPP-specific activities took place in the editor view of *Eclipse*. To which degree the participants had the same focus or viewport is captured in the aspect *Visual Closeness*$^{CAttr\text{-}A}$ of the complex attribute COUPLING OF PARTICIPANTS^{CAttr} (see Section 4.4.13 "Complex Attribute: Coupling of Participants").
- **Dialog**: A dialog or wizard within *Eclipse* was used. Accordingly, this refers to an unshared part of the workspace.
- **Web Browser**: The eDPP phenomenon involved the (unshared) web browser of one or both participants.
- **Context Menu**: The context menu of *Eclipse* was used. Context menus appear when hovering or right-clicking on an element or code segment in the workspace, and are not shared.
- **Terminal**: The participants used their unshared terminal.

Information about the focused area or viewport of an eDPP phenomenon allows conclusions regarding the availability of awareness information and the potential helpfulness of awareness features. It also is interesting with respect to coordination or awareness-bridging mechanisms of the participants.

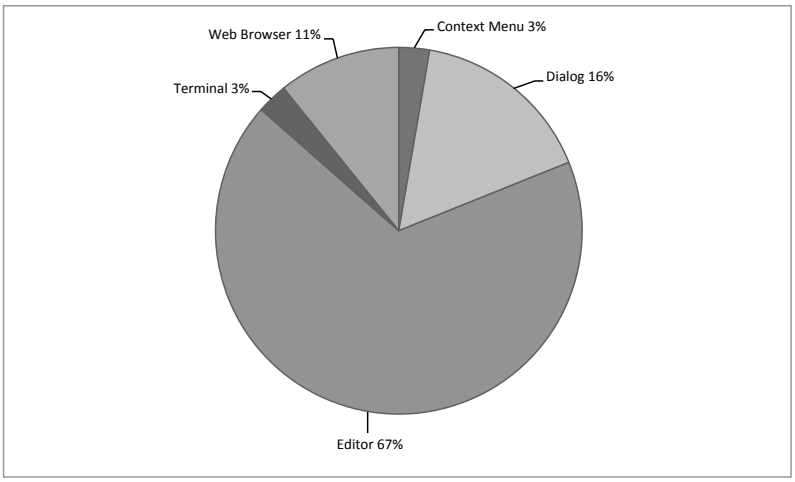

Figure 4.8: Proportional distribution of PLACE OF ACTION^{SAttr} manifestations

4.4.7 Simple Attribute: Awareness Feature

Most of the time the pair used *Saros*' *Follow Mode* to stay visually coupled in the workspace. They also moved separately in their workspaces and thus decoupled visually from time to time. This happened intentionally, for example when the participants decided to explore separately in the workspace. Or unintentionally, when one participant got lost in thought and lost track of his partner's location and activities.

The **usage of *Saros*' *Follow Mode* during an eDPP phenomenon** is noted in the attribute AWARENESS FEATURE^{SAttr}.

? Which question does it answer?

How do the participants use tool support to stay visually coupled?

The following list shows the manifestations of AWARENESS FEATURE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.9.

- **Follow Mode Used**: The *Follow Mode* was enabled when the phenomenon occurred.
- **Follow Mode Disabled**: The *Follow Mode* was disabled during the happening of the phenomenon.
- **Follow Mode Stopped**: The *Follow Mode* stopped during the eDPP phenomenon. In *Saros*, this happens because the *Follow Mode* automatically stops when the follower moves to another artifact than the followee.
- **Follow Mode Manually Stopped**: When the follower wanted to independently move within the same or another artifact as the persecutee, he manually stopped the *Follow Mode*, because it is not possible to independently move within the same artifact with the *Follow Mode* enabled. When the following person starts to scroll to another position, the *Follow Mode* always puts him back to the current viewport of the followee.
- **Jump-To-Position**: In the course of an eDPP phenomenon, a participant made use of *Saros*' *Jump-To-Position* feature.

The visual decoupling can be avoided by using *Saros*' *Follow Mode*. As a variant of the shared browsing pattern (see page 90), this feature helps to automatically follow and see the other one's activities in

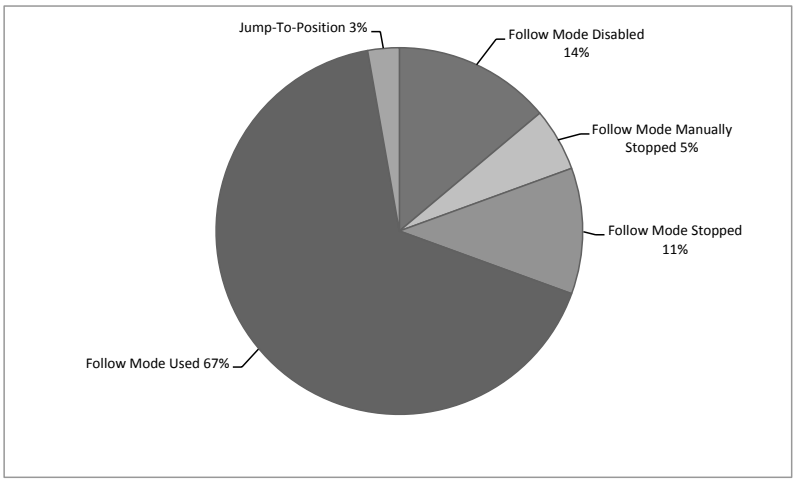

Figure 4.9: Proportional distribution of AWARENESS FEATURE^{SAttr} manifestations

the editor. Moreover, if a participant looses track of his partner's position in the workspace, he can use the *Jump-To-Position* feature to jump to the partner's cursor position in an artifact. The usage of these awareness features significantly impacts the mutual workspace awareness of the participants. AWARENESS FEATURE^{SAttr} gives some indication of whether and how tool support is adopted and used in certain situations and how it may be improved or extended.

4.4.8 Simple Attributes: Awareness Voicing and Awareness Bridging

Verbal communication is vitally important in PP as well as in eDPP. For conceptualizing activities and communication in PP, Salinger and Prechelt (2013) provide a so-called *Base Layer*. The *Base Layer* allows to break down the PP process in specified, fine-grained chunks of activities performed with the computer (human-computer interaction, HCI) or verbal exchange between the participants (human-human interaction). With that it facilitates a more detailed understanding of what is actually happening during the PP process. In the context of this groundwork research, the detailed examination of single utterances was not considered as an appropriate level of analysis. Nevertheless, the *Base Layer* was kept in mind and provided very helpful ideas when investigating eDPP phenomena.

In eDPP, verbal communication is a major source of workspace awareness (see page 71), all the more when the participants do not share viewports. Consequently, **verbal utterances** of the participants were conceptualized in terms of how much **indication about the participant's activities and thoughts they provide** to learn more about the remote partner's awareness.

Depending on the situation and the level of detail, the utterances of the participants served as explicit, intended awareness bridging mechanisms and for action coordination. They were also observed as indirect, unintended awareness improvements, for example verbalizations alongside actions. To capture the participants' **mutual level of awareness**, the awareness revealing aspect of verbal communication was captured for both participants.

The simple attribute AWARENESS BRIDGING^{SAttr} denotes whether and how the participant who initiated an eDPP-specific activity, in most cases the ACTIVE OBSERVER, bridges the physical distance via verbal awareness information about his activity. The simple attribute AWARENESS VOICING^{SAttr} in turn refers to the other participant, usually the DRIVER, and how he verbally reveals information about his current activity and thoughts.

? Which question does it answer?

How do the participants verbally provide awareness information about their current activities and thoughts?

The following list summarizes the manifestations of Awareness Bridging^{SAttr} and Awareness Voicing^{SAttr} observed in the data. Refer to Figure 4.10 for the proportional distribution of Awareness Bridging^{SAttr} and to Figure 4.11 for Awareness Voicing^{SAttr}.

Observed manifestations of Awareness Voicing^{SAttr} and Awareness Bridging^{SAttr}:

- **Thinking Aloud**: A participant verbalized his thoughts – his reasoning, questions or considerations. This was often accomplished by short questions like *"isn't it?"* to receive approval. In unilaterally visible dialogs or when the participants separately explored artifacts, it also included verbalized nuggets of information about the used dialog or the participant's input.
- **Mumbling**: The participant uttered something under his breath alongside with his actions. These utterances were no full sentences, nor were they explicitly directed at the other one. They were hard or in most cases impossible to understand. They thus indicated the participant was busy thinking but withheld further details. See also mumbling in Announcement Style^{SAttr}.
- **None**: The participant was completely silent.

For Awareness Voicing^{SAttr} (for the participant who did not initiate an eDPP phenomenon), also the following instance has been observed:

- **React**: The active participant asked the other one for confirmation of his plans or thoughts and the partner followed his thoughts and reacted to his questions or confirmation requests. Such a reaction often was a simple 'yes'. The participant did not actively provide information, but his reactions indicated that he was following the active participant's activities and reasoning.

The following instances for Awareness Bridging^{SAttr} have been observed for the participant who initiated an eDPP phenomenon:

- **Inform Finding**: The active participant did not provide any information while he was doing something, like looking up something. As soon as he was done or found the information he was looking for, he informed the partner about his thoughts. With that he provided a kind of downstream awareness information about his activity.
- **Rationalization**: The active participant did not verbalize what he was doing or that he was doing something at all. He just shortly gave reasons for his activity. For example, the Active Observer explained that he usually arranges methods according to their access modifier while he was doing it right away or he uttered that a variable was misspelled while correcting it.
- **Unassertive Reaction**: One participant performed a solo action without verbalizing anything about it. When the partner reached out to him, he answered hesitatingly. This happened in a situation in which he was not fully on track of the joint thinking process but concerned with something else, for example when looking up something while the partner continued to work. This behavior is no direct information about the actual activity of a participant, it just indicates that he is in a thinking process.

Verbal communication is fairly important for mutual awareness in distributed real-time collaboration. It not only provides awareness about the other one's activities and thoughts, but equally important maintains the sense of being actively connected. Verbal feedback, if only some mumbling, avoids the 'dead line' impression. This is why Awareness Voicing^{SAttr} and Awareness Bridging^{SAttr} are important contextual factors of an eDPP phenomenon that provide insights about the degree of tool-independent mutual awareness of the participants.

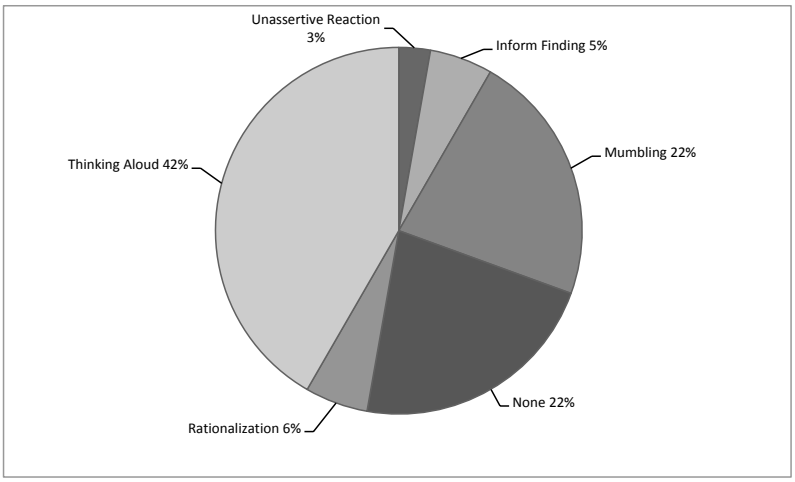

Figure 4.10: Proportional distribution of AWARENESS BRIDGING^{SAttr} manifestations

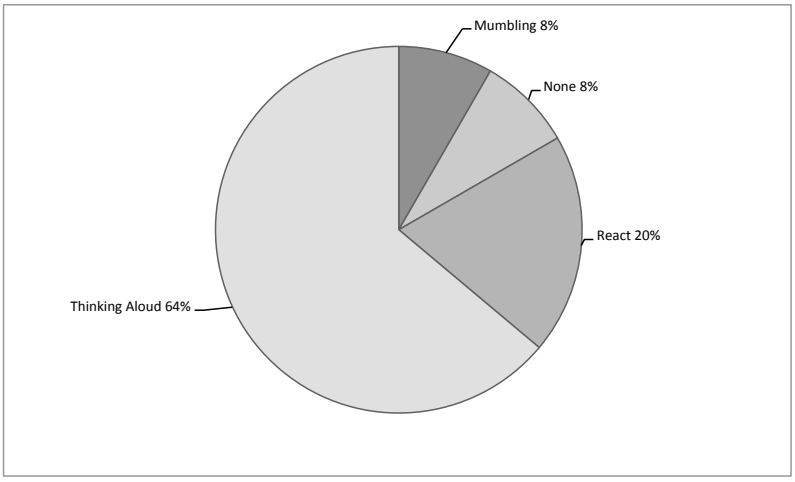

Figure 4.11: Proportional distribution of AWARENESS VOICING^{SAttr} manifestations

4.4.9 Simple Attribute: Start Mode

Before working separately, a pair must take a conscious decision when to start this other form of work mode. This was observed not only before phases of unilateral execution of previously agreed and prepared work but also for episodes that started implicitly without prior determination.

The **degree of previous agreement or predestination of activities** is represented in the simple attribute START MODE^{SAttr} of separated work mode episodes.

? Which question does it answer?

How do episodes of separated work emerge?

The following list shows the manifestations of START MODE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.12.

- **Execute Discussed**: The separation succeeded the agreement on the allocation of tasks. The pair discussed the next work step which included an operation in an unshared view, for example

the creation of a class or interface using a dialog. They also determined who would execute the operation.

- **Issue Unilateral Start**: Due to an issue, one participant started to look up something. The partner became aware of this and, in order to help with clarifying the issue, also started to independently explore in the workspace.
- **Issue Unilateral Request**: A participant started to examine an issue in an unshared view and prompted the other participant to go there, too. For example, when faced with an issue regarding file properties, one participant examined a file list on a server via the terminal and asked the other one to also log in there to have a look at the files. While sharing their view, the pair then further discussed the issue.
- **Proactivity**: One participant started an unshared action, like using a refactoring dialog, single-handedly and without agreeing on it with his partner.

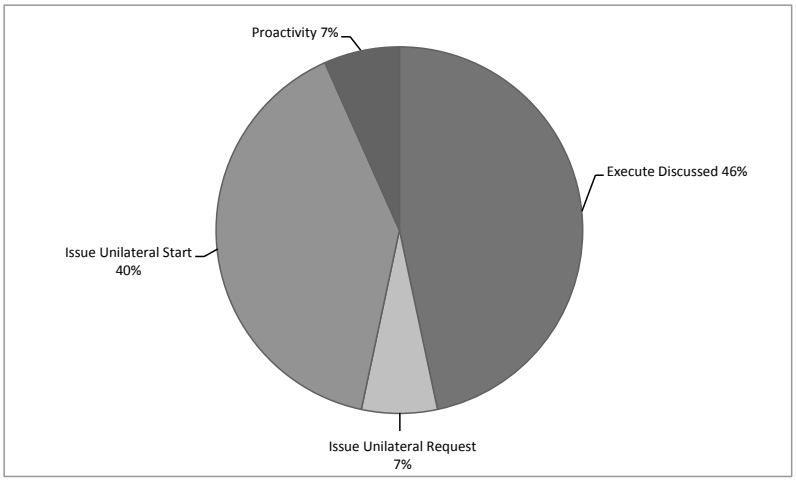

Figure 4.12: Proportional distribution of START MODE^{SAttr} manifestations

START MODE^{SAttr} is a relevant phenomenon attribute since it allows implications about the criticality of the participants' separation regarding the loss of shared problem solving and review effect. It makes a difference whether something pre-determined is just accomplished during the phase of separated work, or whether it comprises activities where having a second mind and a second pair of eyes is very valuable. This in turn is strongly connected to the type and complexity of the performed activity (see Section 4.4.5 "Simple Attribute: Shared-Mind Activity").

4.4.10 Simple Attribute: Termination Mode

After having operated in a separated work mode, a pair eventually leaves their mode and mets again in the workspace. In general the pair got together again because they successfully solved their issue or they decided to take another path.

These **episode-terminating aspects** are comprised in the attribute TERMINATION MODE^{SAttr}.

? Which question does it answer?

Why does the pair stop their separated work mode?

The following list shows the manifestations of TERMINATION MODE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.13.

- **Issue Resolved**: The issue which had been the reason for the separated working phase was resolved.
- **Different Topic**: The participants switched their focus, they 'discovered' another topic that they turned to.
- **Subtask Finished**: When the pair split up to work on different subtasks in parallel, they terminated this parallel work when they were done with their respective subtasks.

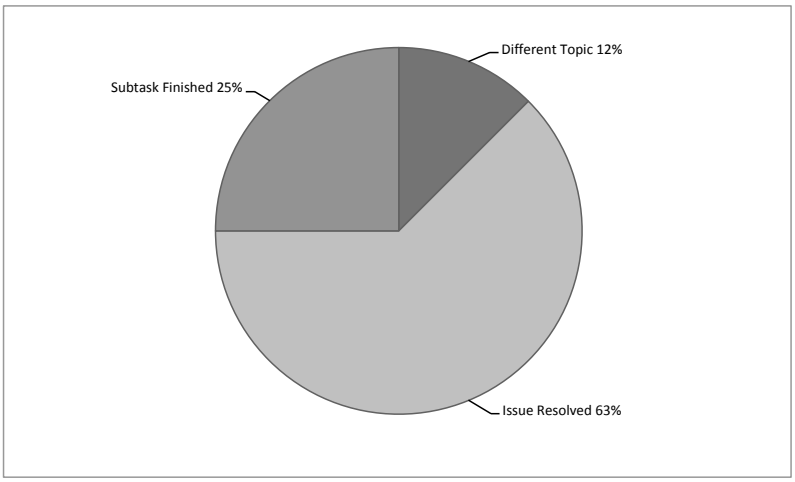

Figure 4.13: Proportional distribution of TERMINATION MODE^{SAttr} manifestations

TERMINATION MODE^{SAttr} provides insights about the reasons for finalization and achievements of separated work phases.

4.4.11 Simple Attribute: Join Mode

After the pair terminated their separated work mode (when they were working in parallel (see Section 4.6.3 "eDPP Phenomenon: Parallelization") or separately exploring something (see Section 4.6.1 "eDPP Phenomenon: Parallel Exploration")), they found together again in the workspace. They either used the *Jump-To-Position* feature of *Saros*, synchronized verbally, or were in the same viewport anyway.

The simple attribute JOIN MODE^{SAttr} captures **how the pair found together again in the workspace after having worked separately**.

? Which question does it answer?

How does the pair find together again in the shared workspace after a period of separated work?

The following list shows the manifestations of JOIN MODE^{SAttr} observed in the data. Their proportional distribution is visualized in Figure 4.14.

- **Meet in Viewport**: The participants closed their separate locations and joined up in a shared viewport, usually the original one.
- **Jump to Position**: One participant used the *Jump-To-Position* feature of *Saros* and thus touched down in the viewport of the other one (automatically being in the *Follow Mode*).

- **Explicit**: A participant's utterance explicitly signaled the desire to come together in the shared workspace, like *"O.K., are you back in Eclipse?"*

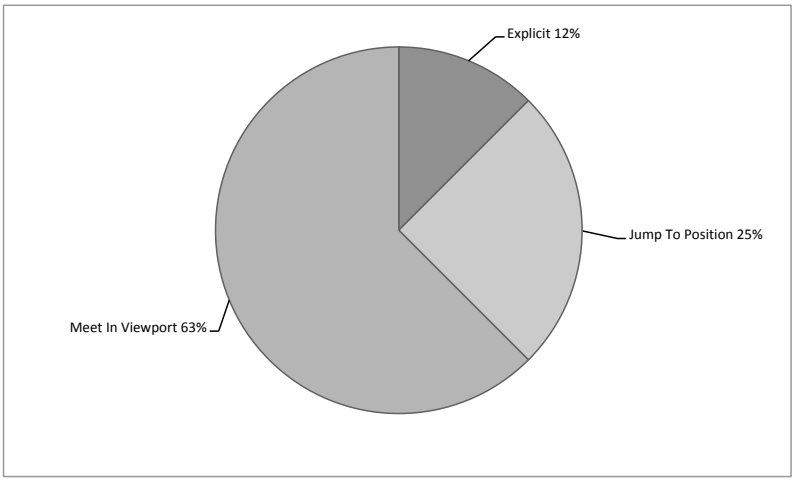

Figure 4.14: Proportional distribution of JOIN MODE^{SAttr} manifestations

JOIN MODE^{SAttr} provides insights about the mechanisms the participants use to locate each other in the workspace. This is also relevant for improving adequate functionality in eDPP tools.

4.4.12 Simple Attributes: Fault and Fault Recognition

When one participant made a unilateral operation (see Section 4.6.2 "eDPP Phenomenon: Unilateral Operation"), his activities and changes were not visible to the other one. Occasionally, in such cases, faults slipped in.

The two related attributes FAULT^{SAttr} and FAULT RECOGNITION^{SAttr} describe **occurrences of (perceived) faults during unilaterally visible operations**.

? Which question does it answer?

Does a fault occur during a UNILATERAL OPERATION, and if so, when is it recognized by whom?

The following list shows the manifestations of FAULT^{SAttr} observed in the data. Figure 4.15 visualizes the proportional distribution of UNILATERAL OPERATION phenomena and, within that, the distribution of manifestations of FAULT^{SAttr}.

- **None**: No fault occurred during a unilateral operation.
- **Typo**: The participant made a typo when using a unilateral dialog, for example misspelling a class name in the new class wizard.
- **Incorrect Operation**: The participant made a mistake when performing a unilateral operation. This was for example the incorrect handling of a dialog for creating a test class, where he did not appropriately specify the class under test.

The following manifestations of FAULT RECOGNITION^{SAttr} have been observed in cases a fault slipped in. Figure 4.16 visualizes the proportional distribution of UNILATERAL OPERATION phenomena where a fault slipped in and, within that, the distribution of manifestations of FAULT RECOGNITION^{SAttr}.

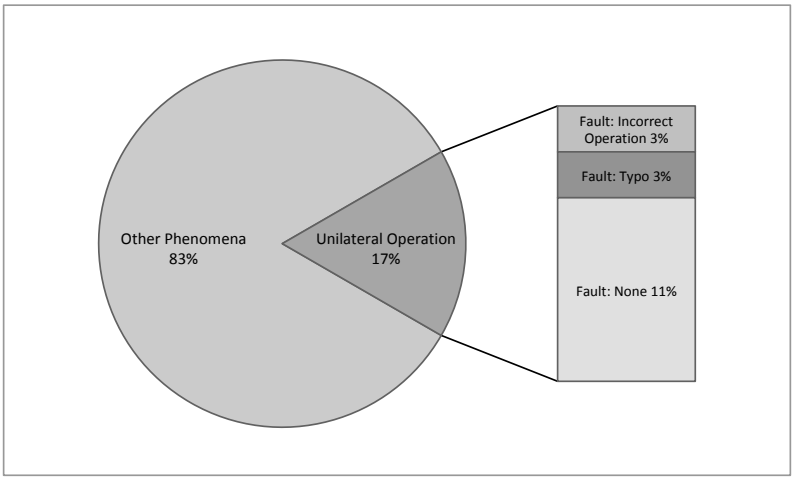

Figure 4.15: Proportional distribution of UNILATERAL OPERATION phenomena and FAULT^{SAttr} manifestations

The pie chart on the left shows the proportional distribution of UNILATERAL OPERATION phenomena versus all other phenomena types. The stacked chart on the right shows the manifestations of FAULT^{SAttr} observed in UNILATERAL OPERATIONs.

- **Immediately, by Observer**: While the DRIVER finished the unilateral operation, the observer recognized the misspelling at once. It was immediately fixed by the DRIVER and had no further impact on the product or process.
- **Delayed, by Driver**: The DRIVER performed a unilateral operation. After a short time (less than a minute), when starting to work with the outcome of this unilateral operation (i.e. the generated artifact), the DRIVER recognized his mistake and corrected it. The fault was corrected with a short delay but still without essential impact on the process or product.

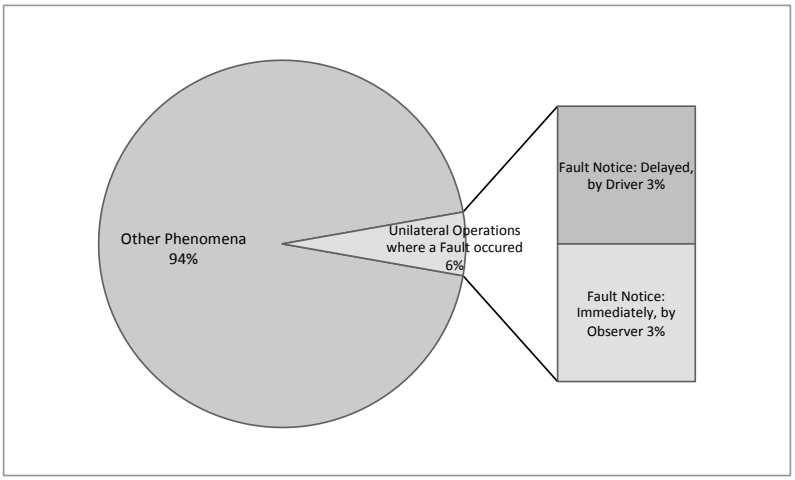

Figure 4.16: Proportional distribution of UNILATERAL OPERATIONs with FAULTs

The pie chart on the left shows the proportional distribution of UNILATERAL OPERATION phenomena with slipped-in FAULTs versus all other phenomena types. The stacked chart on the right visualizes the manifestations of FAULT RECOGNITION^{SAttr} observed for these UNILATERAL OPERATIONs that involved a FAULT.

The attributes FAULT^{SAttr} and FAULT RECOGNITION^{SAttr} help to assess the criticality of the loss of the review effect during a unilateral operation. If a fault is promptly noticed, it can be corrected immediately,

and the short delay of the review does not make an essential difference. It would only be critical if the fault was not recognized after the operation. Then it would remain in the product or result in a downstream debugging effort where the defect's origin is no longer clear.

As can be seen in Figure 4.15 and 4.16, faults slipped in very rarely during UNILATERAL OPERATIONS. As far as it could be observed, none of them remained undetected.

4.4.13 Complex Attribute: Coupling of Participants

"In pair programming, two programmers are assigned to jointly produce one artifact (design, algorithm, code, among others). The two programmers are like a coherent, intelligent organism working with one mind, responsible for every aspect of this artifact" (Williams and Kessler, 2000).

To work efficiently, this *"coherent organism working with one mind"* requires an extremely tight mental coupling of the pair. COUPLING OF PARTICIPANTS^{CAttr} describes the **degree to which the participants follow only one single, joint train of thought** as opposed to two separate strands.

? Which question does it answer?

To which degree do the participants work as one brain?

Even in PP it may occur that the participants mentally decouple, for example due to some kind of disengagement, as examined in Plonka et al. (2012b). In eDPP, the participants can not only decouple mentally, but also visually. They can easily leave the common viewport and go somewhere else without their partner recognizing this stroll. Since the straggling participant is in another context, **visual decoupling significantly increases the risk of mental decoupling**. In eDPP, due to the increased freedom of action, decoupling is not necessarily a disengagement in the collaboration.

Coupling is difficult to observe and to operationalize. It can happen gradually and be related to different aspects. Particularly this attribute is an example of context information that can only be observed and derived because of the researcher's 'big brother' role who simultaneously oversees both screens and webcams of the participants.

Three aspects were observed in the data as appropriate indicators for the coupling or decoupling of the participants:

- *Visual Closeness*$^{CAttr\text{-}A}$: Congruence of both participants' focused area in the shared workspace. The attribute determines the visual availability of awareness information about the other one's actions, view, and location on different levels of detail.
- *Subject Awareness*$^{CAttr\text{-}A}$: Mutual awareness of the other one's mental concern, so to say the subject of his mental focus. This is not necessarily dependent on detailed visual awareness information.
- *Cognitive Affinity*$^{CAttr\text{-}A}$: This denotes to which degree the participants think about the same aspect.

The following paragraphs describe these aspects and their instances in detail.

4.4.13.1 Visual Closeness

eDPP mainly happens in the *Eclipse* IDE, involving shared artifacts with awareness annotations as well as unshared areas. In regard to decoupling, the most obvious deviation in eDPP compared to PP is that the participants can move away from each other in the workspace. This viewport aspect of coupling is represented in the aspect *Visual Closeness*$^{CAttr\text{-}A}$. It represents the **congruence or closeness of both participants' focused workspace area**.

A clear indicator of the current focus of a participant were his verbalizations or editing actions in the respective area since this required his visual and cognitive attention. Other strong indicators for the visual focus were activities like mouse gestures or the selection of text. Finally, the workspace interactions performed by the participants were helpful indicators for precisely determining their *Visual Closeness*$^{\text{CAttr-A}}$.

The following list shows the manifestations of *Visual Closeness*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.17.

The first set of manifestations relates to combinations of when both participants had a common viewport.

- **Shared Viewport**: Both participants had their focus within the same shared viewport. A further determination of their specific focus within this viewport was not possible. At this level, they may be aware of the other one's editing actions and location in the workspace, though not necessarily.
- **Statement**: With a shared viewport, both participants had their focus on the same statement. Possibly, there were a few lines between their foci, but they were in the same method or block of code. Finally, the participants had a good action and presence awareness.
- **Line**: Having a shared viewport, both participants had their focus on the same line of code. Here, the same as above holds for action and presence awareness.

The second set of viewport combinations relates to settings where the participants had different or unshared foci.

- **Unshared View**: Both participants focused on unshared views of *Eclipse*, like the package explorer, a context menu, or a dialog. Both participants still might have had the editor view available, but their respective actions in the unshared views were mutually invisible. At most the effects of this interaction were visible, for example when they opened or created an artifact.
- **Shared Artifacts Independent**: Both participants separately moved within the same or different shared artifacts, periodically having the same viewport or not. Based on viewport annotations, the participants may have known at which part of an artifact the other one was looking or which artifact he opened.
- **Shared Artifact Different**: Both participants were in shared but different artifacts. In this setting, only the awareness information about the other one's currently opened file was available.
- **Different Application**: The participants resided in different applications, for example the IDE one the one side and a web browser on the other side, or web browsers or other applications on both sides. In this mode, the eDPP tool does not provide information about the position of the partner or whether he left the shared workspace.
- **Unshared Application Same Contents**: Both participants used another application than *Eclipse*, but within this application, they viewed the same contents. An example of such a situation was when both participants used their terminal to view the same directory. In such case, no awareness is provided by the tool, and the participants have to verbally sync their actions.

Visual Closeness$^{\text{CAttr-A}}$ determines the likely amount of the participants' mutual action and location awareness depending on the visual availability of awareness information.

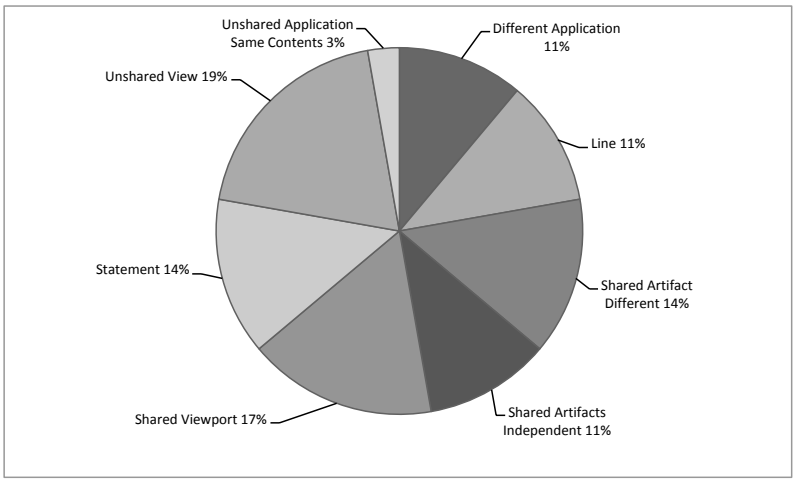

Figure 4.17: Proportional distribution of *Visual Closeness*$^{\text{CAttr-A}}$ manifestations

4.4.13.2 Subject Awareness

Visual perception was not the only source for awareness about what the other one is doing in the workspace. Even if a participant could not see the other one's activities, he often knew what his partner was concerned with. He knew because an agreement had been made on what the partner was going to do, or he could infer it from the partner's implicit or explicit verbal utterances. For example, when one participant created a new interface using the *New Interface* dialog, the other one could not see the dialog input. However, since the participants had agreed that one of them would create the interface, secondly had discussed the interface name and properties, and thirdly, the executive participant had verbalized his thoughts and input alongside his actions, the other participant had a good notion about what the other one was mentally concerned with.

Finally, *Subject Awareness*$^{\text{CAttr-A}}$ is a more abstract action awareness than pure visual perception: it is the **general understanding of what the other one is concerned with**, without necessarily knowing the detailed activities the other is performing.

The following list shows the manifestations of *Subject Awareness*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.18.

- **Bilateral**: Both participants were aware of what the other one is concerned with. In most cases, this mental focus was congruent, but it also happened that the participants split their work while still being aware of what the other one was concerned with.
- **Bilateral+**: Both participants were generally aware of what the other one was doing. Nevertheless, one participant additionally performed a trivial and quite short activity relating to the current work aspect, for example by quickly looking up something while the other one was not aware of this.
- **Stick in Aspect**: Both participants had a bilateral *Subject Awareness*$^{\text{CAttr-A}}$. Then, one of them stuck with an aspect of that joint work strand while the other went on. Accordingly, until the stuck participant was done, both participants were not completely aware of what the other one was concerned with: The stuck participant did not follow the continuing work, and the continuing participant was not aware of the other one's lingering. For example, the pair jointly reflected about a meaningful name for a method they declared in an interface. The DRIVER was happy with the name and continued his work while the ACTIVE OBSERVER checked the appropriateness of the method's name in another file and adapted it.
- **Unilateral Unaware**: One of the participants was not aware of the fact that the other one

cognitively left and engaged in another aspect.

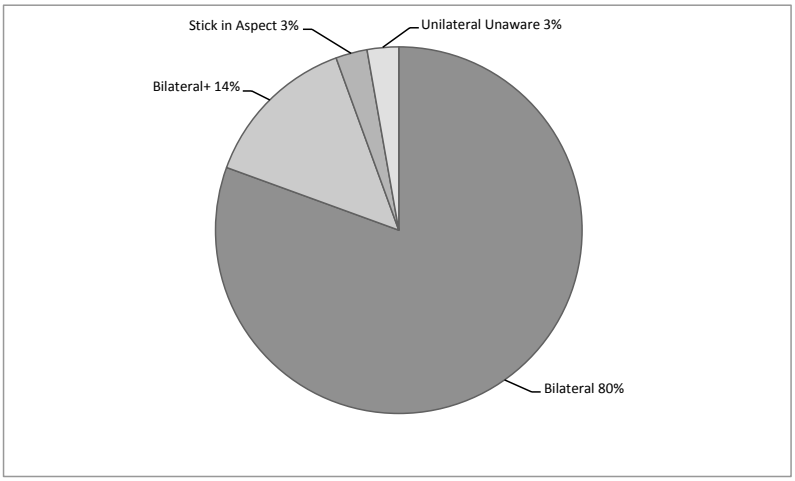

Figure 4.18: Proportional distribution of *Subject Awareness*$^{\text{CAttr-A}}$ manifestations

Similarly to *Visual Closeness*$^{\text{CAttr-A}}$, *Subject Awareness*$^{\text{CAttr-A}}$ relates to awareness of the other one's actions but based on other information than visual feedback.

4.4.13.3 Cognitive Affinity

PP *"is a powerful technique as there are two brains concentrating on the same problem all the time"* (Williams and Kessler, 2000). Due to the independent workspaces in eDPP, this forcing of joint concentration is notably weaker. The participants can more easily deviate with regard to the aspect they are thinking about. At the same time, mental digressions are harder to notice due to the lack of nonverbal clues.

Even when the participants were aware of what the other one was concerned with, they did not always focus cognitively on that to the same degree. They sometimes thought about something different.

Cognitive Affinity$^{\text{CAttr-A}}$ denotes the congruence or affinity of the participants' cognitive focus – the **degree to which they work as one brain**.

The following list shows the manifestations of *Cognitive Affinity*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.19.

- **One Brain**: The participants worked as 'one brain', bundling their concentration on the same aspect of the problem. There was no change of mode when a participant did something of cosmetic or quick fixing character which did not involve noteworthy mental effort. Examples of such no-brainers were minimal corrections or polishing of what the Driver just had done by the Active Observer, such as things the driver simply missed but that did not involve much reflection like fixing a typo or inserting a blank line while cognitively staying with the Driver.
- **One Brain μ**: The participants worked as one brain, but one of them performed a quite short 'micro' activity on code level that was an improvement or addition to the Driver's work. Such an activity is some sort of own consideration relating to the jointly focused aspect without requiring noteworthy mental capacity. For example, the Active Observer cleaned the code by extracting a local variable from a statement the Driver had just written, or he changed the name of a method to be more meaningful.
- **Fork Return**: The participants worked as 'one brain'. Then, the Active Observer had a related but own idea and briefly opted out to work on that idea while the Driver continued to

work on their joint work strand. The ACTIVE OBSERVER was still in the current work context but temporarily concentrated on another aspect. When he was done, he returned to the DRIVER's train of thought. For example, the ACTIVE OBSERVER refactored some code in the context of the general joint work thread but without coordinating this with the DRIVER. *Fork Return*$^{\text{CAttr-A}}$ is a stronger mental digression than *One Brain* $\mu^{\text{CAttr-A}}$, since the ACTIVE OBSERVER briefly turned his hands to another idea but still within the current joint line of thought.

- **Different Aspect**: The ACTIVE OBSERVER turned his attention to another issue than the DRIVER was currently concerned with. This issue was part of their overall goal but not an aspect of the current joint work thread.
- **Two Brains In Unison**: The participants split to concentrate on different aspects of the same task. This separated work mode was mutually agreed upon, for example when the pair realized they needed an object they did not originally consider. This recognition involved the creation of the required object as well as some adjustments in an existing statement. The pair decided to split up and to tackle these two aspects in parallel.

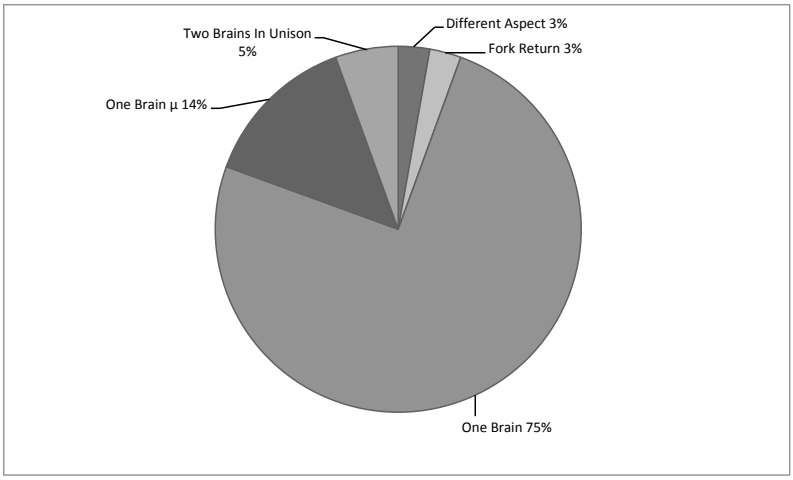

Figure 4.19: Proportional distribution of *Cognitive Affinity*$^{\text{CAttr-A}}$ manifestations

The mental coupling of the participants is an essential aspect of the potential benefits of collaborative work like PP or eDPP. Visual decoupling is not a problem per se, it can happen intentionally, for example if the participants separately explore in the workspace while verbally exchanging their insights and when they afterwards find together again. The important point, however, is that the pair should not lack the brain synthesis effect which would ruin lots of the potential benefits of pair programming. Therefore, COUPLING OF PARTICIPANTS$^{\text{CAttr}}$ is valuable information to assess the reasonability of PP in a distributed setting.

4.4.14 Complex Attribute: Fluency

As already discussed, the independent workspaces in eDPP with social floor control allow the participants to decouple or to slide in the partner's activities.

The data showed that the action taking of the ACTIVE OBSERVER affected the DRIVER's current train of thought to some degree, depending on the invasiveness of the action. In this context, invasiveness can be considered as the severity of the mental or physical context switch of the DRIVER in response to the ACTIVE OBSERVER's activity.

In any case, an eDPP phenomenon influences the eDPP process and its fluency to some extent. This process-influencing is represented in the complex attribute FLUENCY$^{\text{CAttr}}$.

On the process level, FLUENCY^{CAttr} is considered as **smooth in the sense of being free of avoidable interruptions** that originate in the reduced awareness or limited sharing in eDPP.

? Which question does it answer?

How do the activities that are possible in the workspace due to the increased flexibility of an eDPP participant influence the smoothness of the eDPP process?

As with COUPLING OF PARTICIPANTS^{CAttr}, it is neither straightforward to fully comprehend process-fluency in eDPP nor to describe it. It comprises several aspects, and for eDPP phenomena, the following attributes appeared to be appropriate indicators:

- **Synchronization Effort**: Denotes whether in the context of an eDPP phenomenon there was explicit synchronization effort between the participants due to lack of awareness information.
- **Action Recognition**: Describes whether one participant recognized when his partner became active in the workspace in the sense of eDPP-specific behavior. If for example the DRIVER did not recognize an intervention of the ACTIVE OBSERVER, it did not have a direct (local) impact on his behavior.
- **Handling of Intervention**: When one participant recognized the other one's activity, he somehow handled this intervention – he behaved in some way in reaction to that activity.
- **Effect on Train of Thoughts**: Refers to how one participant integrated the partner's activity in his current train of thought or flow of work. Compared to *Handling of Intervention*$^{CAttr\text{-}A}$, which is about the participant's behavior during the other one's activity, this aspect covers the impact on the cognitive thread.

4.4.14.1 Synchronization Effort

Synchronization Effort$^{CAttr\text{-}A}$ indicates whether an **awareness or coordination problem due to an eDPP phenomenon occurred that had to be fixed by explicit efforts**. A *Synchronization Effort*$^{CAttr\text{-}A}$ activity is an **additional, not problem solution-oriented activity**. Due to this process-repairing character with additional expenditure for the participants, it negatively influences the smoothness of the collaboration process – they are not working on the core problem solving.

The following list shows the manifestations of *Synchronization Effort*$^{CAttr\text{-}A}$ observed in the data. Their proportional distribution is visualized in Figure 4.20.

- **Not Observed**: No explicit synchronization efforts could be observed.
- **Quick Context**: For a moment, the ACTIVE OBSERVER did not follow the DRIVER's activity and thus was not in the loop regarding his train of thought. Accordingly, before he was able to adequately react to a question of the DRIVER, he had to ask for its context.
- **Ping Presence**: One participant explicitly aligned his awareness about the other one's presence in the shared workspace. This occurred when the pair separately worked in an unshared application. When they were done with their task in the unshared application, the one who was back in *Eclipse*, the shared workspace, asked the other one whether he was also back: *"Are you back again in Eclipse?"*.
- **Ping Focus**: While the DRIVER operated a unilaterally visible dialog, he checked back whether the other one was mentally following him. He aligned his awareness about the other one's current mental focus with a question like *"Are you with me?"*.

- **Ping Doing**: When one participant overheard that the other one was acting but could not see corresponding changes in the workspace, he asked his partner what he was doing. He perceived a gap between visual and auditive feedback which confused him. He explicitly aligned his awareness about the other one's activity, for example by asking *"What are you typing?"*.
- **Ping Doing Past**: Surprised, the DRIVER recognized the effect of an operation the ACTIVE OBSERVER had done. Confused, he asked him what he had been doing before. He aligned his awareness about the ACTIVE OBSERVER's past activity by asking *"What have you done?"*.
- **Ping Update**: The DRIVER was generally aware of an action of the ACTIVE OBSERVER but could not see the execution. Since the ACTIVE OBSERVER did not proactively give verbal feedback, the DRIVER asked how matters stood in between. For example, the ACTIVE OBSERVER more or less silently looked up something regarding the solution of a problem the pair was faced with, and in between the DRIVER asked *"Is there something?"*.

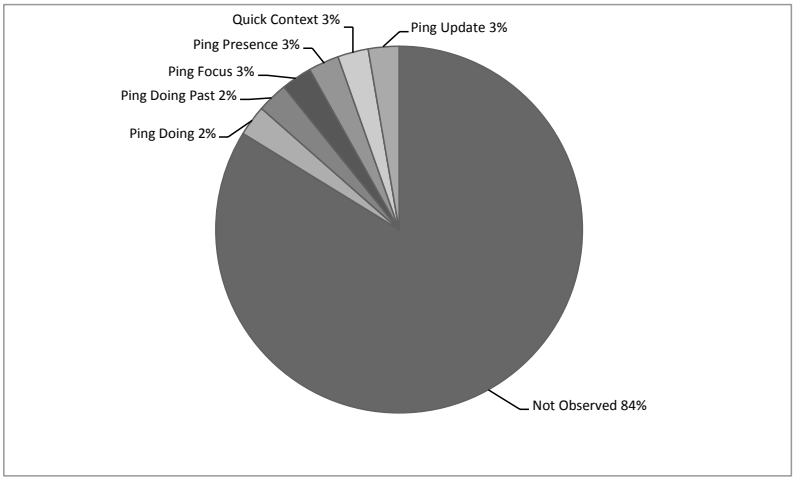

Figure 4.20: Proportional distribution of *Synchronization Effort*$^{\text{CAttr-A}}$ manifestations

4.4.14.2 Action Recognition

When one or both participants started an eDPP-specific action, the remote partner either recognized that action or not. Beyond auditive hints, the current visual coupling plays a major role for the possibility of becoming aware of the other one's eDPP-specific activity, for example when the ACTIVE OBSERVER sprang into action, when the DRIVER started a unilateral operation, or when both explored their workspaces. **The recognition of the other one's eDPP-specific activity in the workspace** is captured in the aspect *Action Recognition*$^{\text{CAttr-A}}$.

The following list shows the manifestations of *Action Recognition*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.21.

- **Aware**: A participant immediately recognized the other one's activity, usually because they had agreed on it previously or the other one announced his intent.
- **Downstream**: The DRIVER did not directly and immediately perceive the ACTIVE OBSERVER's action, but the activity resulted in some visible changes or communicated findings. Accordingly, the DRIVER got an indirect, downstream awareness about the activity. He was not aware of what happened at the ACTIVE OBSERVER's site the whole time, but he saw or was informed about the effects of an activity. Effects of such actions were refactored or pasted code snippets, or the ACTIVE OBSERVER informed the DRIVER about an insight he had when looking up something.

- **Not Possible**: When the ACTIVE OBSERVER became active without announcing his activity and the changes were at a position not visible to the DRIVER, he could not recognize the action. Such 'hidden' actions happened for example in source code that was overlaid by the DRIVER's context menu or in a different artifact than he had focused on.
- **Not Observed**: The ACTIVE OBSERVER's action was within the shared viewport, but no reaction from the DRIVER could be observed. It cannot not be further determined whether he really did not perceive the action or whether he just did not react to it, maybe because he decided the intervention to be minimal enough it would not require his involvement.

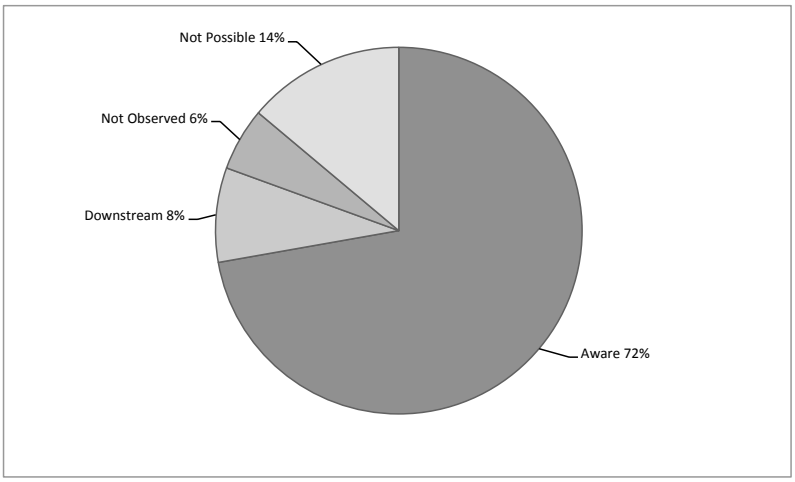

Figure 4.21: Proportional distribution of *Action Recognition*$^{\text{CAttr-A}}$ manifestations

4.4.14.3 Handling of Intervention

If a participant recognized an activity of his partner, this influenced the continuation of his own current work. In this context, work refers to editing or operating as well as to mental activities like reflecting about something.

Handling of Intervention$^{\text{CAttr-A}}$ describes the **effect on a participant's work process when recognizing the partner's activity**.

The following list shows the manifestations of *Handling of Intervention*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.22.

- **Temporizing**: When one participant recognized the intervention of the other one, he stopped his own work, waiting for the other one to finish.
- **Continue**: The DRIVER shortly and slightly slowed down his current activity the moment he noticed the ACTIVE OBSERVER's activity. However, he did not essentially react to the intervention but continued his own work. In most cases, the DRIVER was aware of the ACTIVE OBSERVER's activities, and since they were directly related and joined-up with his train of thought, he continued with his work. When no recognition by the DRIVER was observable, he accordingly continued his work unaffectedly.
- **Pause Join In**: When the DRIVER recognized the ACTIVE OBSERVER's activity, he paused his own work and mentally joined in, often by starting a conversation about what the ACTIVE OBSERVER had started to do. Whether this may be considered as a driver change or not will be discussed later.
- **Not Possible**: When the DRIVER did not recognize the ACTIVE OBSERVER's activity, he accordingly did not react to it.

- **Parallel Handling**: When the pair simultaneously performed actions like exploring in the workspace or working on subtasks, there was no single-sided intervention but parallel actions.
- **Reject**: The DRIVER recognized the intervention of the ACTIVE OBSERVER but stopped him by rejecting his started activity: *"Nope, let's not do it yet."*

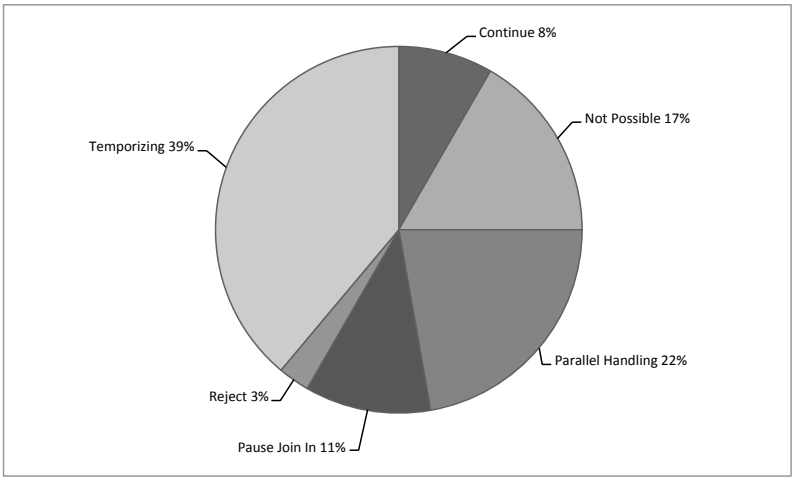

Figure 4.22: Proportional distribution of *Handling of Intervention*$^{\text{CAttr-A}}$ manifestations

4.4.14.4 Effect on Train of Thoughts

The occurrence of an intervention by a participant did not only have an impact on the other one's behavior, but the factual contribution of that intervention also affected the other one's mental process.

Effect on Train of Thoughts$^{\text{CAttr-A}}$ describes **to what extent the interruption of the activity of one participant impacts or changes the partner's thinking process** – whether or how the intervention is included in his line of thought.

Interruptions and excessive, inappropriate turnarounds in the problem solution process are neither helpful nor efficient. Mental context switches and interruptions of the train of thoughts can significantly hamper flow – the mental state of deep concentration and energetic work. *Effect on Train of Thoughts*$^{\text{CAttr-A}}$ addresses the overall process fluency from the perspective of the pair's concentration and focused, purposeful work.

The following list shows the manifestations of *Effect on Train of Thoughts*$^{\text{CAttr-A}}$ observed in the data. Their proportional distribution is visualized in Figure 4.23.

- **Integrate**: When one or both participants became active in the sense of an eDPP-specific activity, the respective partner integrated it into his work or mental thread. He reacted to the intervention or contribution but it did not change his current train of thought because it could be considered as part of it. Maybe his line of thought swerved to some extent. This, however, is hard to observe without knowing the original or planned train of thought.
- **Not Observed**: A participant did not react to or include the other one's activities in his own work progress – his train of thought appeared to be unaffected.
- **Notice**: The DRIVER noticed the ACTIVE OBSERVER's activity in the sense of registering it, for example with a *"Uh-huh"*, but did not further react to it.
- **Trip in Tactic**: The DRIVER paused his current train of thought and engaged in the aspect considered by the ACTIVE OBSERVER. Since this aspect was directly related to the DRIVER's work, it did not change his tactical work process. For instance, while the DRIVER implemented

an interface of the *Observer* design pattern[100], the ACTIVE OBSERVER looked it up. They shortly discussed the required elements of the interface and whether they had declared all required elements for that interface. Thus, the intervention led to an extra clarification episode concerning the current work step.

- **Correct**: Based on the activity or input of the ACTIVE OBSERVER, the DRIVER corrected his current work. For example, the DRIVER implemented a method that threw an exception of a specific type. At the same time, the ACTIVE OBSERVER checked in his web browser whether this was the correct exception type for the current case. He found that this was not the case and told the DRIVER the correct exception type to throw. The DRIVER corrected his implementation but without changing his current line of thought in general.
- **Short Digression**: The intervention of one participant lead to a short thematic excursus. After the excursus, the pair switched back to the original task. Finally, the intervention shortly interrupted the current task execution but did not alter it. For example, initiated by a question of the DRIVER, the pair quickly started to search for a specific function in *Eclipse*, or they started a short exchange about access modifiers allowed to be used in Java interfaces, because the DRIVER had used some of them wrongly.

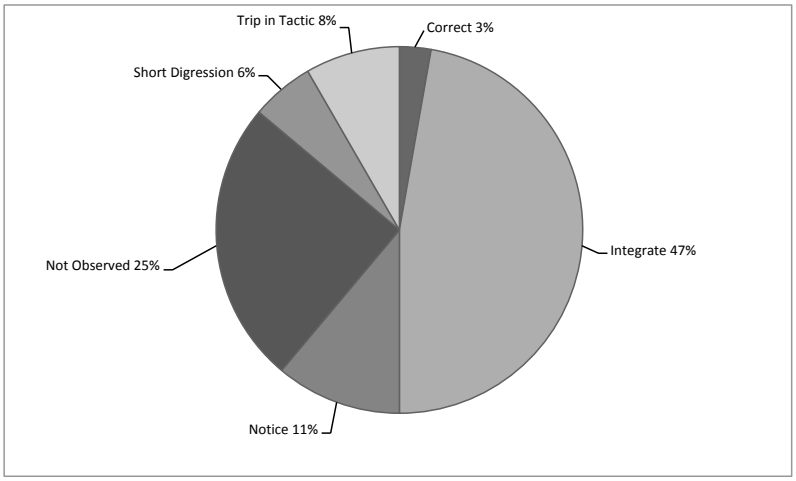

Figure 4.23: Proportional distribution of *Effect on Train of Thoughts*$^{\text{CAttr-A}}$ manifestations

FLUENCY$^{\text{CAttr}}$ allows conclusions about potential beneficial or harmful effects of interruptions on the efficiency and smoothness of the collaboration.

In PP, there are several possible scenarios for when the observer wants to make a contribution: He can decide to interrupt and navigate the driver, to dismiss his idea, or to note down his idea for discussing it when the driver will have finished his current train of thought. The latter requires a very reflective working style and is rather unlikely to happen in PP as well as in eDPP. The positive or negative impact of dismissing an idea depends on the quality and value of the idea for the solution process. In any case, interrupting and navigating the DRIVER in PP for this is very likely to be more invasive than just doing it on it's own as an ACTIVE OBSERVER in eDPP.

[100] `http://en.wikipedia.org/wiki/Observer_pattern` (accessed November 11, 2017)

4.5 Results Relating to an Interleaved Work Mode

4.5.1 eDPP Phenomenon: Direct Fix

In a nutshell: Direct Fix

Unobtrusive, trivial code corrections by the Active Observer.

When the Active Observer recognized a trivial issue in the code, he often used his editing freedom to fix it just on his own instead of asking the Driver to do it: he made corrections of misspellings, complemented names of variables or classes in order to make them more expressive, rearranged methods according to their visibility modifier, made a trivial code refactoring like the extraction of a local variable from an expression, or changed the visibility of a class attribute.

Such a **corrective engagement** of the Active Observer relating to the Driver's recent edits is an eDPP phenomenon of the type Direct Fix.

Example: Direct Fix

The Driver wrote `return x+y*z<t`. Without saying anything, the Active Observer selected the `y*z` in this line and used the *Extract Local Variable* dialog to move the multiplication snippet in a local variable, naming it according to its semantics. The Driver recognized this and, without further ado, continued his work.

Motive and impact

Direct Fix activities were of trivial character, mostly on a clean code level (Motive^{SAttr} = Clean Code Flaw) and occasionally on the level of correcting a syntax flaw or typo (Motive^{SAttr} = Syntax Error). Accordingly, they resulted in code improvements (Impact^{SAttr} = Cleaner Code) or fixes of syntax errors or misspellings (Impact^{SAttr} = Fixed Issue), like misspelled identifiers or forgotten braces and semicolons.

Announcement and awareness

The Active Observer often started his activity without saying anything (Announcement Style^{SAttr} = Silent) or with just mentioning something about the issue (Announcement Style^{SAttr} = Implicit). He never explicitly announced his intended action and mostly did not utter anything during his (usually quite short) activities (Awareness Bridging^{SAttr} = None). Sometimes he posed the reason for his activity (Awareness Bridging^{SAttr} = Rationalization) but without verbalizing his concrete actions. For example, he said that he prefers to order methods according to their visibility modifier while he was about to start reordering them. In other cases, he just mumbled something while he was fixing an issue (Awareness Bridging^{SAttr} = Mumbling).

During the Direct Fix operations of the Active Observer, the verbal behavior of the Driver usually did not change. In particular when he did not recognize or pay special attention to the Active Observer's activities, he continued to verbalize his train of thought (Awareness Voicing^{SAttr} = Thinking Aloud) or to mumble alongside his actions (Awareness Voicing^{SAttr} = Mumbling). In other cases, he said nothing at all (Awareness Voicing^{SAttr} = None) while watching the quick edit of the Active Observer. When the Active Observer provided some reasoning about his action, he reacted to that relatively unperturbedly (Awareness Voicing^{SAttr} = React). Almost permanently, the pair had the *Follow Mode* enabled (Awareness Feature^{SAttr} = Follow Mode Used). If not, the Active Observer used it to quickly jump to the Driver's location in the workspace (Awareness Feature^{SAttr} = Jump-To-Position).

Development focus and activity

Direct Fix phenomena always occurred while the participants were implementing code, actively working on the problem solution (Development Focus^{SAttr} = Implementation, Shared-Mind Activity^{SAttr} = Solution Development).

Due to the editing nature of the DIRECT FIX phenomena, they always happened in the IDE's editor (PLACE OF ACTION^{SAttr} = Editor). Visually, the participants mostly stayed coupled because the ACTIVE OBSERVER's fixes mostly happened in the same statement (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Statement) or even in the same line (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Line) the DRIVER was concerned with. Rather rarely, the DIRECT FIX activity happened somewhat further away but still very near to the block the DRIVER was currently editing (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Viewport). Only sometimes the DIRECT FIX activities were beyond the perception of the DRIVER because they took place in a shared but different artifact (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Artifact Different). Visual and mental coupling

DIRECT FIX activities were minimally invasive and of corrective or cosmetic character, directly related to the current activity of the DRIVER. Such activities like correcting a typo or inserting a blank line did not require substantial mental effort of the ACTIVE OBSERVER and his cognitive focus did not switch to another aspect. The participants always stayed mentally connected – they generally worked as one brain (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain) and were mutually aware of what the other one was concerned with (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral).

Some DIRECT FIX operations involved slight considerations the DRIVER had not thought of, for example improving a class name to be more meaningful or extracting a local variable from what the DRIVER had just written. Then the participants still had their mental focus on the same aspect of the problem solution, but additionally, the ACTIVE OBSERVER shortly reflected about another detail of that aspect (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain μ). In these cases, it occurred that the ACTIVE OBSERVER shortly got caught in such detail, while the DRIVER continued with the next thought (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Stick in Aspect). Once in this moment, the DRIVER raised a question and the ACTIVE OBSERVER could not adequately respond. He jumped to the DRIVER's position in the workspace first and then asked for the detailed context of the question (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Quick Context). Process fluency

In all other cases, no explicit synchronization effort was observed for DIRECT FIX episodes (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed). In just under half of the cases, the DRIVER apparently perceived the ACTIVE OBSERVER's activity (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware) and often delayed his own activity until the ACTIVE OBSERVER was finished (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Temporizing).

Nevertheless, even within the shared viewport, the DRIVER often did not pay special attention to the DIRECT FIX activities (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Not Observed) and continued his work (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Continue). When he could not recognize the actions of the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Not Possible), they accordingly did not affect his behavior (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Not Possible).

A DIRECT FIX never notably changed the DRIVER's train of thought, either because the driver did not react to it (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Not Observed) or because he noticed it, gave a short (agreeing) comment but did not further cognitively involve in it (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Notice).

DIRECT FIX is an *Editing-Freedom* phenomenon. It did not disturb the DRIVER's train of thought. Since the activities were short (usually less than a minute) and the code changes trivial, no mental decoupling occurred. The minimal additional cognitive effort required by the ACTIVE OBSERVER appeared to have no impact on his cognitive following of the DRIVER's work. This is why for the observed case, DIRECT FIX is considered to have no *Freedom-Negative-Effect*. Discussion of research questions

The interventions did not lead to notable awareness or coordination problems. Neither *Reduced-Workspace-Awareness* nor *Reduced-Physical-Awareness* seemed to be an issue.

Due to their minimal invasiveness, DIRECT FIX activities essentially did not appear to interrupt the DRIVER's train of thought or to actually stop his current work. The ACTIVE OBSERVER did changes

just on his own and thus did not have to stop the DRIVER to explain an issue to him and direct his attention to it. Thus, for the observed pair, DIRECT FIXes had a *Freedom-Positive-Effect*, leading to higher process fluency.

PP context In PP, a DIRECT FIX is not feasible in that unobtrusive manner. The observer cannot carry out fixes on his own without interrupting the driver to some degree.

eDPP context DIRECT FIX activities require trust between the participants because the DRIVER has to trust in the quality and adequacy of the ACTIVE OBSERVER's fixes. A sort of DIRECT FIX activity was addressed from a different perspective in the context of the evaluation of tools for DPP. For the tool Sangam (see Section 2.2 "eDPP Tools", page 98), Doepke and Soworka (2009) found: *"However, the navigator can still type in the editor. This is considered as a bug by the authors, but we found it useful as it can be used for correcting syntax errors without bothering the driver. This requires a protocol between both programmers as a downside."*

4.5.2 eDPP Phenomenon: Jump In

In a nutshell: Jump In

The DRIVER formulates the next work step and the ACTIVE OBSERVER jumps in to do it right away.

When the DRIVER expressed his considerations regarding the next work step and the ACTIVE OBSERVER already had the solution or a concrete implementation in mind, he used his editing freedom to JUMP IN the editor and inserted the solution directly in the code. He thus helped the DRIVER with the next step. For example, when the DRIVER was just about to finish his current code statement, he verbalized his thoughts regarding the next work step. The ACTIVE OBSERVER, however, knew or saw the respective code snippet and implemented it right away. In one situation, the ACTIVE OBSERVER jumped in when he noticed that the DRIVER was somehow overwhelmed with the mass of auto-generated methods of an artifact. On the spot, he hopped in the code and kicked out useless method declarations.

JUMP IN activities are directly related to what the DRIVER is currently concerned with. They appear to be a longer and more invasive DIRECT FIX where the DRIVER temporizes until the ACTIVE OBSERVER is done. The relevant difference between these two phenomena types is their character – a DIRECT FIX is a correction or improvement of what the DRIVER just did, a JUMP IN is an **assistance in the sense of an active continuation of his train of thought**.

Example: Jump In

At some point, the DRIVER thought out loud about the statement that had to go into the if-block he had just created. Right away, the ACTIVE OBSERVER cut a statement from a few lines above and put it in the if-block. The DRIVER gratefully accepted this (*"ah, yes"*) and continued his work.

Motive and impact The ACTIVE OBSERVER used his editing freedom to jump in the editor for several reasons. When he had a solution in mind, he either expressed in the form of a concrete code statement ($\text{MOTIVE}^{\text{SAttr}}$ = Realize Idea) or in form of creating method skeletons ($\text{MOTIVE}^{\text{SAttr}}$ = Create Artifact). With that, he either realized the upcoming work step ($\text{IMPACT}^{\text{SAttr}}$ = Step Forward) or apparently accelerated the work process by inserting the solution of a problem the DRIVER had been thinking about ($\text{IMPACT}^{\text{SAttr}}$ = Accelerate Next Step). The ACTIVE OBSERVER also jumped in to help the DRIVER with cleaning up code ($\text{MOTIVE}^{\text{SAttr}}$ = Clean Code Flaw), resulting in a tidier artifact ($\text{IMPACT}^{\text{SAttr}}$ = Cleaner Code) as the base for the DRIVER's subsequent work.

Announcement and awareness

The ACTIVE OBSERVER did not provide any verbal clues when jumping in the editor (ANNOUNCEMENT STYLE^{SAttr} = Silent). While the DRIVER verbalized his thoughts, questions, and considerations alongside his own activities (AWARENESS VOICING^{SAttr} = Thinking Aloud), the ACTIVE OBSERVER only rarely gave verbal awareness indicators about what he was doing – he either mumbled to himself (AWARENESS BRIDGING^{SAttr} = Mumbling) or did not say anything at all (AWARENESS BRIDGING^{SAttr} = None). During all JUMP IN activities, the *Follow Mode* of *Saros* was enabled (AWARENESS FEATURE^{SAttr} = Follow Mode Used).

Development focus and activity

JUMP IN occurred either during the development of tests (DEVELOPMENT FOCUS^{SAttr} = Testing, SHARED-MIND ACTIVITY^{SAttr} = Write Test) or while the pair actively coded the solution (DEVELOPMENT FOCUS^{SAttr} = Implementation, SHARED-MIND ACTIVITY^{SAttr} = Solution Development).

Visual and mental coupling

When the ACTIVE OBSERVER sprang into action, this happened in the editor view of *Eclipse* (PLACE OF ACTION^{SAttr} = Editor) and within the same viewport as the DRIVER (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Viewport). The participants were mutually aware of what the other one was concerned with (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral) and the DRIVER fully engaged in the ACTIVE OBSERVER's activity (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain).

Process fluency

No awareness or coordination problems occurred (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed) and the DRIVER always recognized the jumping in of the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware). He integrated the ACTIVE OBSERVER's work in his train of thought (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate) and waited until the DRIVER was done (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Temporizing) before resuming the editing process.

Discussion of research questions

JUMP IN is an *Editing-Freedom* behavior. The overall process appeared smooth – it was free of coordination problems and neither *Reduced-Workspace-Awareness* nor *Reduced-Physical-Awareness* seemed to come into effect. The ACTIVE OBSERVER stayed within the shared viewport in the editor, and the DRIVER was aware of the ACTIVE OBSERVER's changes. Accordingly, the pair did not lack the review effect nor did they mentally decouple. JUMP IN activities were realizations or continuations of the DRIVER's thoughts and did not change his train of thought. Therefore, JUMP IN potentially provides a beneficial possibility for the active involvement of the observer in the shared problem solving process. This is why JUMP IN is considered as *Freedom-Positive-Effect*.

PP context

In the local PP setting, these supportive contributions could be emulated with a quick succession of two driver changes. Obviously, this is more cumbersome unless there are two keyboards and two mice (as in ePP, see page 42).

eDPP context

Behavior similar to JUMP IN is mentioned in Chong and Hurlbutt (2007), where the authors observed a team doing PP with dual keyboard and mouse setting, having their focus on the frequency of role-switches. They found that one reason for role switches was hesitation of the driver: *"By convention, the programmers refrained from typing when their partner was actively typing, but they frequently jumped in during pauses or periods of hesitation"*. So, wisely used, JUMP IN activities in eDPP may be a minimal-invasive way to facilitate the natural desire of jumping in to help bringing forward the joint solution process.

4.5.3 eDPP Phenomenon: Check

In a nutshell: Check

Self-motivated verification by the Active Observer regarding an assumption of the Driver or a personal uncertainty.

The Active Observer sometimes **used his viewing freedom to check something** related to what the Driver was doing or just did, such as when he recognized an uncertainty of the Driver who, nevertheless, continued to work. This was a verification of the current work step of the Driver. In such a Check activity that aimed at validating what the Driver was about to do, the Active Observer used his independence to look up something on the web. For example, he checked whether the Driver really declared all required methods for a specific design pattern interface or whether he really threw the correct exception type for a method.

In other cases, the Active Observer himself was unsure with regard to a proposal of the Driver and decided to clarify this before giving his okay. For example, he quickly checked back in a test class before giving his okay for the name proposed by the Driver, or he revisited the requirements specification before agreeing with the next work step suggested by the Driver. For the Driver, these proposals were more or less rhetoric questions where he did not wait for a reply but continued his work. Nevertheless, the Active Observer used his viewing freedom to check it anyway.

Such a behavior of the Active Observer is called Check and is of **reassuring character**.

Example: Check

The pair was implementing the *Observer* design pattern. The Driver put the required method declarations in a newly created interface and decided to be done with it. In the first instance, the observer agreed with the Driver, but then he was uncertain whether they really put in all required methods.

While the Driver already was concerned with the next step (implementing the class for the observer interface), the Active Observer visited Wikipedia[a] and looked up the relevant method declarations for the interface. Since the Active Observer informed the Driver about his doubts and his activity, a short discussion started based on what the Active Observer looked up. Finally, both participants agreed that they had declared all required methods, and thus, were done with this step.

The Driver then continued the implementation of the observer class.

[a] `http://en.wikipedia.org/wiki/Observer_pattern` (accessed November 11, 2017)

Example: Check

The Driver implemented a validity check for specific file types. This involved several aspects to check for a file. At some point, he proposed the next aspect to check: *"So, now we have to do this with this strange time frame."* Before the Active Observer agreed to this step, he hesitated: *"Hm, yes that is just the question . . ."* (elongated pronunciation of the word question) while he quickly visited the requirements specification and only then gave his okay *". . .but yes, nope, it's correct"*.

Motive and impact

When the ACTIVE OBSERVER noticed an uncertainty during their work or found himself not quite sure regarding some aspect, he used his viewing freedom to look up something (MOTIVE^{SAttr} = Reassure). This resulted in gaining clarity for himself (IMPACT^{SAttr} = Unilateral Verification) or in sharing and discussing the issue with the DRIVER (IMPACT^{SAttr} = Clarification Episode).

Announcement and awareness

The announcement behavior of the ACTIVE OBSERVER for his CHECK activities varied greatly. Often he did not give any verbal hint when starting his activities (ANNOUNCEMENT STYLE^{SAttr} = Silent). Sometimes he explained what he was going to do (ANNOUNCEMENT STYLE^{SAttr} = Explicit). In other cases he just expressed his own lack of knowledge before he started looking up the respective issue (ANNOUNCEMENT STYLE^{SAttr} = Implicit). His verbal feedback during his CHECK activities was also quite mixed. He either did not give any verbal feedback (AWARENESS BRIDGING^{SAttr} = None), verbalized his thoughts (AWARENESS BRIDGING^{SAttr} = Thinking Aloud), or he informed the driver about a finding just after he had looked it up (AWARENESS BRIDGING^{SAttr} = Inform Finding). Or, in other cases, merely his unassertive reaction in form of an elongated pronunciation was an indicator that he was in a thinking process (AWARENESS BRIDGING^{SAttr} = Unassertive Reaction).

The DRIVER in turn verbalized his train of thought (AWARENESS VOICING^{SAttr} = Thinking Aloud) either alongside his own work or related to the information from the ACTIVE OBSERVER's look-up. Usually, the Follow Mode of *Saros* was enabled during CHECK phenomena (AWARENESS FEATURE^{SAttr} = Follow Mode Used) or it stopped automatically (AWARENESS FEATURE^{SAttr} = Follow Mode Stopped) when the ACTIVE OBSERVER switched to another artifact for his CHECK activity.

Development focus and activity

CHECK activities occurred when the pair was actively coding the problem solution (DEVELOPMENT FOCUS^{SAttr} = Implementation, SHARED-MIND ACTIVITY^{SAttr} = Solution Development), when they were were constructing the structure and relationships of the module's artifacts (DEVELOPMENT FOCUS^{SAttr} = Local Design, SHARED-MIND ACTIVITY^{SAttr} = Decide Artifact Structure), as well as when they tested their implementation (DEVELOPMENT FOCUS^{SAttr} = Testing, SHARED-MIND ACTIVITY^{SAttr} = Write Test).

Visual and mental coupling

The ACTIVE OBSERVER either checked back in the editor view (PLACE OF ACTION^{SAttr} = Editor) but in a different artifact than the DRIVER was residing in (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Artifact Different). Or he consulted his web browser (PLACE OF ACTION^{SAttr} = Web Browser) and thus used a different application than the DRIVER (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Different Application). Notwithstanding the visual decoupling, the participants did not mentally decouple. And although the DRIVER did not know the details of the ACTIVE OBSERVER's short look-up trip (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral+), they were still jointly concerned with the same aspect (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain) of their current work step. When the ACTIVE OBSERVER involved the DRIVER in his clarification process, both had quite a clear understanding of the other one's subject of contemplation (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral).

Process fluency

When the DRIVER did not recognize the ACTIVE OBSERVER's parallel checking actions because they were out of his sight (FLUENCY^{CAttr}:Recognition = Not Possible), he continued his work (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Not Possible). There was no evidence that they influenced his train of thought (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Not Observed). Even when the DRIVER did not recognize the CHECK activity, it occurred that they resulted in suggestions by the ACTIVE OBSERVER which then revealed that the ACTIVE OBSERVER had clarified something 'in the background' (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Downstream) and the DRIVER adapted his current work step (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Correct).

When the DRIVER recognized the look-up actions of the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware), he paused his work and engaged in the clarification discourse (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Trip in Tactic). The pair never had to explicitly synchronize in the course of a CHECK phenomenon (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed).

Discussion of research questions

CHECK is a phenomenon belonging to *Viewing-Freedom*. Neither *Reduced-Physical-Awareness* nor *Reduced-Workspace-Awareness* lead to observable process problems. It rather appears that the independent workspaces with unshared areas and the *Viewing-Freedom* are a profitable combination for this kind of phenomenon. It enables the ACTIVE OBSERVER to perform his observing job more actively in cases where he needs additional resources (like a web browser or other artifacts) without necessarily having to interrupt the DRIVER. Due to the possibility of such unobtrusive action taking resulting in quality enhancements of the current or subsequent work steps, CHECK is considered as *Freedom-Positive-Effect* for the observed pair.

PP context

In PP, this situation would be possible if the ACTIVE OBSERVER had a laptop computer alongside. However, even then the driver would see the actions of the ACTIVE OBSERVER and would very likely pay some attention to it. When sharing one computer, such actions of the driver would be much more disruptive for the driver. It is questionable whether the observer would interrupt the driver at all for quite small checks.

eDPP context

The ACTIVE OBSERVER always returned to the shared viewport in the editor when he finished his generally rather short CHECK activities. For longer clarification episodes, the DRIVER joined the ACTIVE OBSERVER's considerations because the ACTIVE OBSERVER kept him in the loop about unilaterally visible information. The pair did not mentally decouple and the PP benefits 'shared problem solving' and 'review driver's work' seemed not to be harmed.

The support of CHECK has been discussed by Reeves and Zhu (2004) as a potential benefit of relaxed WYSIWIS groupware: *"The model is necessary to allow the document's participants the freedom to browse its contents, while not disrupting their pair's position. For example, while pair programming one user may check the implementation of method, while their partner continues to code."* CHECK phenomena are the actual use of this relaxed WYSIWIS potential.

4.5.4 eDPP Phenomenon: Contribution

In a nutshell: Contribution

The ACTIVE OBSERVER contributes an idea of his own directly related to the DRIVER's current task.

Sometimes the ACTIVE OBSERVER did not assist the DRIVER's current work by performing an action but rather contributed an idea of his own in the context of the DRIVER's work. These changes comprised the moving of code parts, writing a comment for memorizing a thought, creating variables, or complementing the statement currently written by the DRIVER. Thus, a CONTRIBUTION can but does not necessarily have to relate to the identical aspect the DRIVER is currently concerned with. Nevertheless, it is a consideration of the ACTIVE OBSERVER that is directly related to the pair's current work step.

Such an action is called CONTRIBUTION. It means that the ACTIVE OBSERVER **contributed an own thought to the Driver's work that did not aim at correcting or challenging him but to realize a separate idea** the DRIVER did not think of.

Example: Contribution

The ACTIVE OBSERVER recognized a property they had to implement for a method. In order not to forget this idea he told the DRIVER he would like to make a note and promptly added a comment in the method to remember his thought.

Motive and impact

Such a complimentary idea was of substantial character for the solution process like noting down a finding or adding a variable (MOTIVE^{SAttr} = Realize Idea), or it was of code improvement character (MOTIVE^{SAttr} = Clean Code Flaw). Often, a CONTRIBUTION entailed a clarification episode (IMPACT^{SAttr} = Clarification Episode) where the participants resharpened their understanding regarding the aspect raised by the ACTIVE OBSERVER. Or the CONTRIBUTION put the solution process a step forward by immediately complementing the DRIVER's code (IMPACT^{SAttr} = Step Forward).

Announcement and awareness

CONTRIBUTIONs were announced either explicitly (ANNOUNCEMENT STYLE^{SAttr} = Explicit) or the ACTIVE OBSERVER was mumbling to himself before starting his activity (ANNOUNCEMENT STYLE^{SAttr} = Mumble). During his activity, the clarity of the verbal utterances of the ACTIVE OBSERVER were mixed, he either verbalized his thoughts (AWARENESS BRIDGING^{SAttr} = Thinking Aloud), mumbled something to himself (AWARENESS BRIDGING^{SAttr} = Mumbling), or said nothing at all (AWARENESS BRIDGING^{SAttr} = None). However, the DRIVER was verbalizing his thoughts throughout all CONTRIBUTION episodes (AWARENESS VOICING^{SAttr} = Thinking Aloud). In most cases, the *Follow Mode* was used (AWARENESS FEATURE^{SAttr} = Follow Mode Used) or it was technically stopped because the ACTIVE OBSERVER detached from the DRIVER's viewport for his independent activity (AWARENESS FEATURE^{SAttr} = Follow Mode Stopped).

Development focus and activity

CONTRIBUTION phenomena happened while the participants worked on the production code (DEVELOPMENT FOCUS^{SAttr} = Implementation) and were commonly engaged in problem solving (SHARED-MIND ACTIVITY^{SAttr} = Solution Development).

Visual and mental coupling, process fluency

The contributions were always in the editor (PLACE OF ACTION^{SAttr} = Editor), either within the same editing area (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Line || Statement || Shared Viewport) or in another artifact (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Artifact Different). For the latter case they were not verbally announced, and the DRIVER had no clue of the ACTIVE OBSERVER's action (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Unilateral Unaware) until he observed some changes in the workspace (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Downstream). Then he was confused and explicitly asked the ACTIVE OBSERVER what he had been doing (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Ping Doing Past). This happened when the DRIVER was editing a class that implemented an interface. The ACTIVE OBSERVER moved the declaration of a protected method of that class to the interface. Since the declaration of protected methods is not allowed in Java interfaces, the DRIVER was confused by an error annotation he got from the IDE. The ACTIVE OBSERVER in turn had his mental focus on a different aspect of the current work step than the DRIVER (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = Fork Return) for a short time (less than a minute).

In all other cases, the DRIVER recognized the ACTIVE OBSERVER's activity (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware) and in general the pair stayed mentally coupled (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral, COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = OneBrain || One Brain μ). Moreover, there appeared to be no process repair (FLUENCY^{CAttr}:Synchronization = Not Observed).

CONTRIBUTION activities sometimes interrupted the DRIVER's work but did not essentially change his direction of thought. Often he paused his own work and engaged in the action initiated by the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Pause Join In). When the CONTRIBUTION was a supplement of the DRIVER's code, he continued his work (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Continue) while seamlessly integrating the CONTRIBUTION to his considerations (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate). They were either a clarification of an aspect of his work (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Trip In Tactic) or they led to a short thematic excursus (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Short Digression). Only once the DRIVER rejected the ACTIVE OBSERVER's intervention (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Reject) and unperturbedly continued his work (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Not Observed). This happened when the ACTIVE OBSERVER was about to slide in the DRIVER's editing activity because he had the idea to create two arrays. The DRIVER, however, did not want to do that at this point.

Discussion of research questions

CONTRIBUTION is an *Editing-Freedom* phenomenon. Inappropriately used, it can easily lead to confusion on the part of the DRIVER. This may happen due to the *Reduced-Workspace-Awareness* and the *Reduced-Physical-Awareness* when he has no clue about the CONTRIBUTION but suddenly observes changes in the workspace. Then the CONTRIBUTION would require synchronization effort interrupting the DRIVER's train of thought. In this case it would be an example of *Freedom-Negative-Effect*. This, however, has not been observed in the data.

PP context

Compared to PP, CONTRIBUTION allows equal participation for both participants in the solution solving process. If done well, like with DIRECT FIX and JUMP IN, it lowers the hurdle for the ACTIVE OBSERVER to contribute to the process. It then appears to lead to higher process smoothness by avoiding the negotiation required for a driver change. Since they handled it wisely, CONTRIBUTION counts as *Freedom-Positive-Effect* for the observed pair.

eDPP context

By facilitating the contribution of ideas for the ACTIVE OBSERVER, CONTRIBUTION may improve the brain synthesis effect of PP, which has been mentioned in Section 1.4.1.4 "Principles for Good Pair Programming": *"a pair will come up with more than twice as many possible solutions as two individuals working alone. They will then proceed to more quickly narrow in on the best solution and will implement it more quickly and with better quality"* (Williams and Kessler, 2000).

4.5.5 eDPP Phenomenon: Local Solution

In a nutshell: Local Solution

A LOCAL SOLUTION is an accomplishment of a self-contained subtask by the ACTIVE OBSERVER, embedded in the DRIVER's work.

In some cases, the next step in the solution process was quite a small, **somewhat self-contained subtask for which the motivation or expertise of the** ACTIVE OBSERVER **was more advantageous than that of the** DRIVER. Hence, the ACTIVE OBSERVER relieved the DRIVER of this subtask by undertaking it himself. When the ACTIVE OBSERVER had finished, the control went back to the DRIVER. The delegated subtasks ranged from looking up how to solve a specific problem up to implementing small methods. For this kind of subroutine-calls, the DRIVER did not feel responsible for this subtask.

This capability-driven engagement of the ACTIVE OBSERVER is called LOCAL SOLUTION.

Example: Local Solution

The pair was about to sort a list of files and needed a file comparator object. Without further ado, the ACTIVE OBSERVER started to search the web for an example implementation of a file comparator. He came up with one, pasted it into the source code, adapted it to the current needs, and gave the control back to the DRIVER who then continued his work.

Motive and impact

LOCAL SOLUTION episodes always started because the pair was faced with a problem ($\text{MOTIVE}^{\text{SAttr}}$ = Faced with Problem) and immediately resulted in the solution provided by the ACTIVE OBSERVER ($\text{IMPACT}^{\text{SAttr}}$ = Step Forward) or in a shared clarification of an aspect with the ACTIVE OBSERVER as the knowledge provider ($\text{IMPACT}^{\text{SAttr}}$ = Clarification Episode).

Announcement and awareness

The pair more or less explicitly agreed on the local role switch. The ACTIVE OBSERVER either took the lead in solving a problem implicitly, *"There is certainly something [...]"*, and started typing right away ($\text{ANNOUNCEMENT STYLE}^{\text{SAttr}}$ = Implicit), mumbling to himself while doing so ($\text{AWARENESS BRIDGING}^{\text{SAttr}}$ = Mumbling). In the meantime, the DRIVER was silently waiting for his input ($\text{AWARENESS VOICING}^{\text{SAttr}}$ = None).

It also occurred that he explicitly took over the responsibility, *"I will consider something"*, but actually started to search for a solution later in the process. In that case, he did not inform the DRIVER about his activity (ANNOUNCEMENT STYLE^{SAttr} = Silent) and thus the DRIVER continued with his work while verbalizing his current train of thought (AWARENESS VOICING^{SAttr} = Thinking Aloud). When the ACTIVE OBSERVER found what he was looking for, he informed the DRIVER about it (AWARENESS BRIDGING^{SAttr} = Inform Finding). LOCAL SOLUTION phenomena occurred either with *Follow Mode* enabled (AWARENESS FEATURE^{SAttr} = Follow Mode Used) or when it was already disabled (AWARENESS FEATURE^{SAttr} = Follow Mode Disabled).

Development focus and activity

LOCAL SOLUTION phenomena always happened when the participants were working on the problem solution (DEVELOPMENT FOCUS^{SAttr} = Implementation, SHARED-MIND ACTIVITY^{SAttr} = Solution Development).

Visual and mental coupling, process fluency

For LOCAL SOLUTION activities, the ACTIVE OBSERVER either only used his web browser (PLACE OF ACTION^{SAttr} = Web Browser, COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Different Application) or both, his web browser and then the shared editor (PLACE OF ACTION^{SAttr} = Editor) within the same focused area as the DRIVER (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Statement).

The DRIVER recognized the activity of the ACTIVE OBSERVER either immediately or with a short delay (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware || Downstream). Since LOCAL SOLUTION activities were rather short and noticed by the DRIVER, the pair members were more or less mutually aware of what the other one was concerned with (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral || Bilateral+).

LOCAL SOLUTIONs were either directly related to the DRIVER's work (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain) or the ACTIVE OBSERVER additionally thought about an issue the DRIVER was currently not concerned with (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = Different Aspect).

During a LOCAL SOLUTION that was related to the DRIVER's work, he waited for the ACTIVE OBSERVER's input (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Temporizing) and afterwards continued his work based on that input (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate). Or he stopped his own train of thought (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Trip In Tactic) and joined the considerations of the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Pause Join In). Notable synchronization effort was not observed (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed). When the DRIVER was waiting for the ACTIVE OBSERVER's solution, he wanted to stay in the loop by cautiously asking for an update about the findings of the ACTIVE OBSERVER (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Ping Update).

Discussion of research questions

LOCAL SOLUTION was observed for rather small subtasks for which the participants felt and agreed that the observer had more expertise. It is a phenomenon enabled by *Editing-Freedom* and *Viewing-Freedom*. Conflicts or negative consequences resulting from it are easily conceivable when the ACTIVE OBSERVER over-stresses his freedom by overly distracting the DRIVER in a know-it-all manner. In this form it would be a *Freedom-Negative-Effect*.

For the observed pair, however, the DRIVER seemed to be fine with this kind of local role switch and no notable synchronization effort was noticed. Accordingly, LOCAL SOLUTION is considered as *Freedom-Positive-Effect*.

PP context

In local PP, the DRIVER could pass the respective subtask to the partner, thereby performing a role switch. The execution of the driver change as such is of less importance – the episode is long enough to amortize its effort. Rather, the one hand, the hurdle for the motivated observer is lower because he can jump right into action. On the other hand, the DRIVER can hand over responsibility without having to explicate it.

As mentioned in Section 1.4.1.2 "Appropriateness of Pair Programming", Plonka et al. (2012b) observed that pair developers paired up to combine their complementary competence for solving a task and that

they split up for sub-tasks according to their competence. Local Solution is a way to realize this desire without explicitly having to split.

eDPP context The work mode can be considered as a fine-grained, expertise-driven work allocation where the participants can easily combine their complementary competence. The observing duty of the Active Observer fades and the Driver's support is foregrounded. For the observed pair, this was a *Freedom-Positive-Effect* leading to a more dynamic and efficient process.

In PP, Local Solution behavior also seems to be a natural collaboration pattern: *"In the course of completing their tasks, programmers often temporarily deferred control of the keyboard to their partners when they knew that their partner was more practiced in a particular subtask, such as using a particular feature or plug-in of the IDE"* (Chong and Hurlbutt, 2007). In eDPP, this behavior can be acted out more easily and thus, if used in a sensible way, the synergetic effect of the combined expertise potentially manifests better.

4.5.5.1 New Role: Supplier

A Local Solution episode can be considered as a local role switch.The Active Observer did not assume the overall editing process. Moreover, he relieved the Driver of a specific subtask which was still embedded into the overall editing process of the Driver. In such cases, the Active Observer is a kind of Supplier for the Driver. The Driver in turn puts down his driving role for this subtask.

4.6 Results Relating to a Separated Work Mode

4.6.1 eDPP Phenomenon: Parallel Exploration

In a nutshell: Parallel Exploration

In Parallel Exploration episodes, pair members independently move in their workspaces, performing actions relating to the same issue.

When the pair, or one of it's members, was faced with an issue to which there was no immediate solution, **both participants independently explored the problem**. None of them wrote code but they independently moved in their workspaces at the same time (in shared as well as in unshared viewports). This happened, for example, when the participants were unsure about a requirement as well as when they were checking for specific information. Or when they moved through existing code in order to understand its meaning or to localize defects. Then, both independently went through the code in their own time and manner. Beyond that, Parallel Exploration also occurred for tool-related issues. For example, one participant was searching for a specific feature in his IDE and communicated this to the other who did not know the answer either but also started to search in his IDE. The dialog of this situation is given in the following box.

Such phenomena where **the pair separately performed actions aiming at the same goal** are referred to as Parallel Exploration.

Example: Parallel Exploration

> DRIVER: *"So, however, here I can . . .[opens the context menu of an implemented interface name]" "Can I create a new class that automatically implements the interface here? So with just a simple key-shortcut, is that possible?"*
> ACTIVE OBSERVER: *"Good question, let me see [and also opens his context menu of the interface name]."*
> DRIVER: *"Implementation, that ought to be possible somewhere."*
> ACTIVE OBSERVER: *"You're right. Ehm . . .no, I can't see it now."* DRIVER: *"Okay, then I'll just do it."*

Motive and impact

PARALLEL EXPLORATION also occurred when the pair was about to finish for the day (MOTIVE^{SAttr} = End Session). They independently browsed their day's work, doing a kind of wrap-up (IMPACT^{SAttr} = Dialog). In all other cases, PARALLEL EXPLORATION was driven by a problem the pair was faced with (MOTIVE^{SAttr} = Faced with Problem) which they jointly but separately examined (IMPACT^{SAttr} = Clarification Episode).

Announcement and awareness

Some PARALLEL EXPLORATION episodes started with an explicit announcement (ANNOUNCEMENT STYLE^{SAttr} = Explicit). In such cases, either one participant declared his exploring activity like *"Wait, I shortly look at the requirements"* and the other one joined him, or one participant explicitly requested the other one to follow him *"Go to it [meant a directory on a server]"*. In other cases, they started in an indirect fashion (ANNOUNCEMENT STYLE^{SAttr} = Implicit). Then the pair talked about the examined issue, but none of them explicitly mentioned they had started exploring.

The independent movements of both participants would not have been possible with *Saros*' *Follow Mode* enabled. Either it was already disabled before the episode started (AWARENESS FEATURE^{SAttr} = Follow Mode Disabled), it was technically stopped due to the autonomous movements of the follower (AWARENESS FEATURE^{SAttr} = Follow Mode Stopped), or the participants manually stopped it when they started their separate workspace browsing (AWARENESS FEATURE^{SAttr} = Follow Mode Manually Stopped). Only when the exploration happened in unshared workspace areas like IDE menus, the *Follow Mode* remained enabled (AWARENESS FEATURE^{SAttr} = Follow Mode Used), but in fact without being relevant to the activity.

During their PARALLEL EXPLORATIONs, the pair always actively kept each other in the loop regarding their findings and train of thoughts (AWARENESS BRIDGING^{SAttr} = Thinking Aloud, AWARENESS VOICING^{SAttr} = Thinking Aloud).

Development focus and activity

PARALLEL EXPLORATION phenomena mostly happened while implementing the solution (DEVELOPMENT FOCUS^{SAttr} = Implementation), where the pair had to clarify some issues regarding their current work or their tool (SHARED-MIND ACTIVITY^{SAttr} = Clarification). They also occurred during the testing phase (DEVELOPMENT FOCUS^{SAttr} = Testing) for locating the root cause of a failing test (SHARED-MIND ACTIVITY^{SAttr} = Defect Localization). Moreover, when the pair was about to end their workday, they independently browsed their work, reflecting about its design (DEVELOPMENT FOCUS^{SAttr} = Local Design, SHARED-MIND ACTIVITY^{SAttr} = Reflect Result).

Start and termination

PARALLEL EXPLORATION episodes mostly began because one participant started the exploration (START MODE^{SAttr} = Issue Unilateral Start). However, it also occurred that one participant explicitly requested the other one to go to an unshared view (START MODE^{SAttr} = Issue Unilateral Request). The pair ended the separate work mode usually because the issue was solved (TERMINATION MODE^{SAttr} = Issue Resolved) or, rather rarely, because they turned their attention to a different topic (TERMINATION MODE^{SAttr} = Different Topic).

Visual and mental coupling

Task-related PARALLEL EXPLORATIONs happened mainly within the editor view (PLACE OF ACTION^{SAttr} = Editor) and the pair often explored the same or different shared artifacts (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Artifacts Independent). They also separately used their (unshared) terminal (PLACE OF ACTION^{SAttr} = Terminal) but went to the same directory on a

server (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Unshared Application Same Contents). When searching a specific functionality in the IDE, they independently explored their (unshared) context menus (PLACE OF ACTION^{SAttr} = Context Menu, COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Unshared View).

In all instances, both participants were mutually aware of the other one's exploration (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral). Although they were not mutually aware of the explicit position or detailed actions of each other, both were concerned with the same aspect and continuously exchanged their thoughts (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain).

Process fluency and getting together in the workspace

Both participants were in the PARALLEL EXPLORATION work mode (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware, FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Parallel Handling). Task-related PARALLEL EXPLORATIONs were part of the joint reasoning in the pair's current work step (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate). Tool-related PARALLEL EXPLORATIONs led to a short non-task related digression of both participants (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Short Digression).

When the pair was done with their separate exploration, they met again in the workspace in different ways. Often, they closed their separate views or stopped their independent movements and met where they had left (JOIN MODE^{SAttr} = Meet in Viewport). In other cases, they used *Saros' Jump-To-Position* feature (JOIN MODE^{SAttr} = Jump to Position). When both were in an unshared application, at the end, one participant waited for the other one in the shared workspace and signaled that he was waiting by asking his partner whether he was also back in the shared workspace (JOIN MODE^{SAttr} = Explicit, FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Ping Presence). The other one was not back but his typing could be heard. The waiting participant was confused, and asked the other one what he was doing (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Ping Doing) (the other one was typing a message to a colleague on a different computer he had aside). In the majority of cases, however, no further explicit synchronization effort was observable during or while finishing a PARALLEL EXPLORATION episode (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed).

Discussion of research questions

PARALLEL EXPLORATION is a phenomenon of the type *Viewing-Freedom* with mixed dispositions. When it involves unshared views, *Reduced-Workspace-Awareness* negatively comes into effect. In this case, PARALLEL EXPLORATION is a workaround in the sense that the participants are not able to jointly perform this activity. On the other hand, as in PP, the exploration of shared artifacts could have been done together using *Saros' Follow Mode* feature, but, anyway, the participants decided to do it separately.

PARALLEL EXPLORATION may be seen as a workaround due to *Reduced-Workspace-Awareness*. *Freedom-Negative-Effect* can easily slip in such a separate working mode. However, as seen in the data, it may be beneficial for the exploration of issues and can be considered as *Freedom-Positive-Effect*.

PP context

If intentionally used, for example to look up things, PARALLEL EXPLORATION is possible in PP with an additional laptop aside, too. The situation becomes somewhat more cumbersome if the most recent changes in the jointly edited artifacts are relevant. Then, the changes have to be committed on the joint machine and updated on the laptop aside. The challenge of exploring things separately simply because they are not shared does not exist in PP.

Williams et al. (2000) and Hulkko and Abrahamsson (2005) found that even in the local setting PP developers prefer to be undisturbed while reading or trying to understand problems, to do it in their own speed and line of thought (see Section 1.4.1.2 "Appropriateness of Pair Programming"). It is plausible but not verified that PARALLEL EXPLORATION facilitates this natural desire. In that sense PARALLEL EXPLORATION would count as *Freedom-Positive-Effect*.

eDPP context

In most cases, PARALLEL EXPLORATION episodes did not involve explicit synchronization effort. Additional verbalizations alongside their actions did not seem to require extra cognitive load for the participants, either. Maybe they were even helpful in the sense that they potentially increased

the cognitive performance because a *"proven instructional technique for improving comprehension is think-aloud"* (Duke and Pearson, 2002).

For the observed pair, PARALLEL EXPLORATION episodes were nearly always used for task-related problems which were successfully solved. They did not seem to be an issue for the process. At least for the observed pair, no *Freedom-Negative-Effect* was observed. Nevertheless, no final statement can be made about whether PARALLEL EXPLORATION is a technically induced workaround due to *Reduced-Workspace-Awareness* or whether it counts as *Freedom-Positive-Effect* because it facilitates natural behavior in exploration tasks.

4.6.2 eDPP Phenomenon: Unilateral Operation

In a nutshell: Unilateral Operation

A UNILATERAL OPERATION is an operation of an unshared dialog, menu command, or alike, done by one participant where the remote partner can only see the results of that operation appearing in the workspace.

While a unilaterally visible dialog was operated, the remote partner did not see anything happening in his workspace. He could not review the dialog inputs and just saw the effects of the operation appear in his workspace afterwards, like for example a new class including auto-generated methods.

In contrast to direct manipulation of artifacts by direct editing, it is also possible to indirectly manipulate artifacts, using symbolic actions like *"menu commands, buttons, toolbars, and keyboard shortcuts [...] Symbolic manipulation lets users interact with artifacts in ways that are often not possible in the real world"* (Gutwin and Greenberg, 1998), because changes are applied automatically and simultaneously to several artifacts at once. By nature, they do not provide details about the action's author, their appearance and their progress (Gutwin and Greenberg, 1998).

In a synthetic workspace like in PP and eDPP, some symbolic actions could be avoided, but others not. A refactoring, for example, can be done via an IDE dialog or manually. In the latter case, the stepwise changes in the artifact would be transparent to the remote partner. When using the refactoring dialog of the IDE, however, the changes appear all at once (in one or more artifacts). Some operations, i.e. creating artifacts, can only be done via symbolic manipulations, though. In any case, such symbolic manipulations are unilaterally visible actions that result in a blind period and a possible surprise effect for the remote partner.

Such **unilaterally visible operations that involve complex dialogs** are phenomena of the type UNILATERAL OPERATION.

Quite short symbolic actions involving merely one input do not count as UNILATERAL OPERATION phenomena, like for example a renaming dialog. They are included in other phenomena types where not the unilateral dialog usage appeared to be the predominant aspect but the type of unilateral activity, like for example a CONTRIBUTION.

The following example is a unilateral operation combined with awareness bridging using the *New Package* dialog of *Eclipse*:

Example: Unilateral Operation

DRIVER: [selects a method] *"OK, now we should write a test case."*
ACTIVE OBSERVER: *"Yes."*
DRIVER: *"That tests just this function"*
ACTIVE OBSERVER: *"Yes."*
DRIVER: *"You are with me, right?"*
ACTIVE OBSERVER: *"Yes."*
DRIVER: *"Then we create a test."* [goes to package explorer] *"Oh, that's missing."* [opens *New Source Folder* dialog and mumbles:] *"source folder"* [types: 'test' in input field for folder name and mumbles:] *"test"* [hits enter to create the folder, then opens new package dialog and mumbles:] *"package"* [mumbles as he types into the *Name* input field:] *"We wanted to call it Transcoder, didn't we? So that I name package correct for the test."*
ACTIVE OBSERVER: *"Here, we wrote it down somewhere."*
DRIVER: *"Oh yes, got it, the TODO down here: newstranscoding"*
DRIVER: [verbalizes typing] *"news-trans-co-ding"*

Motive and impact

For the observed pair, the DRIVER used complex unilateral dialogs because he needed to create some artifacts like classes, interfaces, or test cases (MOTIVE^{SAttr} = Create Artifact). Accordingly, UNILATERAL OPERATION phenomena resulted in new artifacts that were required for the upcoming work step (IMPACT^{SAttr} = Step Forward).

Announcement and awareness

The participant who was about to use a unilaterally visible dialog either explicitly announced that he was going to create a specific artifact (ANNOUNCEMENT STYLE^{SAttr} = Explicit), or he mentioned that they needed one and started the UNILATERAL OPERATION right away (ANNOUNCEMENT STYLE^{SAttr} = Implicit). The participant performing the unilaterally visible operation steadily verbalized his considerations as well as relevant dialog options and his inputs (AWARENESS BRIDGING^{SAttr} = Thinking Aloud) while the other one behaved rather biding, responding to the other one's questions (AWARENESS VOICING^{SAttr} = React). Only when the unilateral dialog involved inputs that required some clarification, he also verbalized his considerations (AWARENESS VOICING^{SAttr} = Thinking Aloud). During all UNILATERAL OPERATION episodes, *Saros*' *Follow Mode* was enabled, although it could only come into effect once the unilateral operation was finished.

Development focus and activity

UNILATERAL OPERATIONs mostly occurred when the pair was writing tests (DEVELOPMENT FOCUS^{SAttr} = Testing, SHARED-MIND ACTIVITY^{SAttr} = Write Test). It also appeared when they were writing production code (DEVELOPMENT FOCUS^{SAttr} = Implementation, SHARED-MIND ACTIVITY^{SAttr} = Solution Development) or designing their artifacts' structure (DEVELOPMENT FOCUS^{SAttr} = Local Design, SHARED-MIND ACTIVITY^{SAttr} = Decide Artifact Structure).

Occurrence and detection of faults

Never the dialog operations of the DRIVER were sneaking solo runs, they were always directly or indirectly jointly agreed upon beforehand (START MODE^{SAttr} = Execute Discussed). Nevertheless, UNILATERAL OPERATIONs have the inherent problem of missing the review effect for the unilateral inputs. This in turn may result in problems and defects. The DRIVER actually made a typo when using a unilateral dialog (FAULT^{SAttr} = Typo) or did not adequately use the options of a dialog (FAULT^{SAttr} = Incorrect Operation). The typo in the appearing artifact name was immediately recognized by the partner as soon as it appeared in his workspace (FAULT RECOGNITION^{SAttr} = Immediately, by Observer) and then directly fixed by the DRIVER. The new artifact resulting from the incorrect operation of the dialog was not correctly named and did not contain the proper auto-generated methods. This was recognized by the DRIVER after a short moment (FAULT RECOGNITION^{SAttr} = Delayed, by Driver) and then corrected by the DRIVER by using the dialog once again but with proper inputs and selections. Apart from that, the vast majority of UNILATERAL OPERATION episodes were free of faulty inputs (FAULT^{SAttr} = None).

Unilateral Operations always happened because the Driver used a dialog (Place of Action^{SAttr} = Dialog) and thus were in an unshared part of the workspace (Coupling of Participants^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Unshared View). Nevertheless, the pair was aware of this limited sharing (Fluency^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware) and stayed mentally coupled: The Driver accomplished the unilateral operation alone while the partner simply waited for the result to appear, i.e. the new class, (Fluency^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Temporizing). Without exception, the pair members remained mentally concerned with the same aspect (Coupling of Participants^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = One Brain) and mutually aware of that (Coupling of Participants^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral). The creation of artifacts via Unilateral Operations were part of the current work step (Fluency^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate). Once the Driver pinged the observer whether he was with him (Fluency^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Ping Focus), but in all other cases, no synchronization effort was observable (Fluency^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed). Process fluency and coupling

A Unilateral Operation cannot happen in local PP. It is a phenomenon due to *Limited-Workspace-Awareness* and therefore quite prone to awareness issues. The participants naturally used prior coordination and verbal awareness bridging mechanisms to deal with the *Limited-Workspace-Awareness* and it worked out for them. As for Parallel Exploration, the verbalizations seemed to be not harmful to the cognitive process. In conclusion, it cannot be said that Unilateral Operation is of essential *Freedom-Positive-Effect*. Likewise, for the observed pair no noticeable *Freedom-Negative-Effect* could be observed. Discussion of research questions

The observed pair appeared to be sufficiently familiar with all dialogs it used, so that it was not a problem for the remote partner not to be able to see the actual dialog. Plausibly, the familiarity with the dialogs plus the extensive verbalization of the operating participant are critical factors during such blind periods for the other one. eDPP context

4.6.3 eDPP Phenomenon: Parallelization

In a nutshell: Parallelization

During Parallelization the pair members separate from each other and work in parallel on different aspects within their shared workspace.

Sometimes the participants **left their joint work mode and split to work in parallel on different aspects of the same task**. Such episodes were either initiated by the Driver, who asked the observer to do a small subtask while he would work on another aspect of that task, or by the Active Observer who started to create test data while the Driver was still implementing the corresponding test case.

Such episodes of parallel work on straightforward subtasks are called Parallelization phenomena.

Example: Parallelization

Driver: *"Could you quickly adapt the thing above, the regex?"*
Active Observer: *"Sure."*
Driver: *"And meanwhile I'll write the calendar here."*
Active Observer: *"Yeah, do that."*

The motivation of the Driver to initiate a Parallelization episode was to pass a part of his current task to the observer (Motive^{SAttr} = Split Task). When the Active Observer initiated the Parallelization, he performed the next required step (Motive^{SAttr} = Realize Idea) for the Driver's current work. In both cases, the Driver's as well as the Active Observer's tasks were rather Motive and impact

straightforward. Such parallelizations accelerated the pair's working process (IMPACT^{SAttr} = Accelerate Next Step).

Announcement and awareness

No matter who started the PARALLELIZATION episode, he explicitly announced it (ANNOUNCEMENT STYLE^{SAttr} = Explicit). They both mumbled while working in parallel (AWARENESS BRIDGING^{SAttr} = Mumbling, AWARENESS VOICING^{SAttr} = Mumbling). The *Follow Mode* was either already disabled (AWARENESS FEATURE^{SAttr} = Follow Mode Disabled) or stopped manually (AWARENESS FEATURE^{SAttr} = Follow Mode Manually Stopped).

Start, termination, development focus and activity

PARALLELIZATION episodes occurred when the pair was writing tests (DEVELOPMENT FOCUS^{SAttr} = Testing, SHARED-MIND ACTIVITY^{SAttr} = Write Test). They either had discussed what needed to be done before splitting up (START MODE^{SAttr} = Execute Discussed) or the ACTIVE OBSERVER started it on his own initiative (START MODE^{SAttr} = Proactivity). After finishing their subtasks (TERMINATION MODE^{SAttr} = Subtask Finished), the participants waited for each other in the shared viewport where they had left (JOIN MODE^{SAttr} = Meet in Viewport).

Process fluency

Both participants were in the parallel work mode (FLUENCY^{CAttr}:*Action Recognition*$^{CAttr\text{-}A}$ = Aware, FLUENCY^{CAttr}:*Handling of Intervention*$^{CAttr\text{-}A}$ = Parallel Handling). The parallel activities took place in the editor view (PLACE OF ACTION^{SAttr} = Editor) and regardless of whether they worked in a common viewport (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Viewport) or not (COUPLING OF PARTICIPANTS^{CAttr}:*Visual Closeness*$^{CAttr\text{-}A}$ = Shared Artifact Different), both knew with which aspect the other one was concerned with (COUPLING OF PARTICIPANTS^{CAttr}:*Subject Awareness*$^{CAttr\text{-}A}$ = Bilateral). They followed an overall common train of thought (COUPLING OF PARTICIPANTS^{CAttr}:*Cognitive Affinity*$^{CAttr\text{-}A}$ = Two Brains In Unison) and performed tasks relevant to the current work step (FLUENCY^{CAttr}:*Effect on Train of Thoughts*$^{CAttr\text{-}A}$ = Integrate). The pair had a mutual general idea of their activities and did not need to explicitly coordinate (FLUENCY^{CAttr}:*Synchronization Effort*$^{CAttr\text{-}A}$ = Not Observed).

Discussion of research questions

PARALLELIZATION is an extreme manifestation of *Editing-Freedom*. Since the participants split with well-defined and hardly demanding subtasks, this mode underpins the behavior of pair programmers preferring to work alone for simpler tasks, as for example discussed in Williams et al. (2000) and Plonka et al. (2012b). In that sense, it belongs to *Freedom-Positive-Effect*.

Obviously, PARALLELIZATION involves the risk of inadvertently leaving the PP or eDPP work mode. This may result in a loss of associated benefits for the pair, then counting as *Freedom-Negative-Effect* (this was, however, not observed in the data).

PP context

PARALLELIZATION in PP requires the pair to stop the PP mode and to continue on separate machines. As with PARALLEL EXPLORATION, this would involve to commit recent changes on one machine and apply them on the other one. Then again, when the pair is done with the parallel work and wants to continue with the PP work mode, one member has to commit all changes while the other one has to update the workspace. The change between the joint and parallel work mode is possible but more cumbersome in the PP setting.

eDPP context

Since PARALLELIZATION occurred only twice, it illustrates the potential of real parallel work in eDPP. It, however, does not provide enough range to reach a deeper understanding and accordingly judge the advantages and disadvantages.

4.7 Discussion

The results mainly provide insights regarding the following eDPP aspects:

- **New Roles**: Independent workspaces open doors for the participants to new possibilities of interactions, in particular for the observer.
- **Independent Workspaces**: When participants make use of their increased flexibility by carrying out separate or tightly interleaved viewing and editing operations, they run the risk of visual and mental decoupling. This in turn thwarts the PP benefits like continuous real-time review, shared problem solving, or knowledge transfer. However, wisely used, the independence has the potential to involve the observer more actively and to make the process more efficient through tightly interleaved or parallel work. This potentially reduces the PP overhead for simple tasks and allows developers to complement their expertise more efficiently.
- **Reduced Awareness**: Due to the technically limited sharing of their workspaces, participants cannot see the same things and are not informed in detail about the other one's activities in the workspace. It is not clear where this in fact is a problem and to which degree the awareness workaround mechanisms like verbalization are an issue.
- **Process Fluency**: Phenomena relating to the independent workspaces in combination with reduced awareness are quite likely to harm the process fluency by introducing additional efforts for mental and spatial synchronization in the workspace.

The following sections recap and discuss these aspects.

4.7.1 New Roles in eDPP

The data showed that in eDPP the Active Observer's *activity level* was higher due to increased interaction possibilities.

Depending on his personality, whether he tends to enforce his will or is rather thoughtful, these new possibilities of intervening in the Driver's work potentially result in more (inappropriate) interruptions than without these new opportunities.

Interruptions in PP or eDPP are not negative per se – they are part of the synergy effect, encouraging the exchange of ideas. Interruptions, however, that detract from the task-related work or disrupt the smoothness of the collaboration are not desirable. Explicit awareness repair or scrambling for the editing-sovereignty are examples for such unwanted phenomena. They should be avoided in eDPP.

Accordingly, the appropriateness of the interruptions is an important aspect. On the one hand, this refers to the character of the interruptions, they should for example not be pedantic or of know-it-all nature. Of equal importance is the timing. Interruptions that lead to a context switch of the Driver but are of low value for the actual problem solving at that moment, are potentially harmful with respect to the fluency of the overall process. The Driver may loose track of thoughts or forget important ideas. The adverse effects of mental context switches should also not be neglected.

On the other hand, if the Active Observer does not express himself immediately, this may result in the loss of his valuable thoughts.

However, the challenge of appropriate interruptions at the right moment is not an eDPP-specific issue, although perceiving a good moment for interruptions is easier in PP due to the contextual cues in the face-to-face setting. In PP as well as in eDPP, the observer has to find the right balance between facilitating a fluent process and contributing own ideas, whereas in PP he has more awareness information available to assess the situation.

To remember his ideas, the observer may note them down in both settings. In PP, the driver might be irritated by this. In eDPP, the driver is less likely to notice the observer doing this, but, as already mentioned, it was never actually observed that the observer took a pen and paper to note his ideas.

However, the ACTIVE OBSERVER often used his freedom to promptly realize his inspirations, without having to explicate each of them (in particular the trivial ones).

A comparable intervention in PP would be longer or more distracting because the observer would have to verbally describe it instead of having the possibility to just do it in the source code (see also Section 5.3 "Conjecture: The Magic of Source Code"). And it appears that also DPP practitioners thought about and value this phenomenon: *"In my opinion, both people pairing should have the ability to edit at any time except for very short sessions. I find it frustrating to have to spell out exact changes when I do not have control"*[101].

The ACTIVE OBSERVER not only used his editing freedom, he also employed his viewing freedom to reassure himself or to look things up for the DRIVER. This enabled him to more actively engage in the collaboration and maintain his activity level.

When the ACTIVE OBSERVER can participate more actively, this counteracts disengagement. In his blog post about the use of Floobits[102], an eDPP tool also supporting such viewing freedom, Anthony Panozzo described this aspect as follows: *"I can even look at different parts of the code while other parts are being edited, which increases engagement. Compare this to being locked into whatever view our pair is looking at when using* tmux*"*[103].

Another effective antidote to disengagement in PP are verbalizations and explanations (Plonka et al., 2012b). The observed pair automatically verbalized their thoughts alongside their actions (referred to as 'thinking aloud'). Although this can be pair-specific and not due to the distributed setting, these **increased verbalizations in combination with the increased activity level of the observer seemed to have quite a positive effect**: In the data, no disengagement of the observer was observable or reported by the pair and the process appeared to be smooth and fluent.

Finally, it is not possible to give a comprehensive, generally valid positive or negative assessment of the increased interaction possibilities of the observer because this depends on complex contextual factors as well as on the participants' personality. Nevertheless, when done moderately, the viewing and editing freedom in eDPP yields the potential for the observer to stay engaged in the collaboration and to do a better observer job by providing more knowledgeable but discreet aid for the DRIVER.

Altogether, in particular the ACTIVE OBSERVER role opens up new possibilities of collaboration. Or, as expressed by Kent Beck: *"Saros takes pair programming to a new level because of the multi-driver support (e.g. two cursors, one file). Not many other pairing solutions support this. This is an added productivity multiplier beyond the keyboard sharing / looking over shoulder style"*[104].

In conclusion, it can be said that **eDPP is not just the same as doing pair programming in a distributed fashion – it is a completely different work mode.**

4.7.2 Impact of Viewing and Editing Freedom

When the ACTIVE OBSERVER made use of his independent viewing and editing freedom and this was noticed by the DRIVER, it more or less affected the DRIVER's train of thought. The DRIVER in turn somehow handled the intervention by deciding to either continue or interrupt his work.

Interventions interrupting the DRIVER are not generally worse than those which do not have a major impact on his train of thought. In general, for the right balance of interventions as well as for their

[101] `http://www.panozzaj.com/blog/2014/01/08/floobits-for-remote-pairing/` (accessed November 11, 2017)
[102] `https://floobits.com/`
[103] `http://www.panozzaj.com/blog/2014/01/08/floobits-for-remote-pairing/` (accessed November 11, 2017)
[104] `http://www.threeriversinstitute.org/blog/?p=584` (accessed November 11, 2017)

constructive handling, **relevant aspects seem to be discipline and personality of the participants as well as the duration of the independent activities**.

On the part of the Active Observer, this means that he should not use his editing freedom to an absurd extend. It is reasonable not to permanently interrupt the Driver's work in a pedantic way. Otherwise this would easily pollute the social climate between the participants, and the Driver may get confused which in turn distracts his flow. Consequently, the Active Observer should use his freedom in an appropriate manner and have the discipline to use it only for straightforward and short activities so that he can mentally stay with the driver. He should also have the discipline to immediately return to the Driver's focus and resist digressions he may encounter due to his independent viewing capabilities. In particular when the Driver does not recognize the independent activity of the Active Observer, this discipline is essential. Otherwise the Driver would wonder about things suddenly appearing in the workspace or might get the feeling that the line has gone dead due to missing verbal feedback at the other end.

In the observations of this study, the behavior of the Active Observer was generally characterized as follows:

- Discipline: Short freedom episodes of less than a minute duration.
- Discipline: He immediately (physically and mentally) returned to the Driver.
- Discipline: He performed short, simple activities that did not require notable cognitive effort.
- Discipline / Communication: Longer activities requiring cognitive resources were previously agreed upon.
- Communication: Through active listening the Active Observer continuously sent signals of mental presence.
- Personality: His interventions were constructive and not redundant or pedantic.
- Personality: He intervened in a gentle manner, not giving the impression to take the lead.

The Driver obviously did not feel the urge to check and discuss everything – he trusted in the adequacy and quality of the Active Observer's contributions and fixes.

Being egoless and welcoming the improvements from the partner also seems to be an essential character trait for smooth eDPP.

The Driver's behavior was characterized as follows:

- Personality: He accepted the activities of the Active Observer.
- Personality: He had a positive attitude regarding the improvements of the Active Observer, not interpreting them as an attack to his leading editing role.
- Discipline: He was concentrated, focusing on on his task and did not lie in wait for every little distraction.
- Communication: He made commentary alongside his actions, and in particular verbalized actions and input not shared.

eDPP worked very well for the observed pair. The pair dismissal hypothesis by Canfora et al. (2003), that distributed pairs tend to stop collaboration and begin working as solo-programmer, cannot be confirmed for the present case.

The behavior of the observed pair can be described, and there is a plausible relation between their successful, efficient collaboration and their trustworthy relationship along with their communicative and disciplined manner. Nevertheless, a clear statement about cause and effect of specific behaviors cannot be given at this level of knowledge and only based on the observation of one pair. Further eDPP research might focus on a further examination and specification of best conditions and behavioral patterns and anti-patterns for eDPP.

4.7.3 Impact of Reduced Physical Awareness

In a shared physical environment, the participants use body language to significantly simplify their verbal communication (see page 72). In the distributed setting of eDPP, this rich nonverbal communication channel is not available. The participants totally lack body language, gesture, and facial expression. Due to the proven relevance of nonverbal communication, one would expect that such a remote communication is clumsy and inefficient, since it requires more effort to communicate less information with audio-only communication (Isaacs and Tang, 1993).

When the observed pair needed to decide on a design approach or work strategy, they extensively discussed. During these intensive dialogs, they did not operate their computers nor did they permanently look at their screens. Their interaction was mainly reduced to the audio channel. Only now and then did they look at their screen and positioned their cursor or the remote selection as a virtual index finger at the position in the code they were referring to in their discussion.

For the observed pair, which knows each other quite well and has worked together in person, this audio-only discussions worked well. Both were quite focused during their discussions and appeared to be happy with reclining in their chair, staring into nothingness (the ceiling, out of the window, etc.) and focussing only on their sense of hearing. It cannot be said if and how much more effort their verbal communication required compared to a similar face-to-face situation.

Verbal communication of the observed pair was not inefficient. It is very likely that their familiarity and good communication habits were an essential factor for their smooth discussion phases. **During their discussions, they continuously showed presence by talking and kept each other in the loop regarding their thoughts**. No notable 'dead-end' feeling arose, even when the other one was busy thinking. Plausibly, this behavior is crucial for audio-only communication.

A conclusion drawn from these smooth audio-only discussions is that staring into the void allows to fully focus on the auditive information without being distracted by other visual stimuli. In-between, the screen may serve as a focused and expressive visual conversation context (see also Section 5.3 "Conjecture: The Magic of Source Code"). **This recognition suggests that a distributed setting has its own characteristics and advantages and does not necessarily have to copy the face-to-face setting as much as possible** (see also Section 5.5 "Rethinking CSCW Research"). The idea that computer-mediated interaction must be an imitation of the 'natural' setting has also been doubted by Hollan and Stornetta (1992) who claim that an imitation is never as good as the original and will always be compared to the original. Moreover, copying something hinders new considerations that may develop the potential of new ways of collaboration.

To sum it up: **proximity and visual contact can lose some of their relevance** at least for pairs such as the observed one who are well familiar with each other, who master a considerate style of communication, and who have a clear and expressive communication context like source code. *Reduced-Physical-Awareness* does not necessarily pose a problem in discussion phases.

4.7.3.1 Social Presence in eDPP

Another aspect that is affected by the *Reduced-Physical-Awareness* in eDPP is the social presence of the pair — the degree to which the remote partner is perceived as 'a real person being there' (Gunawardena, 1995). It is a subjective, socio-emotional aspect, shaped by non- or paraverbal cues like eye contact, mimic, gestures, proximity or tone of the voice of the other one (Bradner and Mark, 2001). Short et al. (1976) state that the more cues like this a medium transmits, the higher is the perceived social presence. A video conferencing system, for example, is at the high end of transmitting social presence whereas text messaging is at the low end.

Since usually used without video (see Section 1.4.5.1 "Efficiency of DPP" on page 75), eDPP tools are not at the high end regarding the transmission of cues for social presence. However, research showed that the less social stimuli a computer medium transmits, the more focused the participants are on

the actual task (Yoo and Alavi, 2001). Or, the other way around, media that transmit rich cues, like video conferencing systems, are well suited for "ambiguous and equivocal tasks that require resolution of different views and opinions among people"(Yoo and Alavi, 2001). In contrast, lean media are better suited for tasks that require the quick exchange of facts (Yoo and Alavi, 2001).

In eDPP the types of tasks vary. It is neither only about the quick exchange of facts, nor only about sharing opinions. Programming involves sharing information, discussing issues, making decisions, solving problems and manifesting ideas or solutions in source code. Social presence and the relevance of media richness has not been examined for collaborative programming. Thus, it is not clear to which degree the success and focus of the observed pair can be ascribed to the fact that the connection rarely transmits nonverbal cues. The relevance of the media leanness cannot be assessed in comparison to other factors like for example the expressivness of source code (see Section 5.3 "Conjecture: The Magic of Source Code"), the fact that eDPP tools show details about the task related actions (real-time transmission of editing actions), or the social binding of the partners due to a joint working history. Bradner and Mark (2001) made an experiment to find out more about the effects of application sharing, which can be considered similar to a eDPP setting, and videoconferencing on task performance: "Surprisingly, we found no significant difference in awareness of the observer's presence between the application sharing and the two-way video conditions. This is surprising because application sharing lacks visual feedback of the observer. This finding contradicts social presence theory which claims that media which provides visual feedback of others produce the greatest sense of social presence" (Bradner and Mark, 2001).

As discussed in the last section 4.7.3 "Impact of Reduced Physical Awareness", the familarity of the pair seems to offset the *Reduced-Physical-Awareness* to some degree. In the context of social presence it has been found that group cohesion appears to positively affect task involvement and efficacy of work (Yoo and Alavi, 2001). Group cohesion refers to the social binding of the group members which is shaped through joint past experiences (Yoo and Alavi, 2001). The observed pair members know each other well and they had a joint face-to-face working history. They appeared to have a good social binding but, as above, the relative importance of this aspect in comparison with other factors cannot yet be assessed.

4.7.4 Impact of Reduced Workspace Awareness

The workspace awareness information provided by an eDPP tool is only a small fraction of the rich information available in a face-to-face setting. The tool, however, only provides awareness of editing activities. Dialogs and other views within *Eclipse* are not shared.

4.7.4.1 Awareness During General Editing

When editing and viewing files in normal PP manner, the pair regularly enabled *Saros*' *Follow Mode*. This apparently worked well combined with the other awareness indicators like the Remote Scrollbar, Remote Caret, Colored Editing, and Remote Selection – **no essential detours or misunderstandings due to a lack of workspace awareness could be observed**.The pair verbalized its actions and perceptions with no apparent effort, and used Remote Selection or the Remote Caret as an extended index finger. They routinely used local ('here', 'there', ...) and object ('this', 'that', ...) deictic references in their verbal communication (see Section 1.4.4.3 "Part Three: How Workspace Awareness is Used"). No notable misunderstandings or non-understandings were observed that appeared to be induced by the *Reduced-Workspace-Awareness* or *Reduced-Physical-Awareness*.

The lack of cues from facial expression, gesture (nodding, shake of the head) and posture had no apparent negative impact on the pair's collaboration.

4.7.4.2 Limited Sharing of *Eclipse* Dialog Windows

When one participant opens an unshared view in *Eclipse* and the other one is in *Follow Mode*, an ideal eDPP tool would automatically transmit all relevant information from that view to the remote partner, too (such as a refactoring dialog). This could be done by enabling screen sharing for this dialog or by using the DISTRIBUTED COMMAND pattern (Schümmer and Lukosch, 2007), which would mean that local actions are simultaneously executed on the remote side as well.

The participants' way of dealing with limitations of their workspace awareness was interesting to observe, since *Saros* neither supports selective screen sharing nor the DISTRIBUTED COMMAND pattern. Actually, they never used screen sharing for dialog windows. Instead, a combination of the following two strategies to handle unilaterally visible dialogs has often been observed:

- **Thinking aloud**: While operating an unshared dialog, the DRIVER lively verbalized relevant dialog options and his inputs, which provided a far more complete picture of what was happening on his side than the mere verbalization of actions. He did this for simple dialogs such as the *Rename* dialog as well as for more complex ones, e.g. the *Create Class*.
- **Agreed solo**: The unilateral operation of complex dialogs was often preceded by an explicit negotiation and agreement regarding the intended work step. The DRIVER accomplished this step alone while the partner simply waited for the result, for example the new class.

An important aspect in these situations is that **the pair appeared to be sufficiently familiar with all dialogs it used**. It was never a problem for the remote partner not to see the actual dialog. The observer relaxed for a moment during an *Agreed Solo*.

One potentially problematic aspect of an *Agreed Solo* is the loss of the review effect, which means that faults may easily slip in. The data showed that this actually happens, although rarely. These mistakes, however, were quickly detected when the result of the unilateral operation became visible in the workspace.

The result of a dialog operation is rather not especially complex and the observer can quickly check when it appears in his workspace. This is why the operation of unshared dialogs seems to be less of a problem in spite of *Reduced-Workspace-Awareness*.

4.7.4.3 Awareness during Automated Testing

The observed pair wrote JUnit[105] test cases and ran them using the *Eclipse* plugin which visually indicates the pass or failure of tests and shows their failure messages. The view containing these information is not shared in *Saros*. Interestingly, for running tests, the observed pair emulated the DISTRIBUTED COMMAND patterns manually: They performed **parallel testing**, where both started the test locally and verbally aligned their screen contents to perform identical input actions.

Lukosch and Schümmer (2007) propose the DISTRIBUTED COMMAND pattern in particular to facilitate the testing activity in DPP, meaning that if one partner runs a test, this test is automatically executed on the other's side, too.

Lukosch and Schümmer (2007) consider the support of distributed testing as an important aspect of DPP tools: *"This practice shows that it is not sufficient to have a* SHARED EDITOR *when considering tool support for distributed XP. Instead, the developers need support for collaborative execution of tests. In a first approach, the system would create* DISTRIBUTED COMMAND *for test actions. This would allow the developers to execute the tests locally on their machines and inspect the results. In a next step, the system would allow coupled debugging including collaborative inspectors of application data and collaborative stepwise execution of a program.* APPLICATION SHARING *may ease the technical*

[105] `http://junit.org/` (accessed November 11, 2017)

problems at this stage. However, the application will not be collaboration aware, which makes it again difficult to, e.g., point at specific artifacts" (Lukosch and Schümmer, 2007).

However, *Saros* does not support the collaborative execution of tests or collaborative debugging. Notwithstanding this, the mere fact that the participants emulated the DISTRIBUTED COMMAND is an indicator of the good idea behind this pattern. Examining how they behaved when emulating it can provide valuable insights of how the pattern could be reasonably approached in DPP or eDPP tools.

Another conceivable strategy the pair could have used to do testing is a one-sided execution of the test cases with additional verbalization of the output, like with unilateral dialogs. However, this behavior was not be observed. A plausible reason is that the output of test runs is too complex to be verbally shared easily. Additionally, the JUnit plugin visually encodes information – colors (red, yellow, green) indicate the success or failure of tests. This can be evaluated visually more quickly than described verbally. And in case of failures or problems, the test's call stack is written out. This, all the more, is expensive to verbalize and even more difficult to understand or inspect without seeing it.

In the end, the manual DISTRIBUTED COMMAND is a workaround for *Reduced-Workspace-Awareness* and although it appears cumbersome, there were no indications that it inhibited the process beyond the few seconds of additional time required. This is a valuable finding, but nevertheless, an integrated shared visual testing context is very likely to make the task easier and faster.

4.7.5 Importance of Mental and Auditive Connection in eDPP

Even a short splitting of the mental focus immediately hurts the one brain mode, like when one participant was stuck in an aspect for a moment while the other one continued his work. During that short moment, the pair was not mentally connected and an explicit context sync was necessary when one contacted the other one by asking for something.

Although the criticality of such a mental splitting depends on aspects like severity and duration of splitting, effort for repair, and alike, this points out the **high risk of mental decoupling**, which would be a *Freedom-Negative-Effect*.

On the other hand, own thoughts and contributions of the ACTIVE OBSERVER can improve the code quality. If the ACTIVE OBSERVER had no freedom to act in the workspace, they might be discarded due to insignificance or to avoid interrupting the DRIVER. From this perspective, such short mental forks can turn out to be a *Freedom-Positive-Effect*.

It became evident that in eDPP it is important to hear the partner acting. In eDPP, observing the changes in the artifact as a result of the other one's activities is the only source for detailed awareness of what the other one is doing. More general information about the other one's activity is also transmitted via the audio channel, like typing or clicking. **A gap between the acoustic and visual feedback caused obvious confusion**. When one participant heard the other one typing but lacked the visual feedback of that activity, he immediately was confused and asked the other one what he was doing.

This immediate confusion is an indicator of the sensitivity and importance of auditive information due to *Reduced-Workspace-Awareness* and *Reduced-Physical-Awareness*.

4.7.6 Process Fluency in eDPP

In **more than 85% of the phenomena no synchronization effort was observable** (see Figure 4.20). In the remaining cases, the synchronization was relatively trouble-free, involving a short ping by one of the partners about information that could not be derived due to the *Reduced-Workspace-Awareness* and *Reduced-Physical-Awareness*.

As discussed in Section 4.4.14 "Complex Attribute: Fluency", a fluent eDPP process is considered to be free of avoidable interruptions that originate in the reduced awareness or limited sharing in eDPP. The

pair's process was not only almost free of interruptions but even smoother due to the increased freedom of the Active Observer. Compared to PP, he can more actively engage in the collaboration and contribute several quality improvements with no or minimal disruption of the Driver.

Finally, it appears that wisely used, **the increased freedom of the Active Observer is a kind of lubricant for the fluency of eDPP collaboration**.

Chapter 5

Conclusions

Chapter Structure – *What to expect in this chapter and why*

This chapter starts by contrasting theoretical issues of eDPP with what has actually been seen in the data. It then concludes potential benefits and problems of eDPP on a general level. Since the surprising findings of this research raised new promising research questions, these will be briefly presented, formulated as conjectures. The chapter closes with an outlook on possible subsequent steps.

5.1 eDPP – Potential and Hazards

A fundamental insight of this thesis is that eDPP is a different work mode than just doing PP in a distributed fashion. The new opportunities of interaction, in particular for the observer, change the way the pair can interact with shared artifacts. Balanced interaction possibilities result in a collaboration that is different to that of PP.

One observed and quite evident difference concerned the participants‘ activities, which reached from being highly interleaved to absolutely parallel. As a result the process was enriched by several small contributions whereas there was no notable evidence of negative effects of the independent interaction capabilities.

To detect advantageous behavioral patterns for eDPP, it is important to understand why there were only few negative impacts and why independent workspaces are advantageous. Therefore, theoretical risks of eDPP are outlined before discussing them based on the actual behavior of the observed pair.

The following problems of eDPP can be theoretically derived from a distributed setting:

Risk-Decoupling: Independent workspaces may lead to visual and mental decoupling of the participants. This in turn gradually ruins the benefits of PP, like continuous reviews, shared mind problem solving, knowledge transfer, etc.

Risk-MissActions: Reduced workspace awareness might impede keeping track of the partner's position and actions in the workspace.

Risk-MissFocus: Reduced physical awareness can result in problems with following the partner's reasoning and mental focus.

Risk-DeadEnd: Audio-only communication can lead to a dead-end impression and this in turn either result in explicit coordination effort or in disengagement.

Risk-DifficultComm: Audio-only communication which is lacking any body language may complicate verbal exchange, resulting in cumbersome communication.

Considering these potential hazards, the observed pair's behavior may be summarized in this way:

Risk-Decoupling: In general, the interventions of the ACTIVE OBSERVER were quite short, mostly within the shared viewport, and simple activities that did not tie considerable cognitive resources. The ACTIVE OBSERVER did not linger in his autonomous actions but immediately returned to the DRIVER. Accordingly, if at all, the pair only decoupled visually for a short moment while staying mentally coupled. Somewhat longer separate activities were previously agreed upon and the pair communicated moderately so that both could concentrate on their small subtask while roughly keeping each other in the loop. Only trivial subtasks were carried out in such separated working style, so that the loss of knowledge transfer was negligible for that moment. The results of unilateral visible operations could not be reviewed in real-time, but were reviewed downstream. In summary, the pair did not decouple and no loss of PP benefits was observed.

Risk-MissActions: Information concerning the remote partner's position is provided by *Saros* on several levels. When being in the same viewport, the REMOTE CARET provided detailed information about the other one's position and editing activities. For non-overlapping viewports within the same artifact, the REMOTE SCROLLBAR indicated which part of the artifact was currently seen by the other. On project level, the project explorer annotations as well as the filename next to the user's entry in the USER LIST showed which artifact the other one was currently working on. During editing phases, the participants mostly used the *Follow Mode*, so that they automatically had a shared viewport. For non-editing, unilateral visible activities, the pair effectively used thinking aloud. For more complex non-shared windows like test results, they emulated the DISTRIBUTED COMMAND pattern and synchronized verbally. Such an emulation of the DISTRIBUTED COMMAND pattern was certainly not optimal, but it did not evidently hinder the collaboration.

Risk-MissFocus: The good communication culture of the pair helped them to mentally stay aware of each other. They verbalized their thoughts and intentions and made utterances alongside their actions. One intended advantage of PP is that participants mutually explain their thoughts and that the pure process of verbalizing thoughts has the effect of noticing errors in reasoning. In the software development domain, this is referred to as *rubber duck programming*[106]. Seen from that angle, the verbalizations may have had positive side effects, or at least, were not harmful.

Risk-DeadEnd: Through ongoing verbalizations alongside their actions and active listening, the pair kept each other in the loop. Their mental connection was maintained throughout their collaboration. Bjørn et al. (2014) found that when collaborators work on dependent tasks (not real-time collaboration), these dependencies force them to engage *"in the extra effort of articulation work, such as the frequent interaction and communication that is required when working remotely"*. The observed pair also communicated quite intensively and articulated their actions. Thus, the behavioral pattern found by Bjørn et al. (2014) could be confirmed for the level of tight real-time collaboration, too.

Risk-DifficultComm: During phases of pure discussion, the pair fully concentrated on their auditive perception. In-between, their screens provided a focused conversation context. Source code is a very precise, unambiguous representation of information. Possibly, this relativizes the importance of body language during verbal communication.

[106] *"In their influential book 'The Pragmatic Programmer' Andy Hunt and Dave Thomas describe a technique for finding solutions to hard problems you are struggling with. They recommend to just tell the problem to somebody, not for getting an answer, but because while explaining the problem you are thinking differently about it. And, if there is nobody around you, get yourself a rubber duck you can talk to, hence the name of the tip."* `http://blog.florian-hopf.de/2013/08/the-pragmatic-programmers-rubber-duck.html` (accessed November 11, 2017), see also `https://en.wikipedia.org/wiki/Rubber_duck_debugging` (accessed November 11, 2017)

Taken as a whole, the familiarity of the participants as well as their constructive and disciplined behavior had a significant impact on the smoothness of their collaboration. The data confirms that trust, respectful behavior, strong communication skills, and discipline to stay with the partner are important personality traits for successful eDPP. **Personality fit appears to be even more important in eDPP than in PP**. Justifiably so, pedantic, narcissistic, or similar personality traits can be more easily acted out in eDPP than in PP, resulting in wrangling and conflicts. The relevance of personality fit and good soft skills for PP is underpinned by the statements of pair programmers who were asked *"Was there ever someone with whom you simply couldn't pair program? If so, please comment on why below"* (Williams, 2000). The following answers were given:

- *"Person took any comments as mistrust"*
- *"Person with large ego/always thought he was right"*
- *"Person always agrees (there needs to be some disagreement)"*
- *"His/her way or the highway"*
- *"Person with great insecurity or anxiety about their skills"*
- *"Overly introverted"*
- *"You need to be able to trust the other person's judgement"*

These aspects in accordance with the data of this study allow a summary of the following beneficial behavioral patterns (in particular for eDPP):

- Both partners should send signals of mental presence through active listening.
- Both participants should involve the other one in their own activities and reasonings by verbalizing perceptions and cognitive processes.
- When the Driver operates unilateral dialogs, he should keep the partner in the loop by explicitly verbalizing his inputs and thoughts.
- Longer or more complex unilateral operations should be previously discussed and agreed upon. In that way, the participant who operates the unilateral dialog is more or less the executer of determined micro-tasks.
- Participants have to trust each other with regard to their respective quality of work. Nevertheless, after unilateral operation, the observer must review the result instead of blindly accepting it.
- The Active Observer should use his editing and viewing freedom wisely. He should not intervene permanently and his contributions must be useful.
- The Active Observer should intervene discreetly, not trying to gain editing sovereignty.
- The Driver in turn should positively accept interventions of the Active Observer, not interpreting them as an attack to his professional ego or leading editing role.
- The Driver should be robust concerning interventions of the Active Observer – he should be focused and not feel disturbed by trivia.

Apart from that, the question why there were only minimal problems is of interest. There is another, more general aspect why eDPP can work better than expected, contrary to the popular belief of CSCW research. The crux of the matter is that **CSCW research often takes collocated collaboration on physical artifacts as a reference model when talking about workspace awareness** (see Section 1.4.4 "Workspace Awareness in Real-Time Groupware"). These models, however, do not adequately represent the PP collaboration situation.

Collaboration in PP already happens in a synthetic workspace with non-physical artifacts that are manipulated through input devices instead of being physical artifacts manipulated by human hands. As a consequence, awareness information provided by artifacts and interactions in PP workspaces are

already different to that in physical workspaces, and **the awareness-Δ of PP and eDPP is smaller than that of collocated collaboration with physical artifacts and eDPP**:

- People's bodies in the workspace, the physical appearance of workspace artifacts and the characteristic sound they give off when they are manipulated are not available as awareness sources in PP (as described in Section 2.1.1 "Adapting the Workspace Awareness Framework to eDPP").
- Editing changes in the artifacts are visible to both partners in PP as well as in eDPP.
- Other actions like operating dialogs or running tests are visible to both partners in PP, but they are not observable for the remote partner in eDPP.
- Gesturing and pointing is done via artificial substitutes in PP and it is visible to both partners, whereas the driver controls them. In eDPP, the artificial substitutes are reduced to the Remote Caret and the Remote Selection, but both partners can equally use them.
- Body language outside the workspace is completely missing in eDPP, which remains a fundamental difference between PP and eDPP.
- Source code has another expressiveness than physical visual objects or snippets that are arranged on a flat surface (for more refer to Section 5.3 "Conjecture: The Magic of Source Code").

5.2 Conjecture: ePP – the Better PP

A surprising insight is that, compared to PP, eDPP is feasible without notable disadvantages. Even more, the wise use of the extended interaction possibilities provide collaboration-optimizing opportunities. These findings inspire the following question: When eDPP is quite as beneficial as PP plus some more benefits, would local PP done via an eDPP tool (referred to as ePP) be the better PP? I conjecture that ePP is the better PP because it removes some essential PP problems as well as some essential eDPP issues: During editing, workspace awareness does not seem to be an issue in eDPP. Body language and physical awareness would be available in ePP. For unshared views and dialogs the participants can quickly glance over to their partner's screen while otherwise having a comfortable workspace and view on the monitor. The latter aspect has been stated to be a critical requirement for successful PP by the respondents of the PP survey of Williams (2000): *"The physical layout of our workspace allows us to both see the screen and to share the keyboard"*: 58% strongly agree, 38% agree, 2% disagree, 2% strongly disagree (Williams, 2000).

PP could benefit from using an eDPP tool in the following aspects:

- **More resource-efficient PP**: An eDPP tool allows a smooth transition between individual work and PP collaboration. It could facilitate a more targeted use of tight PP collaboration while allowing to easily split up for subtasks. ePP may thus reduce the personnel overhead for PP, one of its main points of criticism.
- **Better observer job**: The observer can easily verify or look up something and fix things without interrupting the driver.
- **Natural interaction**: As discussed in Section 1.4.1.3 "Pair Interaction in Pair Programming", research shows that the role model of PP is a simplified artificial construct that does not reflect the complexity of reality (in particular Chong and Hurlbutt (2007); Salinger and Prechelt (2013); Salinger et al. (2013)). The driver and observer roles are part of the definition of PP and the classic PP setting physically enforces these roles. Apart from the faulty assumptions regarding the type of contributions associated with roles (high level thinking vs. writing code), research shows that role-enforced interaction limitations do not reflect the desired interaction possibilities of developers: When developers had the ability to choose, *"the bulk of the pairs had returned to dual keyboard setups"* (Chong and Hurlbutt, 2007). ePP may even improve this situation because, compared to a dual keyboard setting, the input devices can be simultaneously used. With that,

ePP facilitates a more natural interaction of programmers, also contributing to more efficient PP: *"Our observations revealed pair programmers engaging in a natural pattern of interaction that, aside from designating primary responsibility for keyboard input, lacked an explicit division of labor. Instead, the pairs appeared to be most effective when both programmers took on driver and navigator responsibilities. This suggests that the driver/navigator characterization may not only be inaccurate, but that training pair programmers to work in these roles may actually inhibit more natural and more effective ways of working"* (Chong and Hurlbutt, 2007).

- **More frequent role switches**: When using eDPP tools, role switching does not require physical effort. According to Plonka et al. (2011), *"two thirds of the cases one developer dominates the driving"*. Chong and Hurlbutt (2007) observed that pairs which were equipped with a dual-keyboard setting *"developed a pervasive practice of rapidly switching control of machine input during programming sessions"*. In comparison to that, for a pair using a single-keyboard setting, *"very little switching"* could be observed (Chong and Hurlbutt, 2007). In that sense, ePP can contribute to have more frequent role switches which entail other crucial aspects like the following.

- **Balanced decision making**: In PP, keyboard control leads to a domination of the driver with regard to decision making (Chong and Hurlbutt, 2007). The imbalance of input device control results in an imbalance of the substantial process drive – the driver has *"the final authority in decision making"* (Chong and Hurlbutt, 2007). The observer *"could give suggestions, but fundamentally, the developer at the keyboard decided which suggestion to follow"* (Chong and Hurlbutt, 2007). ePP could level these imbalances by two factors: First, by facilitating more easy and thus more frequent role switches. Second, the increased interaction possibilities of the Active Observer and his contributions do not involve a role switch.

- **Foster engagement**: Programmers in the observer role stated that the pure availability of a second keyboard makes them feel more engaged in the collaboration (Chong and Hurlbutt, 2007). This can be described as continuous driving readiness, as the second keyboard makes them steadily feel faced with a potential role switch (Chong and Hurlbutt, 2007). To be well prepared for such a switch, their attention is increased with regard to their partner's actions: *"Although Evan is not actively typing at the keyboard in either of the situations he describes, he clearly feels more engaged in the task when he has a keyboard available to him. When the prospect of switching roles is more remote, he maintains a much lower level of awareness regarding his partner's activities"* (Chong and Hurlbutt, 2007). Accordingly, *"equipping pair programmers with dual keyboards to facilitate the rapid switching of keyboard control can be a simple way to foster engagement"* (Chong and Hurlbutt, 2007) – an eDPP tool equips both participants with their own keyboard and caret.

Actually, **Saros is used for ePP**. The built-in *Saros* survey involves the question 'Which goals did you want to accomplish in your latest *Saros* session?' The user's answer was: *"To use my own environment and hardware (keyboard and mouse) in pair programming."*

This is why it is worth to challenge that face-to-face PP is the best solution for joint work in a synthetic workspace. Ellis et al. (1991) raise the interesting perspective that another possible *"view of this challenge is that a remote interaction, supported by appropriate technology, presents an alternative medium. While this will not replace face-to-face communication, it may actually be preferable in some situations for some groups because certain difficulties, inconveniences, and breakdowns can be eliminated or minimized"*. This approach does not aim at trying to imitate or replace personal communication, but to *"apply appropriate technological combinations to the classes of interactions that will benefit the most from the new medium"* (Ellis et al., 1991).

5.3 Conjecture: The Magic of Source Code

Source code is a direct, unambiguous textual representation of information. The **magic of source code as a collaboration base** is one of the assumed reasons for the success of eDPP.

This assumption is mainly based on the following characteristics of source code:

Accurate The meaning of source code is accurate in the sense of unambiguous. It always exactly means what it represents. There is no scope for interpretation to derive meaning from visual representation. (Distributed) pair programmers express information or the results of their mental work with characters. Since both are familiar with the semantics of the programming language, both can unambiguously interpret the seen code and thus have the same understanding of it.

Precise Source code is not only accurate, it is also absolutely precise. It fully specifies what it means. Nothing remains to be added (without changing its meaning). Consequently, (distributed) pair programmers do not need to add meaning that in turn could result in a different understanding of the seen code.

Referable Visually represented information like *"drawings, especially when expressing ideas, [...] often do not make sense by themselves"* (Tang, 1991). When they are referenced in a conversation, the visual entity, a part of it, or its position in the workspace must be described. In contrast to many other visual representations of artifacts, source code elements (classes, methods, variables) have an unambiguous meaning and can be referenced by unique names. In PP and eDPP, the pair members can use these unique names to refer to a code segment.

The high degree of **expressiveness** of programming languages may further enhance the aforementioned aspects, though not significantly.

Common ground considerably facilitates conversation and collaboration because the conversation partners align their messages and actions according to their common ground (Gutwin and Greenberg, 2002). Source code is a considerable part of the pair's common ground. With the mentioned characteristics of source code, this part of the common ground is accurate and precise and supports efficient collaboration.

Additionally, the interaction possibilities in the shared code allow the pair to simplify their communication, following the *"principle of least effort"* (Clark, 1996). People *"try to minimize their effort in doing what they intend to do"* (Clark, 1996) when collaborating. In social communication, the principle of least effort is realized by communicative actions that shorten or replace verbal utterances. The mutually visible interaction possibilities of the participants in the shared code allow them to use deictic references and thus to simplify their communication in that way. Means that support deictic references in eDPP are for example a Remote Caret or a Remote Selection (as discussed on page 72).

Beyond that, collaborative code editing also allows **conversational props** and **visual evidence** (Gutwin and Greenberg, 2002). Conversational props are objects that are used alongside verbal communication. These help to focus the conversation or illustrate the subject of discussion, like a selected variable or a part of an artifact. Selected source code snippets are precise conversational props. **Manifesting actions** are actions on or with objects that completely replace verbal statements (Clark, 1996). Placing goods at the supermarket checkout replaces the explicit statement 'I want to pay for these goods'. Changing the name of a variable in an artifact replaces the statement 'I want to give that variable another name'. Manifesting actions can also replace explicit verbal agreement. Starting to rename a variable after the partner suggested to rename it, is a visual evidence that this suggestion has been understood and implies consent.

Altogether, consent, understanding, and thoughts can be easily expressed directly in the code as a principle of least effort. Moreover, it appears that the immaterial nature of the eDPP workspace relativizes the importance of some awareness information that were derived from face-to-face collaboration on physical objects (see Section 5.1 "eDPP – Potential and Hazards").

Plausibly, the source code characteristics **precise**, **accurate**, and **referable** are powerful components for the success of eDPP. The specificity of source code as a base for remote collaboration, however, has not yet been examined. Decoding this magic remains a relevant area for further (interdisciplinary) research.

In addition to the magic of source code as a base for efficient collaboration, tight collaboration possibly strengthens the relationships in virtual teams: Bjørn et al. (2014) doubt that the widely used approach to minimize the coupling of work among sites in GSD is always the best solution. Their observations suggest that interdependent tasks and close collaboration among sites enforces extensive interaction of distributed team members. It creates *"a situation wherein participants get to know each other (as well as their remote colleagues' working habits). This creates the opportunity to strengthen geographically distributed collaboration"* (Bjørn et al., 2014). The observations of Bjørn et al. (2014) refer to the collaboration of distributed teams on task level. eDPP is a far more intense and fine-granular collaboration of two individuals working on the same problem. Thus, it plausibly positively affects team climate, trust, binding, and knowledge transfer among sites. Investigating this assumption and thus the more far-reaching effects of the magic of source code is yet another remaining research question.

5.4 Conjecture: eDPP Helps to Uncover the Process of PP

eDPP might facilitate PP research inasmuch as it could help to uncover its process and manifestations.

The inappropriateness and immatureness of role definitions in PP has already been discussed before: The classic roles suggest that the observer thinks on a strategic level and reviews code, while the driver is writing it (Bryant et al., 2008). Qualitative PP studies yet conclude that this driver/observer distinction is misleading and inappropriate. Some research efforts already addressed this issue, but, nevertheless, understanding in particular the level of thinking of the observer is hard in PP since it can only be inferred by indicators like facial expressions, physical actions, or verbal utterances.

Here, the flexibility provided by eDPP can be helpful. It makes it easier for the observer to manifest his ideas in the shared workspace. This in turn provides direct insights about what he is concerned with.

Assuming that PP and eDPP are similar processes from the cognitive perspective, it is plausible that eDPP or ePP could help to uncover how or on what level the observer is involved in PP or would prefer to work. Of course, insights based on ePP observations only provide some possible starting points that would have to be validated with PP participants.

5.5 Rethinking CSCW Research

The findings of this work strongly indicate that a change of paradigm is overdue in CSCW research. The predominant approach that groupware must try to imitate the face-to-face setting is not only problematic, it also may obscure the vision of new, innovative solutions. Researchers should be open to new perspectives because *"in a new situation with new media depending on the nature of the collaboration object"* it is not clear *"which other constraints and options evolve"* (Hollan and Stornetta, 1992).

The explorative approach and open-minded view on eDPP processes showed that **eDPP definitely deviates from its face-to-face counterpart and involves new opportunities**.

It is time to **stop considering face-to-face situations as the perfect ideal to be imitated in a distributed setting**: *"telecommunications research seems to work under the implicit assumption that there is a natural and perfect state being there -– and that our state is in some sense broken when we are not physically proximate. The goal then is to attempt to restore us, as best as possible, to the state of being there. In our view there are a number of problems with this approach. Not only does it orient*

us towards the construction of crutch-like telecommunication tools but it also implicitly commits us to a general research direction of attempting to imitate one medium of communication with another" (Hollan and Stornetta, 1992).

A mindset like this impedes the opportunity to discover innovative ways for remote collaboration which even might also improve face-to-face situations.

Imitation does not generally lead to success. Being the original is preferable to being an imitation of something else. eDPP has proven to be a different work mode than PP with its own original momentum. This originality of eDPP should be taken into account in further research.

5.6 Outlook

5.6.1 Contributions to Research

This dissertation proves that, contrary to what is commonly believed, eDPP can work well: Negative effects are smaller than expected and, which is more important, eDPP may even have beneficial effects. This work provides an initial understanding of the process of eDPP and how to sensibly use it. The findings reveal the potential of eDPP and lead to new interesting research questions.

Altogether, this work identified the following areas for further research, in particular with regard to developing a catalog of eDPP patterns, anti-patterns, and context requirements:

- Identification and sophistication of the presented as well as of further eDPP phenomena.
- Process-wide examination of eDPP phenomena.
- Investigation of eDPP's efficiency (in comparison to PP or ePP).
- Improvement of eDPP tools.
- Investigation of the conjecture *ePP – the better PP*.
- Investigation of the conjecture *The Magic of Source Code*.

Hopefully, the presentation of this basic positive potential of eDPP might help to inspire further research projects and raise interest and acceptance of eDPP in industry.

5.6.2 Contributions to Practice

This research project contributed the following to practice:

- The usability of *Saros* was significantly improved. Users are neither put off by it nor frustrated when using it, but have a 'friendly', useful, and well-documented tool.
- The data revealed that eDPP is successfully used in industry, and this thesis identified behavioral patterns that plausibly contributed to this success.
- The observations suggest that eDPP tools potentially optimize local PP. eDPP tools are only sporadically used for ePP yet.

Appendix A

Further Aspects of Groupware and its Development

A.1 Definitions of Groupware

"The definition of what groupware is can be a topic of great debate and is often very broad" (Fouss and Chang, 2000) and *"it is not easy to find a consensus on this definition"* (Martín et al., 2003). Therefore, the goal of this section is to **convey the key facets of groupware definitions**, abandoning a single, established definition. Accordingly, the essential aspects of common definitions are extracted, structured and outlined in a synopsis.

Most definitions specify groupware by describing its purpose (Purp). On a very abstract level, they depict for what the technology is used and by whom, independent of concrete tasks, scenarios or user groups. Some definitions additionally use a technology-properties approach (Prop), describing the functional units of the technology. Others in turn complement their definition by including the socio-technical aspects (SocT), taking note of the groupware's usage context, the group and its work processes. The essence of these types of definitions is consolidated in the following:

- **Purp**: Groupware supports a group of people to achieve a common task (or goal) (Martín et al., 2003). It eases their interaction (Lukosch and Schümmer, 2007) by
  - enabling their communication (Martín et al., 2003),
  - facilitating their collaboration (Ellis et al., 1991), and
  - providing means to coordinate their activities (Ellis et al., 1991).

- **Prop**: Groupware provides
  - a shared interface (Ellis et al., 1991; Martín et al., 2003),
  - a shared environment (Ellis et al., 1991) and
  - *"may include different communication technologies, from simple plain-text chat to advanced videoconferencing"* (Martín et al., 2003).

- **SocT**: Groupware systems are *"intentional group processes plus software to support them"* (Lukosch and Schümmer, 2007). Three aspects evolve from this short definition:
  - Group: A group of individuals interacts by means of the groupware. A central concern is to satisfy their actual needs (Lukosch and Schümmer, 2007).

  - Intentional group process: To help the group achieve their objectives, their specific work flow and interaction patterns have to be supported by the groupware (Lukosch and Schümmer, 2007).
  - Software: The role of the software is to be a supporting tool, adapted to the group's needs and not the other way around (Lukosch and Schümmer, 2007). This means the group should not be forced to change their working style or to create artificial structures and processes to be able to use the tool.

- (SocT) An even more holistic definition also comprises the facets named so far, but emphasizes the socio-technical perspective and integrates the central aspect of mutual awareness in group work: Groupware *"provides a shared space for cooperation and enables awareness among group members, representing an outcome of CSCW research which encompasses sociological features of cooperative work in multiple forms and application fields"* (Cruz et al., 2012).

Communication, collaboration, and coordination, as the three aspects of the purpose (Purp) perspective will be in focus of the next Section A.2 "Groupware Taxonomies" when discussing the groupware classification schemes.

None of the definitions above specifies the timely interaction of the group members. We will return to this aspect – synchronous vs. asynchronous interaction – in the classifications section as a distinctive characteristic of groupware applications.

A.2 Groupware Taxonomies

"There are numerous characteristics, spectrums, and terms that have been proposed to help describe and classify groupware, but no single method seems to be adequate. However, by using a combination of the techniques, it is possible to get a reasonably clear picture of a particular application's capabilities and give some means for comparison" (Fouss and Chang, 2000).

The lack of a single satisfactory classification mainly originates in the complex characteristics and diverse aspects of groupware (Cruz et al., 2012). The trade-off between the complexity and adequate classification dimensions that ideally are *"mutually exclusive, exhaustive, and logically interrelated"* (Cruz et al., 2012) could not be resolved. Instead, each classification focuses on one or more different facet(s) of groupware systems. Thus, their appropriateness depends on the purpose. Even within the different taxonomies, the categories overlap. In particular collaboration suites that integrate several collaboration channels hinder a clear classification. These are, for example, shared editors with an included chat or web-meeting applications including audio and video conferencing, desktop sharing and a text chat. They cannot be classified as a whole but they should be classified according to their main purpose (Ellis et al., 1991).

As a consequence, some groupware systems could be classified into more than one category. The usual approach is to categorize the systems *"according to their primary emphasis and intent [. . .] depend[ing] upon the perspectives of the system designers"* (Ellis et al., 1991).

Figure A.1 offers an overview of the most common groupware taxonomies, grouped according to the characteristic they focus on for classification. The following taxonomies are shown and will be discussed in more detail:

- **3C-Classification**: This scheme classifies groupware according to how it supports the three key areas of collaborative work: communication, collaboration, and coordination (Ellis et al., 1991).
- **Time-space taxonomy**: Collaboration naturally takes place at a specific place at a specific time (Cruz et al., 2012). First, in computer-mediated collaboration, the members can be either face-to-face (same space) or remote (different spaces). Second, they may collaborate either synchronously

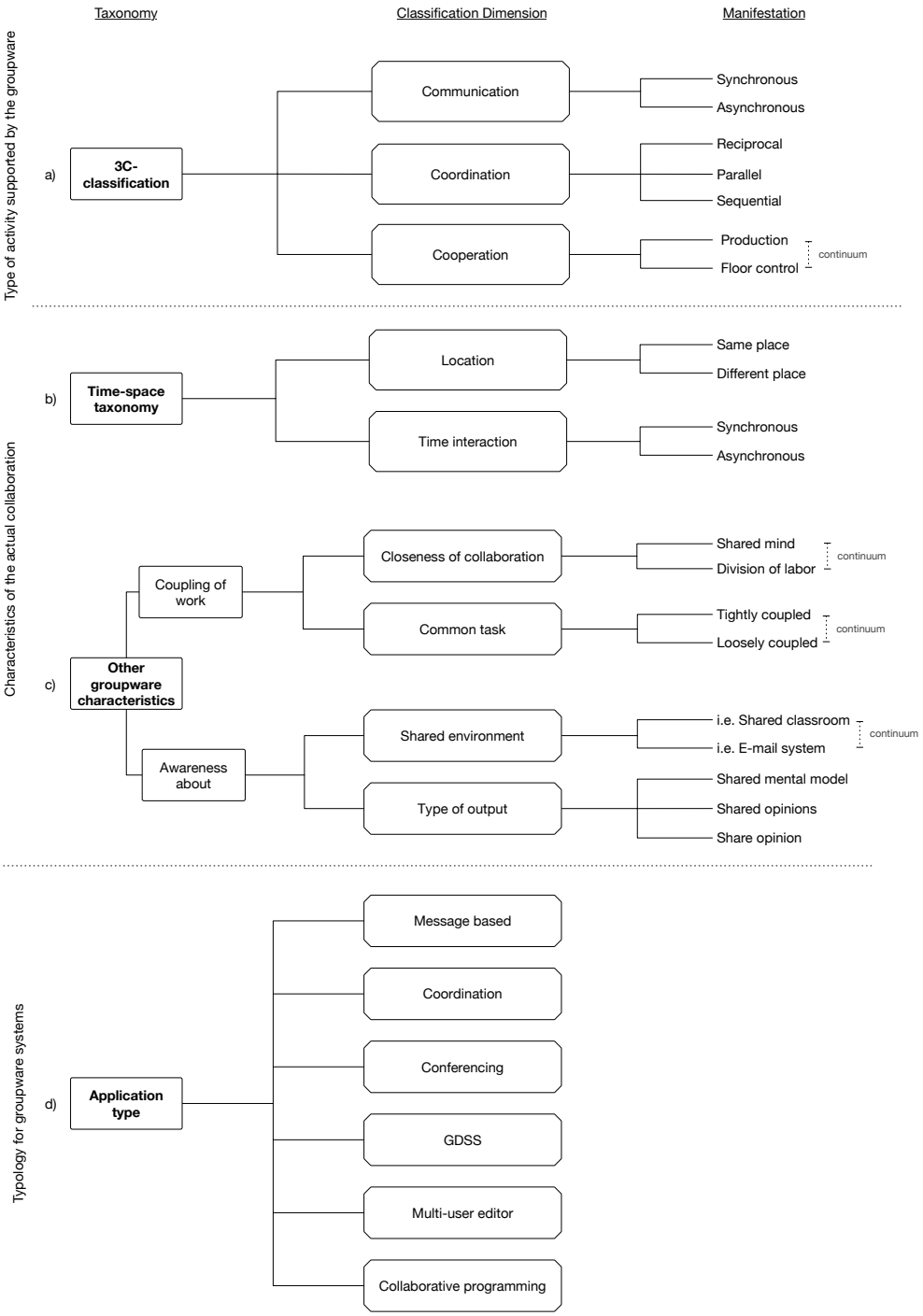

Figure A.1: Common groupware taxonomies

(same time, i.e. simultaneous activities in the shared environment) or asynchronously (time-shifted) (Ellis et al., 1991). Groupware with the purpose of facilitating synchronous interaction *"is called realtime groupware; otherwise, it is non-real-time groupware"* (Ellis et al., 1991).

- **Application type topology**: Classifies groupware systems according to the type of group activity they support.
- **Other groupware characteristics**: Further common collaboration characteristics that have not been integrated in a named model.
- **Overarching classification model**: This model is not shown in the overview of Figure A.1 because it already involves the most common, coarse taxonomies as upper levels and adds more fine-granular, socio-technical aspects as important underlying levels that have to be considered for groupware. Furthermore, awareness is considered as an orthogonal aspect, relevant on all levels in different granularities. It is shown in Figure A.4 on page 222. The purpose of this scheme is to facilitate a more social-oriented groupware development by accelerating a thorough and holistic consideration of relevant requirements for groupware design.

In some cases, the difference between the dimensions of the respective classification models is somewhat hard to grasp. This is due to different abstraction levels and naming of the respective classification dimensions. Some are more general, implicitly involving other aspects that are given as explicit classification criteria in another model. An example is the dimension 'cooperation' of the 3C-classification. It refers to a group's joint work and involves the groupware characteristics 'coupling of work' and 'common task'.

According to the quote at the beginning of this section, understanding the different perspectives of the taxonomies is important. It helps to use them adequately, understand their interlocking and make cross-taxonomy considerations. For example, the interaction between group members can be refined in communicative and cooperative (content-centric) interaction – both are dimensions of the 3C-classification. The latter focuses the collaborative creation process on shared objects in contrast to information exchange as it is meant by communicative interaction. Finally, a shared editor (an application type category) greatly facilitates cooperative creation, whereas real-time conferencing tools (another application type) are focused on communicative interaction.

A.2.1 3C-Classification

The 3C-classification model focuses on the three central aspects of collaborative work – communication, coordination, and cooperation and classifies groupware applications according to which aspect they support (Ellis et al., 1991; Schümmer and Lukosch, 2007).

- **Communication** refers to the participants' **communicative interaction** or their exchange of messages. Their communication can be **synchronous**, or **asynchronous**. And this is independent of whether the messages are anonymous or not, whether the exchanged data is written or verbal/visual and whether the communication is one-to-one or one-to-many. Examples for real-time communication groupware are audio or video conferencing tools or chat applications.
- **Cooperation** means that several people have a joint idea or shared goal they collaboratively work on. They are *"peers working together on an intellectual endeavor"* (Malone and Crowston, 1994) in *"some kind of shared space"* (Cruz et al., 2012). The overall goal is broken down into several tasks which are realized by concrete activities of the individual group members. These tasks can be quite interdependent or rather loosely coupled (see 'common task' in Section A.2.3 "Other Groupware Characteristics") and the concrete activities can be performed together as a 'shared mind' or detached from each other (see 'closeness of work' in Section A.2.3 "Other Groupware Characteristics"). For groupware classification, the cooperation dimension refers to the **degree of support for the joint creation of work results**. In that sense, a chat tool does not support cooperation because it is made for communicative interaction but not for the collaborative

production of artifacts. At the high end of this spectrum are for example collaborative editors, which allow coauthoring, the joint production of artifacts. A wiki system belongs to the upper mid-range of this spectrum (Schümmer and Lukosch, 2007), it also allows a group to share and cooperatively build a page but not with the same dynamic of interaction like a shared editor. Floor control systems with mutual exclusive access to resources in the shared environment range at the lower end.

- **Coordination** is required when several people cooperate on a shared goal which involves interdependent tasks. The **dependencies between these activities have to be managed** (Malone and Crowston, 1994) or coordinated. In the context of groupware, this refers to applications that help a group to coordinate their activities, which can be **sequential, parallel, or reciprocal**. Planning and formal workflow management tools support the sequential order of activities, depending on defined processes or required resources. Parallel coordination systems support synchronous but not interrelated activities, for example a group decision support system (GDSS) that facilitates real-time voting. Reciprocal coordination means that the groupware does not apply a general, coarse exclusion or handover mechanism but handles the actions of the individual members relative to one another. For example, a collaborative real-time editor can use operational transformation (see page 107) to bring the individual activities of the group members in the correct order before they are applied on the jointly edited resource. (Fouss and Chang, 2000; Schümmer and Lukosch, 2007)

  Typically, *"coordination is nearly invisible, and we sometimes notice coordination most clearly when it is lacking"* (Malone and Crowston, 1994).

eDPP tools are located at the high end of each classification dimension in the 3C-classification, as shown in Figure A.2:

- Communication is not supported by a pure shared editor, but usually, at least a chat for synchronous text communication is provided by eDPP tools.
- Reciprocal coordination is required on a very fine-grained level. The participants need a coherent, real-time view on the actual artifact state. Thus, the individual keystrokes and other artifact manipulations of all participants have to be put in the right order in real-time. Conflict detection is a big issue in eDPP systems.
- Since eDPP tools facilitate the joint production of software artifacts, they are to be found at the top end of cooperation support (Schümmer and Lukosch, 2007).

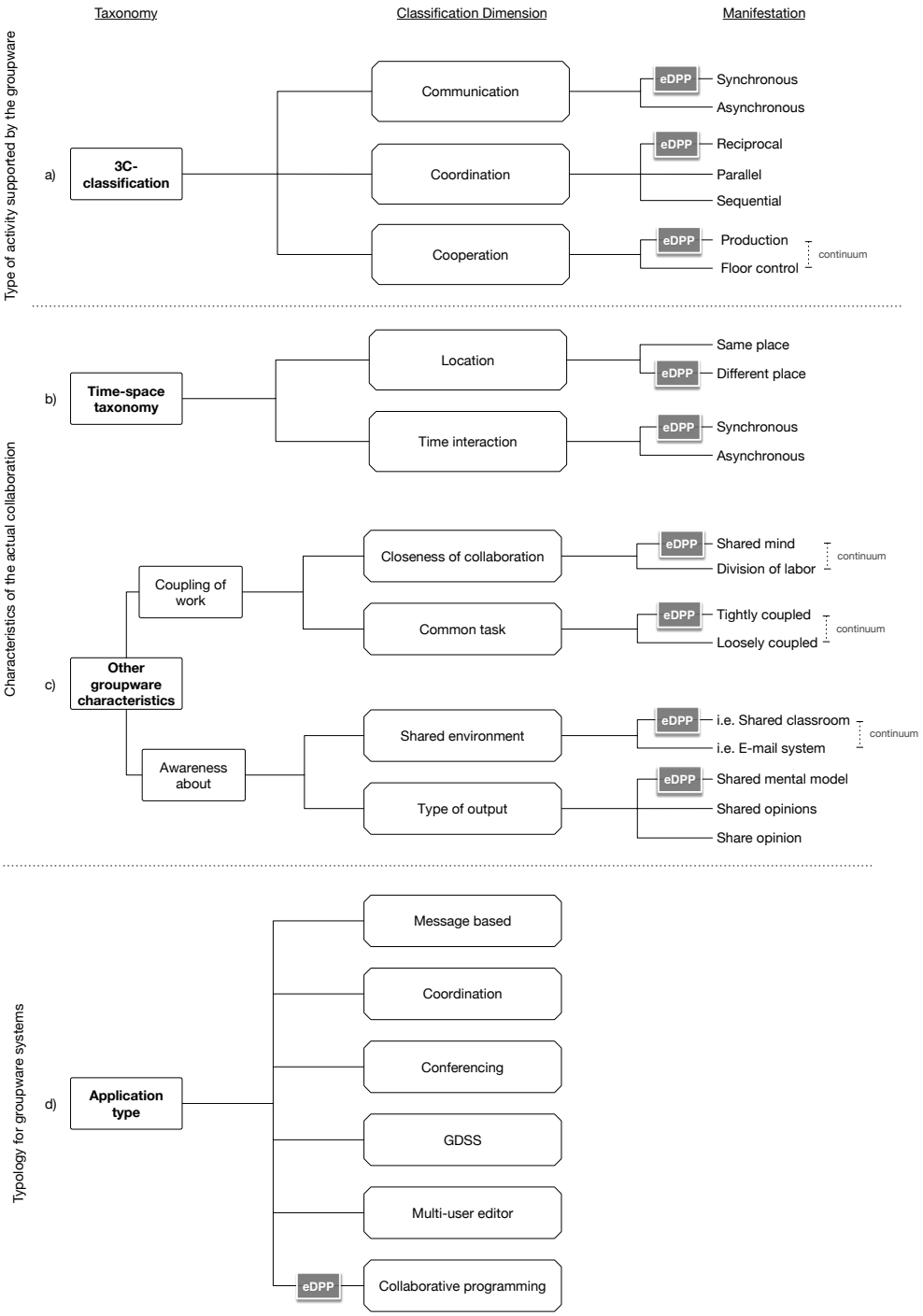

Figure A.2: Classification of eDPP tools according to common groupware taxonomies

A.2.2 Time-Space Taxonomy

Computer-mediated collaboration can bridge timely and/or spatial distance between participants. The time dimension means that the participants interact synchronously or asynchronously. It does not specify the type of interaction, it encompasses communicative as well as collaborative interaction. The spatial dimension refers to the location of the individuals *"in relation to each other"* (Fouss and Chang, 2000). The team mates can either be face-to-face at the same place or remote, at different places (Fouss and Chang, 2000). The time-space taxonomy stretches a 2×2 grid of the possible combinations of time and space distances and classifies groupware systems in the respective quadrants (Cruz et al., 2012; Ellis et al., 1991), as shown in Figure A.3. Here, examples of tools for pure communication (communicative interaction) as well as for collaboration (collaborative interaction) are added for each quadrant.

Cruz et al. (2012) refined the time-space taxonomy by addressing aspects like *"session persistence, delay between audio/video channels, reciprocity and homogeneity of channels, delay of the message sent, and spontaneity of collaboration"*. Grudin (1994) further splits the time and space dimensions in *"predictable"* and *"not predictable"*. This refers to the predictability of where or when the other group members will participate in the interaction. These refinements considerably increase the taxonomy's complexity. They are not relevant in the context of this work and, thus, not further discussed at this point.

Group members should not have to change their working environment depending on the actual timely and spatial group constitution. **Ideally, a groupware system covers the working modes of all four quadrants**. (Ellis et al., 1991)

For example, a document should be editable with the same software when working on it alone as well as when collaboratively editing it with a remote partner. This can be transferred to the domain of DPP, where it is considerably inconvenient for developers to switch their customized development environment when wanting to pair up.

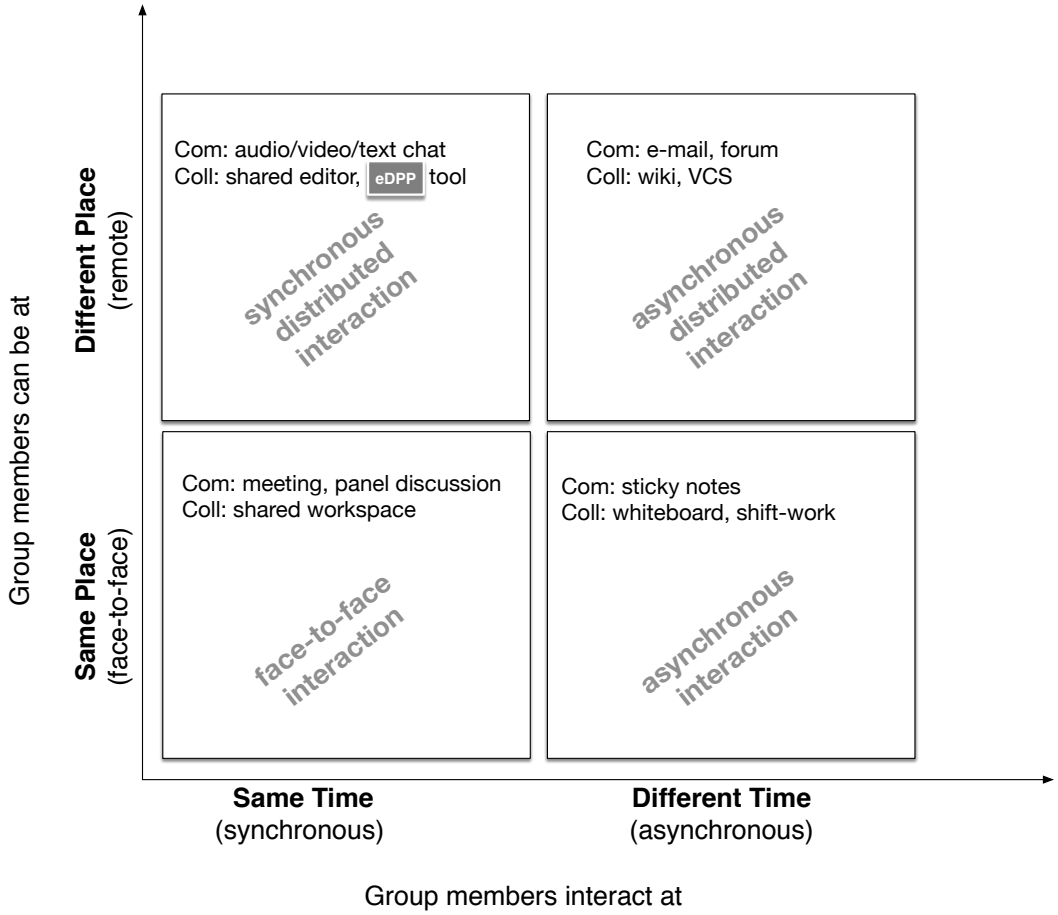

Figure A.3: Taxime-space taxonomy for groupware

The time-space taxonomy spans a 2×2 grid for the combination of the two dimensions spatial and timely distance of group members.

A.2.3 Other Groupware Characteristics

The 3C-classification clusters groupware according to which of the three main activities in group collaboration it mainly supports (communication, coordination, or collaboration). The time-space taxonomy characterizes tools according to the physical distribution of their users and the simultaneity of their collaborative activities. Besides these two models, there are other attributes used to characterize groupware without being represented as specific models. As shown in part c) of Figure A.1, they can be grouped into aspects that characterize the degree of coupling of the group members' work and those that describe the degree to which the group members are mutually aware of the others' presence and tasks.

The **coupling-of-work** facet is characterized by the dimensions **closeness of collaboration** and **common task**.

- **Closeness of collaboration** describes the **division of work among group members**. The spectrum of different possibility ranges from a **division of labor** where each is concerned with a different task up to working in a **shared mind** mode where the tasks are jointly tackled (as in eDPP). In the former case, the collaboration is in fact only the integration of the work results. (Fouss and Chang, 2000)
- **Common task** is another characteristic that spans the spectrum of **how coupled the group members are in their shared environment** when doing their work. At the low end of this spectrum, they work loosely coupled on different tasks in independent areas of the shared environment, for example on a mainframe time sharing system where they have nothing in common but the system they work on. At the high end, there are systems for tightly coupled work, where the members work synchronously on the same task in the same part of their shared environment, for example while commonly performing a code review of an artifact. (Fouss and Chang, 2000)

These two dimensions focus on different perspectives of collaborative work, but their difference is not straightforward because they somehow involve the same aspects. In fact, tightly coupled work, like doing a joint code review (common task dimension), implies working with a shared mind (closeness of collaboration dimension).

Awareness is a big issue in CSCW. Hence, there are characterizations of systems according to the level of information they provide about the shared environment or to which degree the group members have a joint understanding of something after using the groupware:

- **Shared environment** classifies the degree of availability of information about others and their environment. In that regard, an e-mail or ticket system disposes of little informative value because no information about the group members or their context is provided. Virtual classrooms or webinar systems in turn usually include rosters with availability status, text and/or video chat, and desktop sharing, and thus provide richer contextual information. (Fouss and Chang, 2000)
- **Type of output** describes to which degree a groupware system supports groups in sharing opinions or their understanding of a task or an issue. It does not refer to output in the sense of a work result or system output. Systems which allow group members to **share their opinion** range at the lowest level of this dimension. They allow expressing own opinions about something, but do not facilitate further decision making. Above that are systems that allow group members to **share opinions** by enabling them to agree on goals, priorities, and alike. Systems at the upper level allow groups to **share a mental model** by facilitating that group members have *"a clear understanding about what must be done to complete the project and the path that will be followed"*. (Fouss and Chang, 2000)

  Project planning and management tools would rather belong to the latter range.

With regard to the aforementioned dimensions, eDPP tools reside in the upper ranges: The participants share a mental model of their task and work goal. They share their workspace and are tightly-coupled,

usually working synchronously in the same artifact.

A.2.4 Application Type Topology

The application type topology classifies groupware applications based on the primary functionality they provide for a group.

- **Message-based systems** are a very old but familiar and well-established kind of groupware (Ellis et al., 1991; Fouss and Chang, 2000). Messages are exchanged between the users which can contain text as well as other types of content like images, videos, or other file types (Fouss and Chang, 2000). Message-based systems support asynchronous communication either one-to-one, one-to-many or many-to-many. Examples are e-mail, mailing lists, forums, bulletin boards, newsgroups, and social networks. Instant messaging applications like text chat applications often support the exchange of files, too, and are intended to be used for synchronous communication.
- **Coordination systems** help group members to coordinate their work. A list of tasks and others' activities helps to maintain the overview and to integrate own actions in the overall context. Resource allocation, workflow management and task-scheduling features as well as (automatic) notifications help when planning activities and avoid conflicts, redundant work, or slacks. Shared calendars or project management tools for teams are examples of such coordination systems. (Ellis et al., 1991; Fouss and Chang, 2000)
- **Conferencing systems** support a distributed group in having a conference by serving *"as a communications medium in a variety of ways"* (Ellis et al., 1991). The conference can be carried out via different channels – the users may be connected via audio, video, or a shared application. Web-conference tools usually provide a combination of these features. Tools for shared drawing or collaborative brainstorming are often referred to as conferencing aids. Their purpose is to enhance synchronous communication of distributed group members. (Ellis et al., 1991; Fouss and Chang, 2000)
- **Group decision support systems** (GDSS) assist groups in finding a consensus or taking a decision by making this process more efficient and/or improving the quality of the outcome. Depending on their range of functions, they are very similar to conferencing tools but usually more focused on structured decision making. Examples are voting tools, tools for idea generation or issue analysis. (Ellis et al., 1991; Fouss and Chang, 2000)
- **Intelligent agents**: If a virtual session requires a minimum number of participants or a specific role or group structure, these requirements can be fulfilled by intelligent agents. These are non-human participants which are responsible for specific tasks such as surrogates for enemies or teammates in online games or speaking-time controllers in an online meeting. Their 'participation' supports the group's activities. (Ellis et al., 1991)
- **Multi-user editors** allow group members to co-author documents or, more generally spoken, the joint creation of artifacts. In dedicated real-time group editors this can happen synchronously. (Ellis et al., 1991)
- **Collaborative programming tools** aid software development teams in different ways, for example by supporting the collaboration process through systems for version control, software configuration management, etc. Another kind of support for the development team may be provided by dedicated groupware development kits with a specific toolbox to facilitate the development of groupware systems or interfaces. (Ellis et al., 1991)

This typology has a clear relationship to the dimensions of the 3C-classification. Of course, it is neither straightforward, but depending on the main purpose, each application type supports at least one of the three Cs: Message and conferencing systems aim at supporting the group's communication. Coordination systems help group members to manage the dependencies in their collaborative work. Conferencing aids,

group decision support systems, intelligent agents, multi-user editors, and collaborative programming tools support various collaborative activities of the group members.

The core business of eDPP tools is the real-time multi-user editor alongside with relevant awareness information that support coordination to some degree. Usually, they also include other communication and collaboration aids, for example text chats or shared whiteboards.

A.2.5 Model to Facilitate Social-Technical Perspective

Cruz et al. (2012) wish to change that common groupware classifications hardly address the socio-technical aspect of groupware. For this purpose, they provide an overarching model *"to categorize collaboration requirements for a more social-oriented groupware development"* (Cruz et al., 2015), as shown in Figure A.4. Mainly based on literature research, they supplement the existing classification approaches with underlying, finer categories for social facets of groupware design. With that they want to facilitate a structured reflection of these *"community-centered"* (Cruz et al., 2012) aspects when designing or evaluating groupware.

The model involves the two common classification schemes, 3C-classification and the time-space grid from top to down. They are considered to be useful for an initial idea about the purpose and collaboration mode of the groupware. The layer below, the **application level**, supports a structured reflection about groupware features:

- **Regulation** refers to functionality that enables the group members to organize themselves in their shared environment, i.e. by defining roles, rules, areas of responsibility, etc. *"to ensure conformity between the activity and group goals"* (Cruz et al., 2012).
- **Groupware application properties** is the subcategory for concrete features or functional units of groupware, like the internal architecture, the user interface and its collaboration features, access control, awareness information, etc. (Cruz et al., 2012)
- **Group decision support systems (GDSS) elements** can be anything like *"hardware, software, organizationware and people support"* (Cruz et al., 2012) that has to be considered for supporting a group with taking decisions.

For collaborative work via groupware systems, **awareness** about the others and their actions is vital. A separate part of the model thus focuses just on this. Depending on the intensity and closeness of the collaboration, groupware must provide awareness indicators in an adequate granularity. Since *"awareness mechanisms are essential in collaboration systems to reduce work losses"* (Cruz et al., 2012), they must be considered on all levels of detail in groupware design and evaluation.

The **group work level** emphasizes the need to carefully understand the situational, human, and social aspects of group work and individuals:

- **Group characteristics** involve group size, *"composition, location, proximity, structure (leadership and hierarchy), formation, group awareness (low or high, and cohesiveness), behavior (cooperative or competitive), autonomy, subject, and trust"* (Cruz et al., 2012).
- **Group tasks** *"can be subdivided in creativity, planning, intellective, decision-making (choosing, evaluation and analysis, search, report, and survey) [...] having a specific complexity associated to each task"* (Cruz et al., 2012).
- **Situation/contextual factors** are *"organizational support (rewards, budget, and training), cultural contexts (trust or equity), physical setting, environment (competition, uncertainly, time pressure, and evaluative tone), and business domain at an organizational way"* (Cruz et al., 2012).
- **Individual characteristics of the group members** refer to *"personal background (work experience, training, and educational), skills, motivation, attitude towards technology, previous experience, satisfaction, knowledge, and personality"* (Cruz et al., 2012).

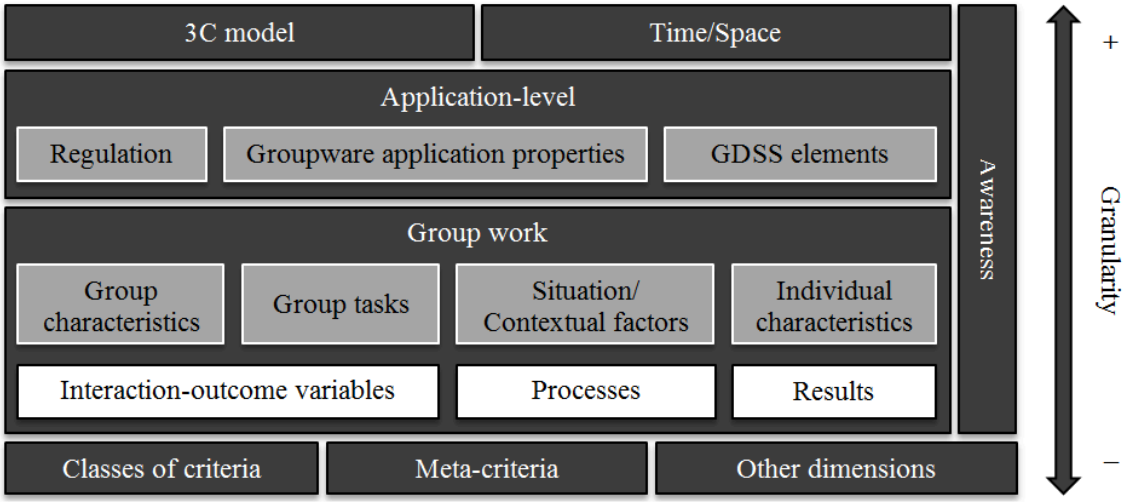

Figure A.4: Overarching groupware classification scheme

Illustration from Cruz et al. (2012)

Moreover, one should explicitly mind the so-called **interaction variables** of a group for groupware design, the group's outcomes, processes and the results of the group work:

- **Interaction-outcome variables** can be subdivided into group outcomes, individual outcomes, and system outcomes (Cruz et al., 2012). Group-related outcomes involve the *"quality of group performance, collaboration processes, and group development"* (Cruz et al., 2012). Aspects to understand about the individual group members are *"expectations and satisfaction on system use, appreciation of group membership, and individual breakdowns in system use"* (Cruz et al., 2012). For the system it is relevant to consider its *"enhancements and affordances"* (Cruz et al., 2012).
- **Processes** followed by the group must be understood *"including individual interpretation, motivation and performance dimensions"* (Cruz et al., 2012).
- **Results** of the group work are an important factor to understand group dynamics and the members' motivations. This involves *"individual rewards, group vitality, and organizational results"* (Cruz et al., 2012).

The bottom area of the model defines some categories for groupware criteria: **classes of criteria**, **meta-criteria**, and **other dimensions**. Groupware criteria involve dimensions for functional, technical, usability, or ergonomic aspects. Meta-criteria are quality criteria for groupware like scalability and orthogonality. Other dimensions refer to complementary dimensions without a specific domain like *"work coupling, shared tasks and goals, information richness and type, control centralization, activities, division of labor, patterns, techniques, scripts, assistance, learning monitoring, interaction degree, assertion, events, strategy, social connectivity, content management, process integration, sharing (view/opinion, knowledge/information, and work/operation), protection, distributed processes loss, or depth of mediation"* (Cruz et al., 2012). And although this model is quite comprehensive in comparison to the others presented here, there still remain gaps, for example concerning accessibility. (Cruz et al., 2012)

Altogether, this model is, as intended, quite comprehensive. It may be considered as a structured collection of topics to reflect upon when designing or discussing groupware systems with a strong emphasis on human/social aspects that are ubiquitous and ever-changing in group interaction.

The numerous facets of groupware as well as the heterogeneity of the different classification attempts could be interpreted as indicators of their respective *"lack of adequacy and/or scope"* (Cruz et al., 2012). There is actually no generally valid classification.

Classification schemes of groupware are models, and thus abstractions, focusing on specific aspects which are considered to be relevant. Each taxonomy is a map for the territory of groupware facets. As already stated by Alfred Korzybski, *"the map is not the territory"*, but it illustrates the territory based on some relevant characteristics. The relevance, in turn, is determined by the needs of the map's user, as for example cyclists, hikers or drivers need different maps of the same territory. Thus, groupware classification schemes must do the splits between completeness/complexity and scope/focus on a territory that is yet not fully explored. Nevertheless, roughly knowing the territory and the maps available broadens the horizon and helps to reflect about available and relevant aspects for a specific context.

Finally, the rich facets discussed in this section point out that groupware development is anything but straightforward. Its facets are not fully understood and structured. Methodical approaches that satisfy the dynamics inherent to group work are missing. Although requirements elicitation techniques in general are not limited to the development of single-user applications, they do not explicitly address groupware design issues. Nevertheless, the situation is serious but not hopeless. CSCW research faces this issue and has already yielded a considerable amount of findings relevant and helpful for the design of groupware. These are presented in the next Section A.3 "Groupware Design: Pitfalls and Fallacies".

A.3 Groupware Design: Pitfalls and Fallacies

To produce smashing products that users love and buy, the users, their domain, and tasks need to be carefully understood. Nevertheless, this prior condition is often not met at all or not properly even for single-user applications, although the awareness for the relevance of usability and user experience has grown over the last few years.

However, the root of the problem is not the lack of experience, as experience could be gathered over time. The basic problem is the ignorance concerning the own limited understanding of groupware design (Grudin, 1988). Accordingly, the design process does not adequately cope with groupware specifics which require a holistic analysis and design approach with multidisciplinary teams.

In that regard a common problem is the tendency to rely on a familiar mindset with established ideas and simply transfer such approaches from single-user application development to groupware design. This strategy does not take into account the essentially different context and requirements of groupware compared to single-user applications. The following areas in groupware design are particularly prone to inappropriate approaches and decisions:

- Requirements elicitation and system validation
- Usage context and group heterogeneity
- System design

They all have far-reaching, serious consequences for the design and consequently for the usefulness and acceptance of the final product. The following paragraphs will exemplify this.

A.3.1 Requirements Elicitation and Validation

Requirements elicitation in groupware development must consider the socio-technical aspects of groupware. As its title suggests, in single-user applications only one user at a time interacts with an application. One part of this human-computer interaction (HCI), the computer, is a deterministic interaction partner. In groupware applications, multiple users interact with each other via an application. This changes the interaction style from HCI to human-computer-human interaction (HCHI), incorporating multiple non-deterministic interaction partners, that is the group members. (Schümmer and Lukosch, 2007)

To deal with that increased complexity, Schümmer and Lukosch (2007) formulated the following aspects that have to be considered in particular during requirements elicitation for groupware (aspects of other authors are explicitly referenced):

- **Social aspects**: human-computer-human interaction involves social sides like trust, privacy, motivation, or micro-politics that influence reciprocal actions (Grudin, 1988).
- **Increased interaction dynamics**: the considerably bigger state space of possible (non-deterministic) combinations of reciprocal actions *"is one of the major gaps related to decomposition of collaboration processes in view of subsequent definition of system requirements and specification"* (Cruz et al., 2012).
- **Inexperience of the users**: Users are rather not as familiar with groupware-mediated collaboration as with single-user applications. This compounds the problem that it is even more difficult for users to adequately express their goals, working style, tasks, or problems.
- **Difficulty of user observation**: The evaluation of complex systems used by groups in realistic settings is hardly feasible.
  - It is not easy to observe groups in the field to get an understanding of their interactions and workflows, as it would require a lot of help from various people. The interactions between the members as well as between the individuals and the software would have to be observed. Moreover, group dynamics unfold over time, and snapshots probably do not provide a holistic image.
  - It is also quite problematic and difficult to organize to invite representative groups to usability labs to validate a developed groupware product. The same problems apply as just mentioned with group observation in the field. Additionally, it is very likely that group members do not act naturally due to the artificial situation. It is nearly impossible *"to create a group in the lab that will reflect the social, motivational, economic, and political factors that are central to group performance"* (Grudin, 1988). Moreover, the interaction of the group members and the members' interaction with the groupware influence one another and accordingly interactions and processes can change over time (Salvador et al., 1996).
  - Completeness of evaluation: An evaluation of the entire scope of functions of complex tools (Cruz et al., 2012) is generally difficult.
  - New approaches are required: As can bee seen from the above, the known evaluation approaches reach their limits for the evaluation of groupware. New approaches *"involving methodologies of social psychology and anthropology"* (Grudin, 1988) are necessary.
- **Hard to learn from experience**: Each group is specific in its constitution, context, background, etc., and group observations produce rich data. This is a big hurdle for drawing and documenting weighty and generalizable analysis results and thus to learn from them. (Grudin, 1988)

A.3.2 Usage Context and Group Heterogeneity

Part of the user research activity in single-application development is to identify the most important user groups of a (future) system. A user group stands for one type of users who share *"a distinct set of behavior patterns regarding the use of a particular product (or analogous activities if a product does not yet exist)"* (Cooper et al., 2012). This helps to concentrate on these most important user types and to focus on their goals and tasks in the product design instead of developing a mediocre jack-of-all-trades device. For groupware, users are a group of people per definition; and a group of people is very likely to be very heterogeneous (Salvador et al., 1996). All members of the group are actually users and no one can be dismissed from the product design. This heterogeneity implicates that the users have different, even contradicting goals and not necessarily an equal motivation to use the system.

Groupware has to take into account the different needs, otherwise it is very likely that it will not be accepted by all users, for example *"because it requires that some people do additional work, while those people are not the ones who perceive a direct benefit from the use of the application"* (Grudin, 1988). A prominent example of such an area of conflict are systems for automatic meeting scheduling. They are only useful for groups or teams if everyone in the group reliably uses them for time management, also people who otherwise would not maintain an electronic calendar. The benefit of this mandatory time management is that meetings can be easily scheduled in the team. This, however, is only a benefit for those persons that have to schedule formal meetings, and these are mostly team leaders and managers. However, the whole team has to work with an additional system they otherwise would not use, just for the benefit of the manager. (Grudin, 1988)

Accordingly, it is important to consider the groupware's collective benefit for a group, not just for individual group members. Or, as stated by Grudin (1988): *"Measuring a 'collective benefit' may be hard. If maintaining the application requires an hour per week from each group member, and its benefit is to save just one person an hour per week, is it worth it?"*

Personal benefit is thus the best motivation to use groupware (Cockburn and Jones, 1995). Any undifferentiated treatment of 'the group' would therefore be unsatisfactory.

A.3.3 System Design

The inexperience of groupware designers and the unadapted usage of ideas from single-user application in multi-user application development leads to severe shortcomings in groupware products. Mandviwalla and Olfman (1994) identified the following **limitations of current groupware systems**:

- **Group interaction (only) support**: Often, groupware only provides functionality for the collaborative tasks of a group. However, the group members would also benefit from functionality they could individually use during participation in the group work, like note taking, calculation, etc.
- **Single-user perspective**: Groupware might only reflect the viewport of one user, as for example the team leader.
- **Simplified view of groups**: Some systems just consider positive aspects of collaborative work, completely ignoring facts like status, power, and interest differences. Other systems try to reduce process loss, which means performance losses due to *"socialization and conformance to pressure"* (Mandviwalla and Olfman, 1994) in group processes. However, research showed that these aspects are not as unnecessary as they may appear to be. For example, socialization between team members is an important aspect for trust building and an open communication culture.
- **Temporal and locational variations**: Tools for the same task type should not have to be switched, dependent on whether the group works distributed or face-to-face or synchronously or asynchronously. For example, a group should not have to use different brainstorming tools, depending on whether they have a face-to-face or a remote meeting.
- **Piecemeal group support**: Groupware should not only support a fraction of a groups' tasks but one groupware system should integrate functionality to support as many as possible of a team's activities in an organization. As above, system switches for the respective activities are expensive and undesired.
- **Implicit prescriptive world view in design**: A world view in the context of application design is an assumption about how something is or works, as for example processes, hierarchies, information flows, etc. Here are two world views regarding how meetings of virtual teams are organized: the leader just outlines the agenda as a guidance for the topics of the meeting, or he strictly decides to sequentially work off each single topic. If a system reflects a world view that does not match reality, the tool is cumbersome or not usable for the group. In case of a single-user application, the users can more easily switch to another application that better fits their requirements. For

groupware, this is different. When multiple users work with the system, a whole group has to switch their system; an individual cannot do it alone.

Acronyms

GUI graphical user interface

CMI computer-mediated interaction

CSCW computer-supported cooperative work

DPP distributed pair programming

DSD distributed software development

eDPP extended distributed pair programming

ePP extended pair programming

GDSS group decision support system

GSD global software development

GTM Grounded Theory methodology

HCI human-computer interaction

HCHI human-computer-human interaction

IDE integrated development environment

PP pair programming

RPP remote pair programming

SbS side-by-side

VoIP voice over IP

VCS version control system

WQHD wide quad high definition

WYSIWIS what you see is what I see

XP eXtreme programming

Explanations of Terms

Bibliography

Pär J. Ågerfalk, Brian Fitzgerald, Helena Holmström Olsson, and Eoin Ó Conchúir. Benefits of global software development: The known and unknown. In *Proceedings of the Software Process, 2008 International Conference on Making Globally Distributed Software Development a Success Story*, ICSP'08, pages 1–9, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79587-1, 978-3-540-79587-2. URL http://dl.acm.org/citation.cfm?id=1789757.1789759.

Ban Al-Ani, Sabrina Marczak, Rafael Prikladnicki, and David F. Redmiles. Revisiting the factors that engender trust of global systems engineers. *2013 IEEE 8th International Conference on Global Software Engineering*, pages 31–40, 2013.

Prashant Baheti. Assessing distributed pair programming. In *Companion of the 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '02, pages 50–51, New York, NY, USA, 2002. ACM. ISBN 1-58113-626-9. doi: 10.1145/985072.985099.

Prashant Baheti, Edward F. Gehringer, and P. David Stotts. Exploring the efficacy of distributed pair programming. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*, pages 208–220, London, UK, UK, 2002a. Springer-Verlag. ISBN 3-540-44024-0. URL http://dl.acm.org/citation.cfm?id=647276.722333.

Prashant Baheti, Laurie A. Williams, Edward F. Gehringer, P. David Stotts, and Jason McColm Smith. Distributed pair programming: Empirical studies and supporting environments. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2002b.

Maryam Bandukda and Zafar Nasir. Efficacy of distributed pair programming. *2010 International Conference on Information and Emerging Technologies (ICIET)*, pages 1–6, 2010.

Sebastian Bauch. Verbesserung von Saros in unzuverlässigen Netzwerken mit hoher Latenz. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Sarah Beecham and John Noll. What motivates software engineers working in global software development? In *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement - Volume 9459*, PROFES 2015, pages 193–209, New York, NY, USA, 2015. Springer-Verlag New York, Inc. ISBN 978-3-319-26843-9. doi: 10.1007/978-3-319-26844-6_14.

Wjatscheslaw Belousow. Verbesserung der Architektur der DPP-Software Saros durch Einführung einer dokumentierten Modulsicht. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Karishma Bhatnagar. Collaboration and communication in agile global software development. In *Proceedings of the Undergraduate Conference on Information Systems*, 2014.

Bernd Bieber. Implementierung eines Mechanismus zur Übertragung von Informationen der Eclipse-

Konsole in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

Patrick Bitterling. Design und Implementierung der neuen Saros Benutzeroberfläche. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Pernille Bjørn, Morten Esbensen, Rasmus Eskild Jensen, and Stina Matthiesen. Does distance still matter? Revisiting the CSCW fundamentals on distributed collaboration. *ACM Trans. Comput.-Hum. Interact.*, 21(5):27:1–27:26, November 2014. ISSN 1073-0516. doi: 10.1145/2670534.

Tobias Bouschen. Bestandsaufnahme und Arbeit an einer Alpha-Version des Saros-Plugins für die IntelliJ-Plattform. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2017.

Kristy Elizabeth Boyer, August A. Dwight, R. Taylor Fondren, Mladen A. Vouk, and James C. Lester. A development environment for distributed synchronous collaborative programming. *SIGCSE Bulletin*, 40(3):158–162, June 2008. ISSN 0097-8418. doi: 10.1145/1597849.1384315.

Erin Bradner and Gloria Mark. Social presence with video and application sharing. In *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '01, pages 154–161, New York, NY, USA, 2001. ACM. ISBN 1-58113-294-8. doi: 10.1145/500286.500310.

Sallyann Bryant, Pablo Romero, and Benedict du Boulay. Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies*, 66(7):519–529, July 2008. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2007.03.005.

Nils Bussas. Entwicklung eines Server-Prototypen fur Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

G Canfora, A Cimitile, and C A Visaggio. Lessons learned about distributed pair programming: what are the knowledge needs to address? *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 314–319, 2003.

Erran Carmel and Ritu Agarwal. Tactical approaches for alleviating distance in global software development. *IEEE Software*, 18(2):22–29, March 2001. ISSN 0740-7459. doi: 10.1109/52.914734.

Lin Chen. Einführung eines Testprozesses. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Jan Chong and Tom Hurlbutt. The social dynamics of pair programming. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 354–363, Washington, DC, USA, 2007. IEEE Press. ISBN 0-7695-2828-7. doi: 10.1109/ICSE.2007.87.

Christian Cikryt. Evaluating the use of a web browser to unify GUI development for IDE plug-ins. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2015.

Herbert H. Clark. *Using Language*. Cambridge University Press, May 1996.

Herbert H. Clark and Susan E. Brennan. Grounding in communication. *Perspectives on socially shared cognition*, 1991.

Alistair Cockburn. *Crystal Clear*. A Human-Powered Methodology for Small Teams. Pearson Education, October 2004.

Alistair Cockburn and Laurie A. Williams. The costs and benefits of pair programming. In Giancarlo Succi and Michele Marchesi, editors, *Extreme Programming Examined*, pages 223–243. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-71040-4. URL `http://dl.acm.org/citation.cfm?id=377517.377531`.

Andy Cockburn and Steve Jones. Four principles of groupware design. *Interacting with Computers*, 7 (2):195 – 210, 1995. ISSN 0953-5438. doi: https://doi.org/10.1016/0953-5438(95)93509-4.

Alan Cooper, Robert Reimann, and David Cronin. *About Face 3*. The Essentials of Interaction Design. John Wiley & Sons, June 2012.

Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990. doi: 10.1007/BF00988593.

Juliet M. Corbin and Anselm Strauss. *Grounded Theory: Grundlagen Qualitativer Sozialforschung*. Beltz: Weinheim, 1996.

Philipp Cordes. Erstellung eines Testframeworks für das Schreiben von Unit-Tests im Projekt Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Armando Cruz, António Correia, Hugo Paredes, Benjamim Fonseca, Leonel Morgado, and Paulo Martins. Towards an overarching classification model of CSCW and groupware: a socio-technical perspective. In *CRIWG'12: Proceedings of the 18th international conference on Collaboration and Technology*, pages 41–56, Berlin, Heidelberg, September 2012. Springer Berlin Heidelberg.

Shirley Cruz, Fabio Q.B. da Silva, and Luiz Fernando Capretz. Forty years of research on personality in software engineering: A mapping study. *Computers in Human Behavior*, 46:94 – 113, 2015. ISSN 0747-5632. doi: http://dx.doi.org/10.1016/j.chb.2014.12.008.

Bernardo José da Silva Estácio and Rafael Prikladnicki. Distributed pair programming: A systematic literature review. *Information and Software Technology*, 63(Supplement C):1 – 10, 2015. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2015.02.011.

Daniela Damian and Didar Zowghi. RE Challenges in multi-site software development organisations. *Requirements Engineering*, 8(3):149–160, 2003. ISSN 0947-3602. doi: 10.1007/s00766-003-0173-1.

Daniela Damian, James Chisan, Polly Allen, and Brian Corrie. Awareness meets requirements management: awareness needs in global software development. In *Proceedings of the International Workshop on Global Software Development, International Conference on Software Engineering*, pages 7–11. Citeseer, 2003.

David Damm. Schnellerer Sitzungsstart in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2015.

Hendrik Degener. Benutzerzentrierte Entwicklung eines Nutzungskonzepts für das Saros-Whiteboard unter Berücksichtigung der Unterstützung von Grafiktabletts. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

Alan Dix. Challenges for cooperative work on the web: An analytical approach. *Comput. Supported Coop. Work*, 6(2-3):135–156, May 1997. ISSN 0925-9724. doi: 10.1023/A:1008635907287.

Riad Djemili. Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, Germany, 2006.

Roland Doepke and Markus Soworka. Pair programming interfaces and research. Technical report, RWTH Aachen, 2009.

Christian Dohnert. Unterstützung mehrerer Projekte in einer Saros-Sitzung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Lisa Dohrmann. Erhebung von Benutzerfeedback aus der Nutzung eines Werkzeugs zur verteilten Paarprogrammierung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.

Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work*, CSCW '92, pages 107–114, New York, NY, USA, 1992. ACM. ISBN 0-89791-542-9. doi: 10.1145/143457.143468.

Nell K. Duke and P. David Pearson. Effective practices for developing reading comprehension. *The Journal of Education*, 2002.

Rafael Duque and Crescencio Bravo. Analyzing work productivity and program quality in collaborative programming. In *2008 The Third International Conference on Software Engineering Advances (ICSEA)*, pages 270–276. IEEE Press, 2008.

Damla Durmaz. Verbesserung der Action-Awareness in Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

Tore Dyba, Erik Arisholm, Dag I. K. Sjoberg, Joe Erskine Hannay, and Forrest Shull. Are two heads better than one? On the effectiveness of pair programming. *IEEE Software*, 24(6):12–15, 2007.

Jeff Dyck, Carl Gutwin, Sriram Subramanian, and Christopher Fedak. High-performance telepointers. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 172–181, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031636.

Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: Some issues and experiences. *Commun. ACM*, 34(1):39–58, January 1991. ISSN 0001-0782. doi: 10.1145/99977.99987.

Umut Pir Erdogan. Verteiltes Debugging. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Maria Formisano. Eine bedarfsorientierte Statuseingabe für das Eclipse Plugin Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

Jonathan D. Fouss and Kai H. Chang. Classifying groupware. In *Proceedings of the 38th Annual on Southeast Regional Conference*, ACM-SE 38, pages 117–124, New York, NY, USA, 2000. ACM. ISBN 1-58113-250-6. doi: 10.1145/1127716.1127744.

Holger Hans Peter Freyther. Beseitigung von Stolpersteinen im Saros-Entwicklungsprozess. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

William W. Gaver. Sound support for collaboration. In *Proceedings of the Second Conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'91, pages 293–308, Norwell, MA, USA, 1991. Kluwer Academic Publishers. ISBN 0-7923-1439-5. URL `http://dl.acm.org/citation.cfm?id=1241910.1241932`.

Elihu M. Gerson. Supplementing grounded theory. *Social organization and social process*, 1991.

Timo Giesler. Simulation einer Netzwerkumgebung für verteilte Paarprogrammierung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2016.

Barney G. Glaser and Anselm L. Strauss. *The Discovery of Grounded Theory. Strategies for Qualitative Research*. Aldine, 1967.

Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. RESRES: The story behind the paper "Research in software engineering: an analysis of the literature". *Inf. Softw. Technol.*, 51(1):68–70, January 2009. ISSN 0950-5849. doi: 10.1016/j.infsof.2008.09.015.

Max Goldman, Greg Little, and Robert C. Miller. Collabode: Collaborative coding in the browser. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '11, pages 65–68, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0576-1. doi: 10.1145/1984642.1984658.

Saul Greenberg. A fisheye text editor for relaxed-WYSIWIS groupware. In *Conference Companion on Human Factors in Computing Systems*, CHI '96, pages 212–213, New York, NY, USA, 1996. ACM. ISBN 0-89791-832-0. doi: 10.1145/257089.257285.

Saul Greenberg, Carl Gutwin, and Andy Cockburn. Awareness through fisheye views in relaxed-WYSIWIS groupware. In *Proceedings of the Conference on Graphics Interface '96*, GI '96, pages 28–38, Toronto, Ont., Canada, Canada, 1996. Canadian Information Processing Society. ISBN 0-9695338-5-3. URL `http://dl.acm.org/citation.cfm?id=241020.241032`.

Jonathan Grudin. Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. In *Proceedings of the 1988 ACM Conference on Computer-supported Cooperative Work*, CSCW '88, pages 85–93, New York, NY, USA, 1988. ACM. ISBN 0-89791-282-9. doi: 10.1145/62266.62273.

Jonathan Grudin. Computer-supported cooperative work: History and focus. *Computer*, 27(5):19–26, May 1994. ISSN 0018-9162. doi: 10.1109/2.291294.

Charlotte N. Gunawardena. Social presence theory and implications for interaction and collaborative learning in computer conferences. *International Journal of Educational Telecommunications*, 1(2): 147–166, 1995. ISSN 1077-9124. URL `https://www.learntechlib.org/p/15156`.

Björn Gustavs. Weiterentwicklung des Eclipse-Plug-Ins "Saros" zur Verteilten Paarprogrammierung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2007.

Björn Gustavs. Stabilitäts- und Testbarkeitsverbesserungen der Netzwerkschicht in Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Carl Gutwin and Saul Greenberg. Workspace awareness in real-time distributed groupware. Technical report, University of Calgary, 1995. URL `http://hdl.handle.net/1880/45956`.

Carl Gutwin and Saul Greenberg. Workspace awareness for groupware. In *Conference Companion on Human Factors in Computing Systems*, CHI '96, pages 208–209, New York, NY, USA, 1996. ACM. ISBN 0-89791-832-0. doi: 10.1145/257089.257284.

Carl Gutwin and Saul Greenberg. Design for individuals, design for groups: Tradeoffs between power and workspace awareness. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, CSCW '98, pages 207–216, New York, NY, USA, 1998. ACM. ISBN 1-58113-009-0. doi: 10.1145/289444.289495.

Carl Gutwin and Saul Greenberg. A framework of awareness for small groups in shared-workspace groupware. Technical report, University of Saskatchewan, 1999.

Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, November 2002. ISSN 0925-9724. doi: 10.1023/A:1021271517844.

Carl Gutwin, Saul Greenberg, and Mark Roseman. Supporting awareness of others in groupware. In *Conference Companion on Human Factors in Computing Systems*, CHI '96, pages 205–, New York, NY, USA, 1996. ACM. ISBN 0-89791-832-0. doi: 10.1145/257089.257282.

Andreas Haferburg. Integration of version control systems in a tool for distributed pair programming. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Brian F. Hanks. Tool support for distributed pair programming. In *Knowledge Management Support for Distributed Agile Software Processes*, 2002.

Brian F. Hanks. Virtual pair programming. In *Doctoral Symposium at the International Conference on Software Engineering (ICSE 2003)*, 2003.

Brian F. Hanks. Empirical evaluation of distributed pair programming. *Int. J. Hum.-Comput. Stud.*, 66 (7):530–544, July 2008. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2007.10.003.

Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Inf. Softw. Technol.*, 51(7):1110–1122, July 2009. ISSN 0950-5849. doi: 10.1016/j.infsof.2009.02.001.

Karl Held. Partielle Projektsynchronisation und bedarfsgerechte Dateisynchronisation in Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

James D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, FOSE '07, pages 188–198, Washington, DC, USA, 2007. IEEE Press. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.11.

James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 81–90, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1050-7. URL `http://dl.acm.org/citation.cfm?id=381473.381481`.

Chih-Wei Ho, Somik Raha, Edward Gehringer, and Laurie A. Williams. Sangam: A distributed pair programming plug-in for Eclipse. In *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '04, pages 73–77, New York, NY, USA, 2004. ACM. doi: 10.1145/1066129.1066144.

Patrick Hobusch. Automatisierte Konfiguration des Build-Servers im Saros-Projekt mit Salt und Git. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2015.

Andreas Höfer. Video analysis of pair programming. In *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*, APOS '08, pages 37–41, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-021-0. doi: 10.1145/1370143.1370151.

Jim Hollan and Scott Stornetta. Beyond being there. In *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 119–125, New York, New York, USA, June 1992. ACM.

Helena Holmstrom, Eoin O. Conchuir, Par J. Agerfalk, and Brian Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *Proceedings of the IEEE International Conference on Global Software Engineering*, ICGSE '06, pages 3–11, Washington, DC, USA, 2006a. IEEE Computer Society. ISBN 0-7695-2663-2. URL `http://dl.acm.org/citation.cfm?id=1170744.1171354`.

Helena Holmstrom, Brian Fitzgerald, Pär J. Ågerfalk, and Eoin Ó Conchuir. Agile practices reduce distance in global software development. *Information Systems Management*, 23(3):7–18, June 2006b.

Hanna Hulkko and Pekka Abrahamsson. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 495–504, New York, NY, USA, 2005. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062545.

Ellen A. Isaacs and John C. Tang. What video can and can't do for collaboration: A case study. In *Proceedings of the First ACM International Conference on Multimedia*, MULTIMEDIA '93, pages 199–206, New York, NY, USA, 1993. ACM. ISBN 0-89791-596-8. doi: 10.1145/166266.166289.

Christoph Jacob. Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.

Ritu Jain and Ugrasen Suman. A systematic literature review on global software development life cycle. *SIGSOFT Softw. Eng. Notes*, 40(2):1–14, April 2015. ISSN 0163-5948. doi: 10.1145/2735399.2735408.

Meike Johannsen. Verbesserung und Pflege der Dokumentation der DPP-Software Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Michael Jurke. Iterative, prototype-driven development of a whiteboard feature. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Björn Kahlert. Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Neha Katira, Laurie A. Williams, Eric Wiebe, Carol Miller, Suzanne Balik, and Ed Gehringer. On understanding compatibility of student pair programmers. In *Proceedings of the 35th SIGCSE*

Technical Symposium on Computer Science Education, SIGCSE '04, pages 7–11, New York, NY, USA, 2004. ACM. ISBN 1-58113-798-2. doi: 10.1145/971300.971307.

Michael Kircher, Prashant Jain, Angelo Corsaro, and David Levine. Distributed eXtreme Programming. Technical report, Siemens AG and Dept. of Computer Science Washington University, 2001.

Alena Kiwitt. Unterstützung unterschiedlicher Programmiersprachen und Plugins in einem Werkzeug für kollaborative Softwareentwicklung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Julia Kotlarsky and Ilan Oshri. Social ties, knowledge sharing and successful collaboration in globally distributed system development projects. *Eur. J. Inf. Syst.*, 14(1):37–48, March 2005. ISSN 0960-085X. doi: 10.1057/palgrave.ejis.3000520.

Sascha Kretzschmann. Verbesserung und Überarbeitung der Saros Homepage unter Verwendung eines User-Centered Design Ansatzes. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

S. Krishna, Sundeep Sahay, and Geoff Walsham. Managing cross-cultural issues in global software outsourcing. *Communications of the ACM*, 47(4):62–66, April 2004. ISSN 0001-0782. doi: 10.1145/975817.975818.

Dominik Kröger. Evaluation von Distributed Pair-Programming mit XPairtise. Master's thesis, Fernuniversität Hagen, Fakultät für Mathematik und Informatik, 2008.

Joe Kutner. *Remote Pairing*. Collaborative Tools for Distributed Development. Pragmatic Bookshelf, December 2013.

Arndt Lasarzik. Refaktorisierung des Eclipse-Plugins Saros für die Portierung auf andere IDEs. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2015.

Stephan Lau. Verbesserte Präsenz durch Screensharing für ein Werkzeug zur verteilten Paarprogrammierung. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Lucas Layman, Laurie A. Williams, Daniela Damian, and Hynek Bures. Essential communication practices for extreme programming in a global software development team. *Information and Software Technology*, 48(9):781 – 794, 2006. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2006.01.004. Special Issue Section: Distributed Software Development.

Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341, July 2005.

Olaf Loga. Verbesserung der Kommunikationsmöglichkeiten in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Stephan Lukosch and Till Schümmer. Enabling distributed pair programming in Eclipse. In *10th European Conference on Computer-Supported Cooperative Work (ECSCW'07), Workshop 'The Challenges of Collaborative Work in Global Software Development'*, 2007.

Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, March 1994. ISSN 0360-0300. doi: 10.1145/174666.174668.

Munir Mandviwalla and Lorne Olfman. What do groups need? A proposed set of generic groupware requirements. *ACM Trans. Comput.-Hum. Interact.*, 1(3):245–268, September 1994. ISSN 1073-0516. doi: 10.1145/196699.196715.

Adriana Martín, Claudia Martínez, Nadina Martínez Carod, Gabriela Aranda, and Alejandra Cechich. Classifying groupware tools to improve communication in geographically distributed elicitation. *CACIC 2003 - RedUNCI*, 2003.

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008. ISBN 0132350882, 9780132350884.

Robert C. Martin. *The Clean Coder*. A Code of Conduct for Professional Programmers. Pearson Education, May 2011.

Philipp Mayring. *Einführung in die qualitative Sozialforschung*. Beltz Verlagsgruppe, 2002.

Natalja Menold. *Wissensintegration und Handeln in Gruppen*. Förderung von Planungs- und Entscheidungsprozessen im Kontext computerunterstützter Kooperation. Springer-Verlag, 2007.

Sanjay Misra, Ricardo Colomo-Palacios, Tolga Pusatli, and Pedro Soto-Acosta. A discussion on the role of people in global software development. *Tehnički vjesnik*, 20(3):525–531, June 2013.

Audris Mockus and James D. Herbsleb. Challenges of global software development. In *Proceedings of the 7th International Symposium on Software Metrics*, METRICS '01, pages 182–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1043-4. URL `http://dl.acm.org/citation.cfm?id=823456.824019`.

Sunila Modi, Pamela Abbott, and Steve Counsell. Negotiating common ground in distributed agile development: A case study perspective. In *Proceedings of the 2013 IEEE 8th International Conference on Global Software Engineering*, ICGSE '13, pages 80–89, Washington, DC, USA, 2013. IEEE Press. ISBN 978-0-7695-5057-2. doi: 10.1109/ICGSE.2013.18.

Nils Brede Moe and Darja Šmite. Understanding a lack of trust in global software teams: A multiple-case study. *Softw. Process*, 13(3):217–231, May 2008. ISSN 1077-4866. doi: 10.1002/spip.v13:3.

Alexander Narweleit. Analyse und Evaluation von Verbesserungsmöglichkeiten der User Experience in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Hiroshi Natsu, Jesus Favela, Alberto. L. Morán, Dominique. Decouchant, and Ana M. Martinez-Enriquez. Distributed pair programming on the web. In *Proceedings of the Fourth Mexican International Conference on Computer Science, 2003. ENC 2003.* IEEE Press, 2003. doi: 10.1109/ENC.2003.1232878.

Jerzy Nawrocki and Adam Wojciechowski. Experimental evaluation of pair programming. *Proceedings of the 12th European Software Control and Metrics Conference*, 08 2001.

Thanh Nguyen, Timo Wolf, and Daniela Damian. Global software development and delay: Does distance still matter? In *Proceedings of the 2008 IEEE International Conference on Global Software Engineering*, ICGSE '08, pages 45–54, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3280-6. doi: 10.1109/ICGSE.2008.39.

John T. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, March 1998. ISSN 0001-0782. doi: 10.1145/272287.272333.

Gary M. Olson and Judith S. Olson. Distance matters. *Hum.-Comput. Interact.*, 15(2):139–178, September 2000. ISSN 0737-0024. URL `http://dx.doi.org/10.1207/S15327051HCI1523_4`.

Judith S. Olson and Gary M. Olson. How to make distance work work. *Interactions*, 21(2):28–35, March 2014. ISSN 1072-5520. doi: 10.1145/2567788.

Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992.

Laura Plonka, Judith Segal, Helen Sharp, and Janet van der Linden. Collaboration in pair programming: Driving and switching. In *Agile Processes in Software Engineering and Extreme Programming: 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings*, pages 43–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-20677-1. doi: 10.1007/978-3-642-20677-1_4.

Laura Plonka, Judith Segal, Helen Sharp, and Janet van der Linden. Investigating equity of participation in pair programming. In *Proceedings of the 2012 Agile India*, AGILEINDIA '12, pages 20–29, Washington, DC, USA, 2012a. IEEE Press. ISBN 978-0-7695-4657-5. doi: 10.1109/AgileIndia.2012.16.

Laura Plonka, Helen Sharp, and Janet van der Linden. Disengagement in pair programming: Does it matter? In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 496–506, Piscataway, NJ, USA, 2012b. IEEE Press. ISBN 978-1-4673-1067-3. URL `http://dl.acm.org/citation.cfm?id=2337223.2337282`.

Florian Pütz. Analyse und Erweiterung der Voice-over-IP-Funktionalität in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu. Can distributed software development be agile? *Commun. ACM*, 49(10):41–46, October 2006. ISSN 0001-0782. doi: 10.1145/1164394.1164418.

Jef Raskin. Viewpoint: Intuitive equals familiar. *Communications of the ACM*, 37(9):17–18, September 1994. ISSN 0001-0782. doi: 10.1145/182987.584629.

D. Manoj Ray and Philip Samuel. Improving the productivity in global software development. In *Innovations in Bio-Inspired Computing and Applications*, pages 175–185, Cham, 2016. Springer International Publishing.

Michael Reeves and Jihan Zhu. Moomba – a collaborative environment for supporting distributed extreme programming in global software development. In *Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004. Proceedings*, pages 38–50, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24853-8. doi: 10.1007/978-3-540-24853-8_5.

Gail L. Rein and Clarence A. Ellis. The nick experiment reinterpreted: Implications for developers and evaluators of groupware. *Office Technology and People*, 5(1):47–75, January 1989.

Oliver Rieger. Weiterentwicklung einer Eclipse-Erweiterung für verteilte Paar-Programmierung im Hinblick auf Kollaboration und Kommunikation. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2008.

Marc Rintsch. Agile Weiterentwicklung eines Software-Werkzeuges zur verteilten, kollaborativen Programmierung in Echtzeit. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.

Marc Rintsch. Technisches Projektmanagement im OpenSource-Projekt Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Edna Rosen. Verteilte Paarprogrammierung im industriellen Umfeld. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.

Edna Rosen, Stephan Salinger, and Christopher Oezbek. Project kick-off with distributed pair programming. In *Proceedings of the 22nd Annual Workshop of Psychology of Programming Interest Group (PPIG '10)*, 2010.

Stefan Rossbach. Einführung einer kontinuierlichen Integrationsumgebung und Verbesserung des Testframeworks. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Stephan Salinger. *Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung*. PhD thesis, Institute of Computer Science, Freie Universität Berlin, 2013.

Stephan Salinger and Lutz Prechelt. What happens during pair programming? In *Proceedings of the 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG '08)*, September 2008.

Stephan Salinger and Lutz Prechelt. *Understanding Pair Programming: The Base Layer*. BoD.com, 2013.

Stephan Salinger, Franz Zieris, and Lutz Prechelt. Liberating pair programming research from the oppressive driver/observer regime. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1201–1204, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL `http://dl.acm.org/citation.cfm?id=2486788.2486962`.

Tony Salvador, Jean Scholtz, and James Larson. The denver model for groupware design. *SIGCHI Bulletin*, 28(1):52–58, January 1996.

Julia Schenk, Lutz Prechelt, and Stephan Salinger. Distributed-pair programming can work well and is not just distributed pair-programming. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 74–83, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591188.

Patrick Schlott. Analyse und Verbesserung der Architektur eines nebenläufigen und verteilten Softwaresystems. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2013.

Till Schümmer and Stephan Lukosch. *Patterns for Computer-Mediated Interaction (Wiley Software Patterns Series)*. John Wiley & Sons, 2007. ISBN 0470025611.

Till Schümmer and Stephan Lukosch. Supporting the social practices of distributed pair programming. In *Groupware: Design, Implementation, and Use: 14th International Workshop*, pages 83–98. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-92831-7_8.

Till Schümmer and Stephan Lukosch. Understanding tools and practices for distributed pair programming. *Journal of Universal Computer Science*, 15(16):3101–3125, 2009.

Till Schümmer and Jan Schümmer. Support for distributed teams in extreme programming. In *Extreme Programming Examined*, pages 355–377. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-71040-4. URL `http://dl.acm.org/citation.cfm?id=377517.377540`.

Barbara Seidl. *NLP*. Mentale Ressourcen nutzen. Haufe-Lexware, October 2015.

Panagiotis Sfetsos, Ioannis Stamelos, Lefteris Angelis, and Ignatios Deligiannis. Investigating the impact of personality types on communication and collaboration-viability in pair programming – an empirical study. In *Groupware: Design, Implementation, and Use*, pages 43–52, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

John Short, Ederyn Williams, and Bruce Christie. *The Social Psychology of Telecommunications*. John Wiley and Sons Ltd, September 1976.

Bastian Sieker. User-centered development of a JavaScript and HTML-based GUI for Saros. Master's thesis, Universität Paderborn, Paderborn, 2015.

Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1094–1101, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1067-3. URL `http://dl.acm.org/citation.cfm?id=2337223.2337366`.

Arsenij E. Solovjev. Evaluation der Mechanismen zum Darstellen der Workspace Awareness in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2011.

Arsenij E. Solovjev. Operationalizing the architecture of an agile software project. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

Tas Sóti. Einladungsprozess in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.

Maria Spiering. Verbesserung der Usability von Saros unter Verwendung eines User-Centered Design Ansatzes. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

Henning Staib. Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Eike Starkmann. Verteilte Paarprogrammierung in Open-Source-Projekten. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Sebastian Starroske. Improving the reliability of Saros using root cause analysis. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2013.

M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS revised: early experiences with multiuser interfaces. *ACM Trans. Inf. Syst.*, 5(2):147–167, April 1987. ISSN 1046-8188. doi: 10.1145/27636.28056.

Lev Stejngardt. Erleichterung des Einstiegs in den Saros-Entwicklungsprozess durch Beseitigung von Hürden. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2013.

David Stotts and Laurie A. Williams. A video-enhanced environment for distributed extreme programming. *Internal Report*, 2002:02–009, 2002.

David Stotts, Laurie A. Williams, Nachiappan Nagappan, Prashant Baheti, Dennis Jen, and Anne Jackson. Virtual teaming: Experiments and experiences with distributed pair programming. In *Extreme Programming and Agile Methods - XP/Agile Universe 2003: Third XP Agile Universe Conference, New Orleans, LA, USA, August 10-13, 2003. Proceedings*, pages 129–141, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45122-8. doi: 10.1007/978-3-540-45122-8_15.

Jörg Strübing. *Grounded Theory: Zur sozialtheoretischen und epistemologischen Fundierung des Verfahrens der empirisch begründeten Theoriebildung*. Qualitative Sozialforschung. VS Verlag für Sozialwissenschaften, 2008. ISBN 9783531158327. URL `https://books.google.de/books?id=4mkCX9MKvegC`.

David Sungaila. Verbesserung und Erweiterung der Core-Bestandteile von Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2016.

Andrej Szaffranietz. Einbindung von Endnutzer-Feedback in den Entwicklungsprozess. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2014.

Sandor Szücs. Behandlung von Netzwerk-und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

James Tam and Saul Greenberg. A framework for asynchronous change awareness in collaborative documents and workspaces. *Int. J. Hum.-Comput. Stud.*, 64(7):583–598, July 2006. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2006.02.004.

John C. Tang. Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies*, 34(2):143–160, February 1991. ISSN 0020-7373. doi: 10.1016/0020-7373(91)90039-A.

Ricardo Terra, Marco Tulio Valente, Krzysztof Czarnecki, and Roberto S. Bigonha. Recommending refactorings to reverse software architecture erosion. In *2012 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 335–340. IEEE Press, 2012.

Daniel Theus. Entwicklung einer Infrastruktur fur wechselbare Projektübertragungsformen und Weiterentwicklung bestehender in Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2015.

Gary Thomas. *Die virtuelle Katastrophe*. So führen Sie Teams über Distanz zur Spitzenleistung. assist Publishing, September 2014.

Jenifer Tidwell. *Designing Interfaces*. O'Reilly Media, Inc., December 2010.

Gitte Tjørnehøj, Mette Fransgård, and Signe Skalkam. Trust in agile teams: Overcoming the obstacles of distributed software development. In *Proceedings of the 35th Information Systems Research Seminar in Scandinavia*, 2012.

J. J. Treinen and S. L. Miller-Frost. Following the sun: Case studies in global software development. *IBM Systems Journal*, 45(4):773–783, October 2006. ISSN 0018-8670. doi: 10.1147/sj.454.0773.

Tomoyuki Urai, Takeshi Umezawa, and Noritaka Osawa. Enhancements to support functions of distributed pair programming based on action analysis. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 177–182, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3440-2. doi: 10.1145/2729094.2742616.

Moritz von Hoffen. Statistische Auswertung von Verteilten Paarprogrammierungssitzungen. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2010.

Darja Šmite, Claes Wohlin, Tony Gorschek, and Robert Feldt. Empirical evidence in global software engineering: A systematic review. *Empirical Software Engineering*, 15(1):91–118, February 2010. ISSN 1382-3256. doi: 10.1007/s10664-009-9123-y.

Alexander Waldmann. Evaluation und Überarbeitung der Usability von Saros. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

Norman Warnatsch. Saros – Verbesserung des algorithmischen Kerns: Gleichzeitiges Editieren. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2012.

Denis Washington. Entwicklung und Evaluation eines unabhängigen Sitzungsservers für das Saros-Projekt. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2016.

Nina Weber. Einstiegserleichterung fur die Weiterentwicklung und Erweiterung der JavaScript- und HTML-GUI von Saros. Bachelor's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2016.

Laurie A. Williams. *The collaborative software process*. PhD thesis, University of Utah, Utah, 2000.

Laurie A. Williams. What agile teams think of agile principles. *Commun. ACM*, 55(4):71–76, April 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133823. URL `http://doi.acm.org/10.1145/2133806.2133823`.

Laurie A. Williams and Robert R. Kessler. All I really need to know about pair programming I learned in kindergarten. *Commun. ACM*, 43(5):108–114, 2000. ISSN 0001-0782. doi: 10.1145/332833.332848.

Laurie A. Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE Softw.*, 17(4):19–25, July 2000. ISSN 0740-7459. doi: 10.1109/52.854064.

Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 2014. ISBN 1412960991.

Youngjin Yoo and Maryam Alavi. Media and group cohesion: Relative influences on social presence, task participation, and group consensus. *MIS Q.*, 25(3):371–390, September 2001. ISSN 0276-7783. doi: 10.2307/3250922. URL `http://dx.doi.org/10.2307/3250922`.

Franz Zieris and Lutz Prechelt. On knowledge transfer skill in pair programming. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 11:1–11:10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2774-9. doi: 10.1145/2652524.2652529.

Franz Zieris and Lutz Prechelt. Observations on knowledge transfer of professional software developers during pair programming. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 242–250, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4205-6. doi: 10.1145/2889160.2889249.

Sebastian Ziller. Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung. Master's thesis, Institute of Computer Science, Freie Universität Berlin, Berlin, 2009.