TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

# Tool Evaluation for Distributed Pair Programming

Ausgeführt am Institut für

Softwaretechnik und Interaktive Systeme

Information and Software Engineering Group

Quality Software Engineering

der

Technischen Universität Wien

unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr. Mag. Stefan Biffl

und

Dipl.-Ing. Dietmar Winkler

durch

Andreas Kaltenbach

Wienerflurgasse 43

1230 Wien

Wien, März 2008

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benützten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 13.03.2008 Andreas Kaltenbach

## Acknowledgements

After long years I have spent studying at the Technical University of Vienna, now that the end of my studies is approaching it is time to say thank you to a number of people that supported me throughout all those years.

First and foremost I would like to thank my wife Angelika and my parents for their continuous support of my studies throughout the years. In times when suffering a setback in my studies brought me down they were always there to set me up again and encouraged me to go on. Especially my wife often had to make compromises when I used my spare time for studying instead of spending it with her. That I am now able to finally complete my studies is to some degree their merit as well.

Further on my gratitude goes to all the people who are or used to be my superiors at Siemens IT Solutions and Services Program and System Engineering. All of them gave me the leeway to proceed with my studies besides the daily project business at work. Without this support it would not have been possible for me to get this far.

I would especially like to thank Stefan Biffl and Dietmar Winkler for supervision of my diploma thesis and for giving advice and feedback. Furthermore my thanks go to the following people for their help and input to the work on my diploma thesis: Matthias Kutzelnigg, Daniela Pehn, Michaela Schein, Christoph Höglinger, Christian Wallek and Matthias Heindl.

Special thanks to Olivia Elbeshausen, Veronika Mayer and Michael Riss for proofreading my diploma thesis.

Last but not least my thanks go to all the fellow students and friends of mine who accompanied me throughout the years of my studies and made these years the amazing though sometimes stressful time they were to me.

# I Abstract

On the one hand the possibilities that arise from the increasing globalisation and the increased availability of information and communication technologies along with the fact that resources in form of software engineers are much cheaper in large sections of Eastern Europe and Asia than they are in the Western World led to the situation, that many software engineering projects today are developed distributed by engineers spread over the world. Nevertheless these projects are still in need of heavy and close cooperation of the parties involved. On the other hand agile methods of software engineering are increasing in importance as nowadays state-of-the-art software engineering processes to address frequently changing requirements as well as fast and early delivery of software products. Extreme Programming is among the most prominent of these agile methods and comes with a large case of practices. Among those is Pair Programming which is a style of software development in which two software engineers team up and collaborate on the same artefact sitting side by side in front of a shared workplace. Such a team consists of the so called Driver who is typing at the computer or writing down a design. The other partner is called the Navigator. His duty is to continuously review the work of the Driver. The pair is in heavy need for continuous communication with each other and switches roles periodically.

From the two points stated above it is clear that there is a need for the appliance of the Pair Programming practice even in development teams in which the Pair Programming team members are not collocated at the same development site. In this case there is the need to have tools that support the cooperation and communication between the pair so that they are able to perform Pair Programming in a similar way as if they were sitting next to each other. Simulating this collocated collaboration in a non-collocated environment puts a number of requirements up against tools that are used to support the Distributed Pair Programming practice. Thus such tools need to be evaluated for their grade of fulfilment of these requirements to enable software engineers and project managers to select the tool best suited in a specific project.

There was already some previous research done in tools that support the Distributed Pair Programming practice as well as into requirements that are recommended to be fulfilled by such tools. Nevertheless there is still the lack of consolidated lists of such tools as well as requirements and of a way to provide means to compare these tools with respect to their usability for the support of the Distributed Pair Programming practice. This work intends to provide an extensive list of available tools with potential to support Distributed Pair Programming as well as a consolidated list of requirements desirable to be fulfilled by tools supporting the Distributed Pair Programming practice. The consolidate list of requirements is used to set up a generic evaluation framework for the comparable evaluation of such tools. The set up evaluation framework is used in a series of case studies to evaluate a selected number of these tools towards their usability in the appliance of the Distributed Pair Programming practice.

The results of this study show that though there is still a significant lack of availability of tool support for some key requirements needed to simulate the collocated situation in Distributed Pair Programming environments there is a number of tools available that provide ample support to effectively apply the Distributed Pair Programming practice.

## II Table of Content

# III List of Figures and Tables

## Figures

## Tables

# 1 Introduction

Software engineering as a term meaning methodological development of software was started to be used in the late 1960s as researchers, managers and practitioners in the area of software development started to try to improve software development practice. The NATO Science Committee initiated two conferences on software engineering in 1968 in Garmisch, Germany, [Naur and Randell, October 1968] and 1969 [Randell and Buxton, October 1969], which gave the research field its initial boost and marked the start of the profession of software engineering.

Since that time the area of software engineering underwent vast changes. Based on the original idea of software engineering, which is to transform software development from an ad-hoc, labour-intense activity to a managed, technology supported engineering discipline, a large number of process-models, methods and practices have been researched, defined and discussed throughout the last decades.

One of the latest streams of software engineering is the introduction of agile processes. Agile processes evolved as part of the reaction against the existing and already settled heavyweight processes which were seen as bureaucratic, slow, and most important contradicted the way software engineers actually worked. Due to increased demands on reducing the time to market, following a detailed plan was no longer the most important goal of a project. Rather the satisfaction of customers at the time of delivery of the finished product now got the highest priority in software engineering projects. It became common practice that major changes in the requirements, scope and technology occurred during the lifespan of a project which were often out of control of the development team. Traditional approaches focused on creating and settling the set of requirements early in advance and in this way eliminating the costs of changes. Today eliminating changes early means being unresponsive to the actual needs of the customer. Today's market demands innovative software fulfilling high quality standards being delivered in as little time as possible. Therefore the focus of agile methods – also called lightweight or iterative incremental methods – lies on simplicity and speed [Abrahamsson, Salo, et al., 2002]. In development work the development group concentrates only on functions needed at first hand, delivering them fast, collecting feedback and reacting to the information received. Today agile methods of software engineering are getting more and more important as nowadays state-of-the-art software engineering processes.

One of the most prominent and widely known of the agile methods is Extreme Programming (XP) [Beck, 2000] which was introduced around the turn from the twentieth to the twenty-first century. XP is actually a collection of ideas and practices taken from already existing methodologies which are not only applied in XP but also in some other agile methods to some extent. It covers the whole software engineering process beginning with requirement negotiation and specification till system test; its lifecycle therefore consists of five phases called Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death. Details about the various phases can be found in [Beck, 2000] and [Abrahamsson, Salo, et al., 2002] and are not in scope of this diploma thesis.

As stated by Beck [Beck, 2000], XP's methodology is by no means suitable everywhere. It is aimed for small and medium sized teams with a suggested team size between three and a maximum of twenty project members and puts significant requirements on the physical environment the team works in. Following Beck’s original idea communication and coordination between the project members is vital and should be enabled at all times. Even having the developers working in different rooms on the same floor is intolerable for XP.

As mentioned above XP is a collection of various practices. The core practice in the Iterations to Release Phase, which is used for the actual creation of the software, is called Pair Programming.

Following the definition given by Williams and Kessler [Williams and Kessler, 2002] the term Pair Programming (PP) describes the interaction and collaboration of two programmers sitting side by side in front of the same computer screen and collaboratively working on the same design, algorithms, code or test. One of the two programmers, called the Driver, is in control of keyboard and mouse and is busy with doing the concrete and current input for the implementation. The other programmer, called the Navigator or Observer, has a more strategic task. He keeps the goal in mind and monitors whether the current state of the implementation is still on track to reach this goal. He thinks about the design and takes caution that the written code is as simple and understandable as possible and that only the necessary code is implemented. To achieve his task the Navigator watches all the time what the Driver is doing and supports him in his work even with things as simple as finding typos and errors in the syntax or discrepancies from programming standards.

Effective Pair Programming is a highly collaborative activity involving continual conversation and gesturing [Williams and Kessler, 2002]. The two roles, Driver and Navigator, are exchanged arbitrarily between the two programmers building the Pair Programming pair throughout the development process and even each Pair Programming session. Both partners are, irrespective of their actual programming abilities, completely emancipated and actively participate in the development all the time. No line of code is written without the other partner’s presence and agreement. They discuss various possibilities of implementing the same functionality and in this way both partners introduce their knowledge and experience. By discussing the proceedings with the partner all the time even misunderstandings can be prevented. While the Driver is implementing he always describes what he is doing currently. The Navigator can ask questions and give suggestions.

Another major change in the field of software engineering arose with the increasing globalisation and the increased availability of information and communication technologies around the world supported by the growth of the Internet throughout the last two decades. Twenty years ago it was common practice to have a complete software project engineered more or less in the same location. Today this is not the case any more.

The fact that resources in form of software engineers are much cheaper in large sections of Eastern Europe and Asia than they are in the Western World along with the ease to collaborate due to the new possibilities of information and communication technologies led to the situation, that in most of the larger software engineering projects today Western Europeans or Americans work alongside with colleagues from Eastern European countries, India, China and many other nationalities. Major corporations launch global teams, expecting that technology will make their collaboration effective and a kind of a virtual collocation possible. Often this work done distributed around the globe is heavily intertwined, as was always the case with work done on the same project, and therefore needs heavy and close cooperation of the parties involved. This naturally introduces new needs on a social as well as technological level in software engineering projects. Even with all our information and communication technologies, distance and its associated attributes of culture, time zones, geography and language affect how humans interact with each other [Olson and Olson, 2000]. There are characteristics of face-to-face human interaction that can not or only partly be overcome with today’s technological possibilities. Nevertheless distributed work provides teams with greater flexibility for collaboration. In distributed software engineering projects team members may decide that distributed collaboration is preferable to collocated work when travel time and stress are accounted for [Mark, Grudin and Poltrock, 1999].

From the points stated above it is clear that there is a need for the appliance of the Pair Programming practice even in development teams where the team members are not collocated at the same development site; for example one of the two parts of a Pair Programming team may sit in Austria, Vienna, the other in India, Bangalore. This spatially distributed way of applying the Pair Programming practice is called Distributed Pair Programming (DPP). Following the definition taken by Baheti et al. [Baheti, Williams, et al., March 2002], the practice Distributed Pair Programming is defined as the work of geographically distributed programmers who apply the practice of Pair Programming in such a way that they collaborate synchronously with each other.

Previous research has indicated that Pair Programming results in better quality than individual programming in situations where the Pair Programming team is physically collocated [Williams, 2000], [Nosek, March 1998] as well as in such situations where this is not the case [Baheti, Williams, et al., 2002], [Baheti, Gehringer and Stotts, 2002], [Baheti, Williams, et al., March 2002]. The outcome of this research strengthens the case of Pair Programming in physically collocated as well as physically non-collocated project environments.

In the case of physical non-collocation of the participants of a Pair Programming team there is the need to have software tools that provide the needed means of collaboration and communication on hand, so that the pair is still able to perform the practice of Pair Programming in a similar way as they do when sitting next to each other in front of the same screen. Further previous research [Hanks, December 2005] already provides some ideas on requirements to be fulfilled by tools used for Distributed Pair Programming as well as on some tools specifically designed for this task.

## 1.1 Focus of this Work

As will be discussed in further detail in chapter 2, some scientific research on success factors of Pair Programming [Williams, 2000], [Williams and Kessler, May 2000] as well as on requirements for tools that can be used for means of Distributed Pair Programming [Hanks, December 2005] was already done throughout the last years. The focus of this diploma thesis is to consolidate the research of already available and defined requirements based on the outcomes of the corresponding studies and to try to extend the list of these requirements if requirements not yet listed in existing work are found to be appropriate to be asked for to be fulfilled by tools that can be used in the practice of Distributed Pair Programming. In this way a consolidated list of properties to be fulfilled to some degree by tools potentially usable in the practice of Distributed Pair Programming is set up against which actual tools for software development by means of Distributed Pair Programming can be checked and evaluated for their usability for this task.

The list of requirements set up in this way is then used to create an evaluation schema based on the abstract model to select software development tools of Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] which is based on the DESMET project [Kitchenham, Linkman and Law, 1997]. In this way a generic evaluation framework for tools supporting Distributed Pair Programming is set up.

In the previous work two primary approaches have been taken with respect to tools supporting Distributed Pair Programming. A number of researchers followed the approach to develop tools specifically designed for the usage in scope of the practice of Distributed Pair Programming (i.e. Schummer and Schummer with their TUKAN system [Schummer and Schummer, 2001], Moomba developed by Reeves and Zhu [Reeves and Zhu, 2004], Hypervideo [Stotts, Smith and Williams, March 2002] and Transparent Video Facetop [Stotts, Smith and Gyllstrom, 2004] developed by Stots et al., COPPER developed by Natsu et al. [Natsu, Favela, et al., 2003], or SubEthaEdit developed by Coding Monkeys [SubEthaEdit]). The second main approach in the previous work was to use off-the-shelf Screen-Sharing-Tools like NetMeeting or VNC which were partly adapted with minor extensions for the support of Distributed Pair Programming (i.e. VNC in scope of Hanks study [Hanks, December 2005], or MILOS ASE [Maurer, 2002] that uses NetMeeting for the support of Distributed Pair Programming). There are also some extensions or plug-ins to existing established off-the-shelf Integrated Development Environments (IDEs) specifically designed for the support of distributed software engineering and thus potentially applicable in Distributed Pair Programming (i.e. collab.Netbeans [collab.Netbeans] or various Eclipse plug-ins like PEP - Pair Eclipse Programming [PEP] or Sangam [Sangam]). This diploma thesis tries to provide a consolidated list of software tools that could possibly be used in or are specifically designed for software development by means of the Distributed Pair Programming practice.

Although a number of such tools have to some extent already been evaluated independently with respect to their usability in scope of Distributed Pair Programming, there is almost no existing work that compares such tools with each other, least of all by evaluating them based on the same requirements respectively a common evaluation framework. In scope of this work a subset of the tools found to be specifically designed for or usable in the practice of Distributed Pair Programming is evaluated by means of a case study against the set up generic evaluation framework for tools supporting the Distributed Pair Programming practice. In this way a comparable evaluation of the strengths and weaknesses of these tools is given. Further on any properties required from tools specifically designed for or usable in the practice of Distributed Pair Programming that are not covered by existing tools usable for this task is provided and discussed.

It is not in the scope of this study to research into the more social research areas of Distributed Pair Programming like cultural differences or the effect of differing usual working hours of the geographically distributed members of a Pair Programming team as it is not expected that any tools potentially usable in or specifically designed for the purpose of Distributed Pair Programming have any significant impact on these purely social matters.

## 1.2 Overview of the Document

Chapter 2 of this diploma thesis discusses related work with respect to the practices of Pair Programming and Distributed Pair Programming, requirements to be fulfilled by tools that are to be used for Distributed Pair Programming and gives an overview of the approaches taken to provide tool support for the Distributed Pair Programming practice.

The research approach taken in scope of this diploma thesis is described in chapter 3. It provides information which research questions are approached by this diploma thesis and gives a description of the approach taken to answer them.

All the tools found in scope of this work being specifically designed for or potentially usable in supporting the Distributed Pair Programming practice are listed and discussed in chapter 4.

A consolidated list of requirements to be fulfilled by tools specifically designed or usable for supporting the practice of Distributed Pair Programming and the meaning and importance of these requirements is described in chapter 5. Further on within this chapter the generic evaluation framework for tools supporting the Distributed Pair Programming practice is set up and described.

Chapter 6 contains the evaluation of a subset of tools specifically designed or usable for supporting the practice of Distributed Pair Programming. It describes how the selection of the subset of evaluated tools and how the evaluation was done, discusses the evaluation results for each of the evaluated tools individually and gives a comparison of the evaluated tools and their fulfilment of the requirements of properties against which they are evaluated by using the evaluation framework.

The discussion of the results and general findings in scope of this work is done in chapter 7.

In chapter 8 the conclusion of this work along with some suggestions on future work in context of the topic tools for Distributed Pair Programming is given.

Chapter 9 gives the list of used references and the appendix of this diploma thesis can be found in chapter 10.

# 2 Related Work

This chapter gives an introduction to the practice of Pair Programming and discusses existing research indicating that by applying the practice of Pair Programming in software engineering the results produced are of higher quality than when traditional individual programming is used. It goes on outlining some critical success factors for effective collaboration between a team applying the practice of Pair Programming.

Moving from collocated to non-collocated or even geographically distributed development environments existing research in the practice of Distributed Pair Programming and its efficiency will be discussed. In scope of this discussion a thorough look into well-known issues resulting from the physical non-collocation of Distributed Pair Programming teams as well as requirements for tools supporting the Distributed Pair Programming practice mentioned in existing research work will be taken.

## 2.1 Pair Programming

The practice of Pair Programming is not new. In 1995 L. Constantine reported in his book “Constantine and Peopleware” about observing so called Dynamic Duos producing code faster and with fewer bugs than ever before [Constantine, 1995]. In the same year the “Developing in Pairs” organizational pattern [Coplien, 1995] was published.

As already mentioned Pair Programming is one of many practices postulated in the Extreme Programming (XP) [Beck, 2000] software engineering methodology. XP is actually a collection of ideas and practices that have been found effective in already existing methodologies during the preceding decades and integrates them into a new process model. It covers the whole software engineering process beginning with requirement negotiation and specification up to system test and its lifecycle; it therefore consists of five phases called Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death (see Figure 1). The various phases of XP along with the practices applied throughout the phases are described in detail in [Beck, 2000] and [Abrahamsson, Salo, et al., 2002].

XP is a so called lightweight or agile or iterative incremental method of software engineering and as such is positioned as an explicit counterweight to the traditional so called heavyweight software engineering methods like the Rational Unified Process [Kruchten, 2000] or the V-Model [Versteegen, 2000]. It discards some elements of traditional software engineering methods, mainly such producing organizational ballast, in favour of faster and more efficient coding and compensates for this by putting more weight on other concepts, especially in the area of testing. Although its name might suggest otherwise, XP actually has quite a detailed description of its methodological aspects. This enables XPs supporters to postulate that by using XP and its practices it is possible to achieve a high level of quality of developed software while spending a comparably low amount of effort and reaching a high level of customer satisfaction.

As stated by Beck [Beck, 2000], XP’s methodology is by no means suitable everywhere but is rather aimed for small and medium sized teams with a suggested team size between three and a maximum of twenty project members and puts significant requirements on the physical environment the team works in.

Following Beck’s original idea communication and coordination between the project members is vital and should be enabled at all times. Scattering the programmers even into different rooms on the same floor is intolerable according to the original postulate of XP’s founders Kent Beck, Ron Jeffries and Ward Cunningham [Beck, 2000] although the geographical distribution of teams is not necessarily outside the scope of XP in case it includes “two teams working on related projects with limited interaction” [Beck, 2000].

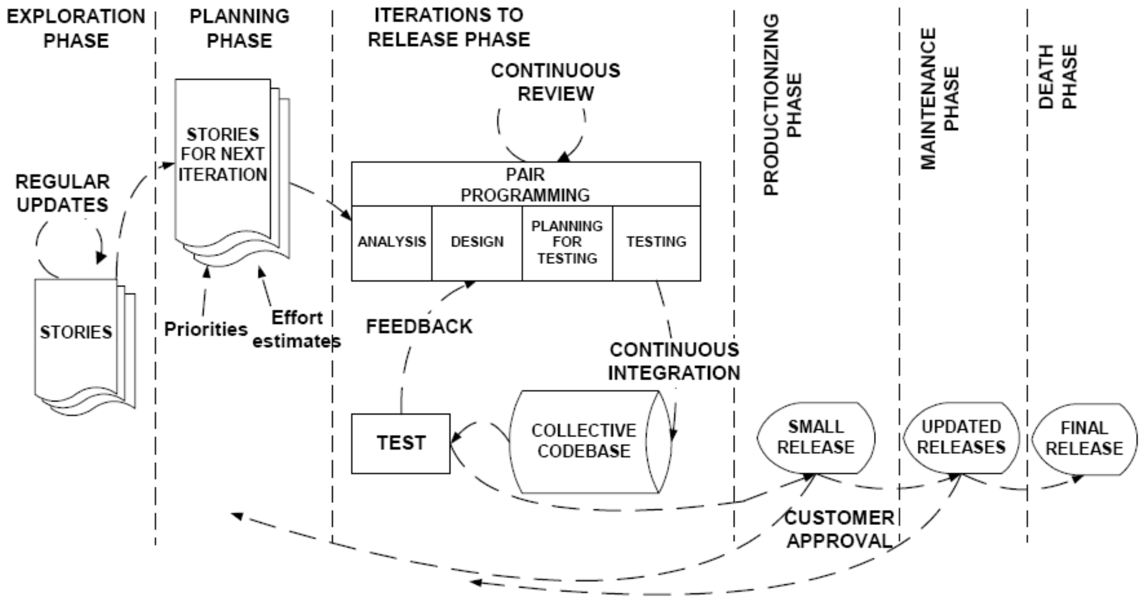


*Figure 1. Lifecycle of the XP process*
*[Abrahamsson, Salo, et al., 2002]*

According to Abrahamsson et al [Abrahamsson, Salo, et al., 2002] XP aims at enabling successful software development despite vague or constantly changing requirements in small to medium sized teams. Short iterations with small releases and rapid feedback, customer participation, communication and coordination, continuous integration and testing, collective ownership of the code, limited documentation and Pair Programming are among the main characteristics of XP.

As can be seen in Figure 1, Pair Programming along with continuous integration and feedback make up the core practices in the Iterations to Release Phase of the XP process model, which is used for the actual creation of the software product.

Pair Programming is not restricted to its usage in XP but rather was introduced into various other software engineering processes and methodologies as well like for example the Crystal Family [Cockburn, 2000], [Abrahamsson, Warsta, et al., May 2003] or house-made software development processes of various large software houses. Studies done by Williams et al as well as Haynes and Friedenberg suggest that Pair Programming is among the safest and most successful practices of agile software development [Williams, Layman, et al.], [Haynes and Friedenberg, May 2006].

### 2.1.1 An Introduction into the Practice of Pair Programming

Following the definition given by Williams and Kessler [Williams and Kessler, 2002] and as stated in [Gerken, 2003/04] the term Pair Programming describes the interaction and collaboration of two programmers sitting side by side in front of the same computer screen and collaboratively and continuously working on the same design, algorithm, code or test.

One of the two programmers, called the Driver, is in control of the input devices like keyboard and mouse and is busy with doing the input for the implementation. The other programmer, called the Navigator or Observer, has a more strategic task. While the Driver is mainly working on the concrete and actual implementation, the Navigator keeps the goal in mind and thinks about whether the current state of the implementation is still on track to reach this goal. He thinks about the design and takes care that the written code is as simple and understandable as possible and that only the necessary code to fulfil the requirements is implemented. To achieve his task the Navigator observes what the Driver is doing all the time and supports him in his work even with simple things like finding typos and errors in the syntax or discrepancies from programming standards.

Both partners are developing everything together. No line of code is written without the partner being present and agreeing to it. They discuss various possibilities of implementing the same functionality and by this introduce their knowledge and experience into their work. By discussing the proceeding with the partner all the time even misunderstandings can be prevented. While the Driver is implementing he describes what he is doing. His partner asks questions if something is unclear and gives hints and advice.

Effective Pair Programming is a highly collaborative activity involving continual conversation and gesturing [Williams and Kessler, 2002]. The two roles, Driver and Navigator, are exchanged arbitrarily between the two programmers collaborating as the Pair Programming pair throughout the development process and even each collaborative session. Both partners are completely emancipated and actively participate in the development all the time without respect to their actual programming abilities.

Williams and Kessler researched into synergy effects of Pair Programming [Williams, 2000], [Williams and Kessler, 2002] resulting from the situation of collaboratively working in a pair. Based on their results those effects will be described in the following.

An important effect to mention here is the so called pair pressure, meaning that simply due to the presence of each other the partners put some kind of positive pressure onto each other. This leads to the situation that programmers having a partner at their side will tend less to violations of programming conventions and standards or to just hack in code. The partners rely on each other and each others' performance and in this way will do their best to produce good work because they do not want to let their partner down. Each partner keeps his counterpart focused and on-task. The pressure resulting from this is not felt as negative in contradiction to such pressure resulting from other control mechanisms like for example the superior watching over one's shoulder.

Another important effect is that the Pair Programming partner acting in the role of the Navigator performs a permanent review of the design or code the Driver is writing. In this way the partners apply the four-eye-principle and will find many faults quite early, often already at the time of typing the faulty code or design section. Compared to this, traditional reviews done by a colleague, interface partner or quality assurance responsible are rather taking much time as such reviews do not happen on the fly and the reviewers are not familiar with the code or design they are reviewing. Moreover such formal reviews often feel somewhat uncomfortable. While in the Pair Programming practice the code is developed collaboratively and both partners feel responsibility and ownership for the code and any defects it contains, traditional reviews are done by external people putting the code owner in some kind of a defensive situation for his work. Further on, many sources including [Humphrey, 1997] state that it is ten times more expensive to remove a defect for each additional process step taken after the introduction of the defect. As due to the continuous review character of the practice of Pair Programming, many faults are already found while coding the faulty code section, thus radically cutting costs on fault detection and correction.

As already mentioned, due to the setting of Pair Programming both developers introduce their experience and knowledge into their work. They might have differing ways of dealing with the same problem and through discussing these divergent approaches they will find more possible solutions than each one of them alone. While a single developer will tend to implement the first solution coming to his mind, the Pair Programming partners will discuss and evaluate their alternative approaches and may find even new approaches in this way. This enables them to find better solutions faster then each one alone.

Discussing differing approaches not only leads to finding better solutions but by working collaboratively on the same work also knowledge transfer between the partners is enabled. This can range from tips about the usage of certain tools to concepts of programming languages and even to strategies of design. By actively applying knowledge learned from the partner they will learn new things faster and better than by for example traditional lectures.

Pair Programming means that there is always a partner by your side. This enables the developers to work on problems which they would have felt overwhelmed by working on their own. Having developed code with a partner gives the pair a better feeling of their code being correct as each of them always knows that another person also having a certain degree of proficiency agreed to what has been written.

Learning from each other and sharing knowledge and responsibility of the developed code has another positive effect. With Pair Programming the risk of losing key programmers is reduced because there are multiple people familiar with each part of the system.

Further on in a working Pair Programming pair the partners will motivate each other. They work on a common goal and support each other on the way. They share the work but also the feeling of success when reaching a goal giving the feeling that at least the partner appreciates the work done. This leads to a higher feeling of appreciation which supports the motivation to take on the next problem.

Last but not least working in a pair means more fun than working alone. In a questionnaire done by Williams anonymously with employees and students in the field of software engineering which formerly used to program on their own a vast majority stated that they had more fun Pair Programming than developing software on their own [Cockburn and Williams, 2001].

An important issue while actually working in a Pair Programming context is the layout of the workplace the Pair Programming Pair works in as discussed by

Gerken [Gerken 2003/04]. Besides general requirements on workplace conditions like a good information technology infrastructure, practical furniture, good light conditions, sufficient fresh air and so on some further aspects are to be taken into account when setting up a workplace for programming in pairs.

In a Pair Programming workplace it is necessary that the developing pair is able to sit side by side in an equitable manner. If this is not the case, for example if one of the pair sits behind the other, the one in the rear is actually cut off from the input devices and looks over the others shoulder, which makes for an uncomfortable feeling for the one in the front. Further on exchanging the roles is bound to changing places in such a Pair Programming environment. As this will hamper the regular role changes Pair Programmers rather fancy the “slide the keyboard, don’t move the furniture” approach [Williams and Kessler, May 2000].

To enable the developers to sit comfortably side by side the desk should not be too small and feature a sufficient depth as the angle of view on the screen is not good if the screen is too close to the developers. The form of the desk should be either square or have a slightly convex curve where the developers are sitting (see Figure 2).

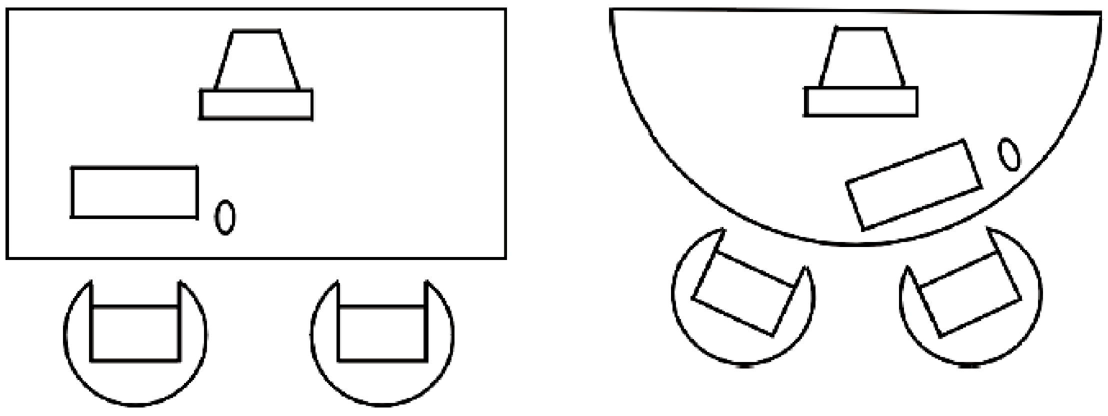


*Figure 2. Workplace Layout for Pair Programming [Gerken 2003/04]*

To enable both partners to have sufficient view of the screen it must not be too small; 17 inch is given as a minimum in [Williams and Kessler, 2002]. The screen is put in the middle of the desk between the two developers. Further on it must be possible to shift the input devices, keyboard and mouse, easily from one partner to the other to support the easiness of role changes as in general the Driver will have keyboard and mouse in front of him and shift them to his partner if a role change takes place.

### 2.1.2 Strengthening the Case of Pair Programming

“From a business standpoint, profit is not only an organization’s goal, it is necessary for its survival. The ultimate aim of engineering is to create the most income from the least expense, thus maximizing profit.” [Tockey, November/December 1997]

“The affordability of Pair Programming is a key issue. If it is more expensive, managers simply will not permit it.” [Williams, 2000]

Sceptics might suggest that applying the Pair Programming practice will double the cost of code development as two developers are needed for the same piece of software. This statement is kind of short sighted as it does not take into account that besides code development costs other expenses such as quality assurance, testing and fault correction as well as maintenance costs for software already in the field are to be considered too. Previous research by a number of researchers in the field of Pair Programming has shown that Pair Programming produces better quality than individual programming at the same or even lower expenses.

One very important metric when it comes to discussing the economic affordability of the appliance of the Pair Programming practice is quality which on one hand impacts in which phases of a software project defects are detected and on the other hand is one of the key issues impacting customer satisfaction as soon as the product is applied in the field. Williams made an experiment at the University of Utah in 2000 [Williams, 2000]. Collaborating student pairs applying the practice of Pair Programming and individual students completed four programs to the same specifications. The bar chart in Figure 3 shows the average of the percentage of the instructor's test cases passed for the two groups. The outcome was that in average the collaborating students' code had about 15 percent fewer defects than the code of the students working individually.

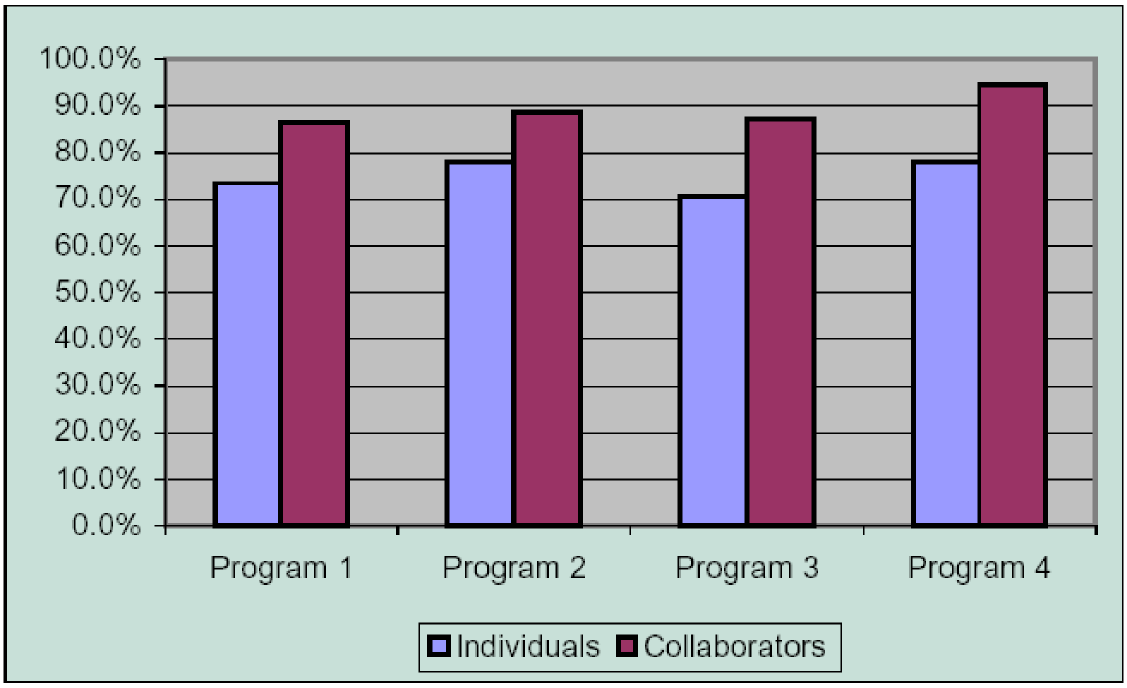


*Figure 3. Bar chart showing average of the percentage of the instructor's test cases passed in Williams' experiment at the University of Utah [Williams, 2000]*

Another issue to discuss when talking about the economic feasibility of Pair Programming is to reject the idea to apply the Pair Programming Practice due to assumption that there will be an increase of the programmers' hours by 100 percent as two programmers are working on the task traditionally dealt with by one programmer. Williams' experiment at the University of Utah [Williams, 2000] indicated that, after an initial adjustment period, the total programmers' hours spent on each program went down dramatically as can be seen in Figure 4. The Pair Programming pairs together spent only about 15 percent more time on the program than the individuals. Additionally after the first program the difference between the times for individuals and for the pairs was no longer statistically significant.

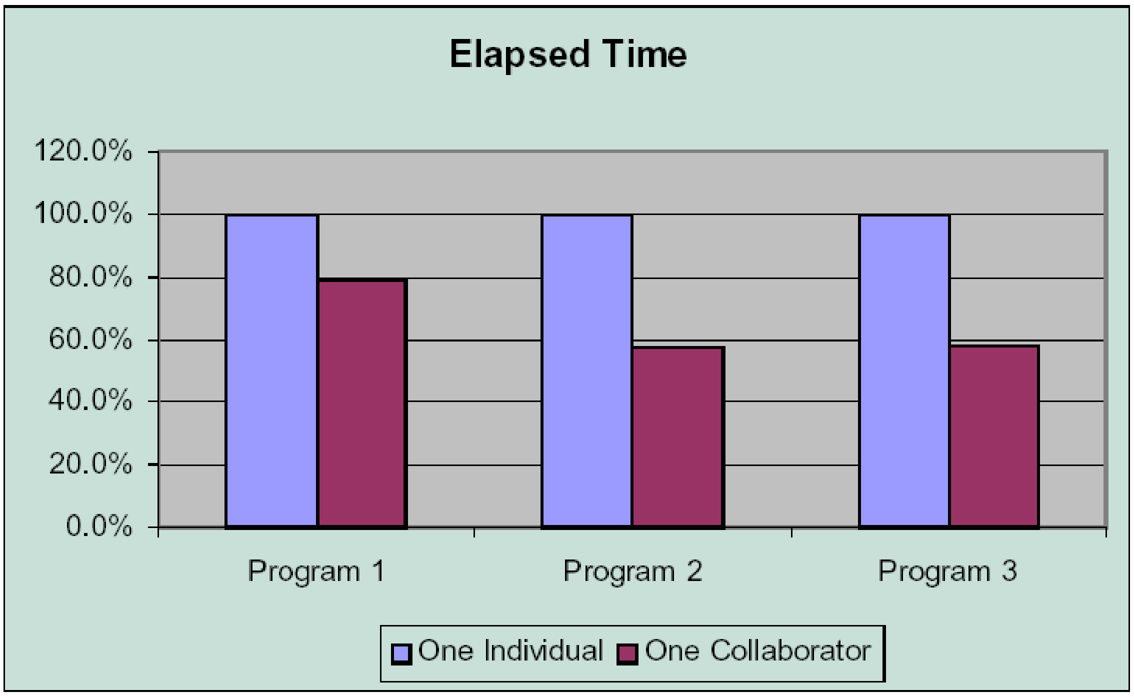


*Figure 4. Bar chart showing elapsed time for one individual and one collaborative working student in Williams' experiment at the University of Utah [Williams, 2000]*

Similar results were found in other studies like [Cockburn and Williams, 2001], [Nosek, March 1998], [Hulkko and Abrahamsson, May 2005], [Lui and Chan, 2003], [Williams and Erdogmus] and [Auvinen, Back, et al., October 2005]. Müller discovered that Pair Programming reduced the time spent on traditional quality assurance activities of a project like reviews by fifty percent [Müller, 2003].

Additional research results in favour of Pair Programming have also shown that Pair Programming improves team communication, enables programmers to enhance their technical skills along their work, reduces the risk to lose key programmers, and is more enjoyable [Williams, 2000], [Williams and Kessler, 2002], [Williams, Kessler, et al., 2000], [Cockburn and Williams, 2001].

From the information given above and as examined by Cockburn and Williams [Williams, 2000], [Cockburn and Williams, 2001], [Williams] as well as Hulkko and Abrahamsson [Hulkko and Abrahamsson, May 2005] and also discussed in "Agile Software Development" [Cohen, Lindvall and Costa, January 2003] the benefits of Pair Programming outweigh the cost. From an economic point of view, it was shown [Cockburn and Williams, 2001], [Williams and Erdogmus], [Auvinen, Back, et al., October 2005] that pairs spent only slightly more effort to develop the same functionality than individuals whilst the code produced by the pairs had measurable fewer defects; since the cost of fixing defects is high, especially in late phases of a software project, Pair Programming is beneficial from an economic point of view. Further on, the same studies showed that the same functionality was programmed within fewer lines of codes by the pairs indicating that the design of the Pair Programming teams was better. Further benefits of Pair Programming are continuous reviews done by the Navigator reducing the effort for separate reviews in the more traditional development methods [Müller, 2003], the possibility to solve problems faster (as two brains work on the same problem and both know the context) and that the task of training on the job is much easier as Pair Programming teams emphasize how much they learned from each other [Cockburn and Williams, 2001] also improving their communication abilities. From the staff and project management point of view, since both parts of the pair are familiar with each piece of code, staff-loss risks are reduced [Williams, 2000].

On the other hand it has to be mentioned here for the sake of completeness that there are other studies of Pair Programming [Nawrocki and Wojciechowski], [Ciolkowski and Schlemmer], [Müller and Tichy, 2001] which either do not strengthen or even contradict the findings mentioned above with respect to the benefits of Pair Programming in comparison to more traditional programming methods.

### 2.1.3 Success Factors of Pair Programming

Prior research gives some critical success factors for effective collaboration between the team applying the practice of Pair Programming [Williams, 2000], [Williams and Kessler, May 2000].

- The pair must cease to consider themselves as two independent individual programmers working in a team and must start to view themselves as a single coherent, intelligent organism working on the given task. This is called Pair-Jelling.
- As in Pair Programming, two developers are assigned to jointly produce one artefact (i.e. a piece of code), the two programmers are jointly responsible for every aspect of this artefact – they share the project ownership. There should be no competition among the collaborating partners. Both must work for a singular purpose, as if the artefact were produced by one person. They are not to blame each other for problems and defects. The pair has to trust each other's judgement and loyalty to the team.
- A large degree of mutual and self respect is needed from both members of the team joining in the Pair Programming practice.
- Excess ego is heavily damaging to the collaborative relationship. Therefore ego-less programming, a term set up by Weinberg in "The Psychology of Computer Programming" [Weinberg, 1998], is crucial for the members of a Pair Programming team to be effective.
- As already discussed, the workspace layout is critical to the success of a Pair Programming team. As will be discussed later on, this success factor has also crucial impact on the tools used, especially in Distributed Pair Programming.
- As already discussed, when applying Pair Programming the Pair Programming pair keeps each other continuously focused and on-task. Due to this fact the practice of Pair Programming can be quite intense and mentally exhausting. Therefore taking regular breaks is important for maintaining the stamina for another round of productive Pair Programming.

Gerken [Gerken, 2003/04] gives some further advises on interpersonal relationship rules that are important in the practice of Pair Programming:

- Both partners must be ready and willing to submerge themselves into the Pair Programming process. They must have a positive attitude to the collaborative work and their partner.
- Each piece of code is developed collaboratively and therefore is also owned by both partners. As mentioned blaming each other for problems and defects is not appropriate. Criticism is to be made constructively and in neutral, team-spirit friendly form. Of similar importance is the readiness to accept constructive criticism.
- For both partners to stay concentrated and have fun in their work, roles are to be exchanged regularly, even and especially if one of the partners has less experience than the other. Special attention is to be put on the Driver not leaving the Navigator behind.
- Each partner is responsible for keeping the other focused and on-task to prevent loss of synergy effects of Pair Programming.
- Pair Programming means that two people focus on one task completely. Therefore diversions are to be prevented as much as possible as this would disturb the process of Pair Programming, break the flow of the work and frustrate the partner. The same is valid for being inattentive.

## 2.2 Distributed Pair Programming

Working in a geographically distributed team came to be today's usual habit for people working in mid- to large-sized software engineering corporations. Globally distributed organizations which are common in international industry today are making such distributed teams an absolute necessity. This trend is even beneficial in many ways, particularly for such people living in geographically disadvantaged areas. Therefore, organizations are asked to adapt their processes, methodologies and practices to the globally distributed scale. Distributed Pair Programming is taking the practice of Pair Programming which is collocated by nature to a non-collocated, geographically distributed environment.

Following the definition taken by Baheti et al. [Baheti, Williams, et al., March 2002], the practice of Distributed Pair Programming can be described as the work of geographically distributed programmers who apply the practice of Pair Programming in such a way that they collaborate synchronously with each other. To collaborate synchronously with each other means that the collaborative work is done at the same time.

By means of this a Distributed Pair Programming pair builds a virtual team. A virtual team can be defined as a group of people who work together towards a common goal but across time, distance, culture and organizational boundaries [George and Mansour, 2002]. As stated by Baheti et al. [Baheti, Williams, et al., 2002] in the context of Distributed Pair Programming the common goal is development of software. The members of such a virtual team need to rely on communication technology to share information, collaborate and coordinate their work efforts.

### 2.2.1 Distance Matters

“Even with all our emerging information and communications technologies, distance and its associated attributes of culture, time zones, geography, and language affect how humans interact with each other.” [Olson and Olson, 2000]

According to Olson and Olson [Olson and Olson, 2000] some characteristics of face-to-face human interaction, particularly with respect to the space- and time-context in which it takes place, is not easy or even impossible to be replicated by the emerging information and communications technologies. Differences in local physical context, time zones, culture and language all persist in spite of the usage of distance technologies.

Olson and Olson list four key-concepts for distributed collaborative work to be successful as a result of their studies [Olson and Olson, 2000]:

Effective communication and collaboration between people requires the communication exchange to take place with respect to some level of *common ground* [Clark, 1996]. Common ground refers to the knowledge the participants of a communication have in common and of which they are aware that they have it in common. But common ground is established not only from knowledge of a communication partner’s background but also on the fly by deducing knowledge of the person’s appearance and reactions throughout communicating with him. As Clark and Brennan show, this is often a collaborative process in which the participants mutually establish what they know so that the conversation can proceed [Clark and Brennan, 1991]. Common ground is constructed from whatever cues about the communication partners are available at the moment. The fewer cues are available, the harder the work is to construct common ground and the more room there is for misinterpretations. Olson and Olson found that in remote communication it is often difficult to establish common ground, particularly in such cases where the communication partners do not know each other well [Olson and Olson, 2000]. One important issue here is also the missing awareness of the state of one’s collaboration partner. In general the more common ground collaborating people can establish, the easier the communication and the greater the productivity.

*Coupling in work* refers to the extent and kind of communication required by the collaborative work. Tightly coupled work, as defined by Olson and Olson [Olson and Olson, 2000], is work that strongly depends on the talents of collections of workers and is non-routine or even ambiguous, whereas loosely coupled work has fewer dependencies or is more routine. Many collaborative tasks in distributed software engineering, amongst which Distributed Pair Programming is to be listed, are tightly coupled tasks. Olson and Olson found in their research that tightly coupled work is very difficult to do remotely and needs heavy technological support [Olson and Olson, 2000] which was not available at the time of their study.

Working in a distributed collaborative environment further on needs a fair amount of *collaboration readiness*. According to Olson and Olson [Olson and Olson, 2000] using shared technology assumes that co-workers need to share information and are rewarded for it. Different fields and work settings engender a willingness to share. If the strategy for progress or productivity involves knowledge management in which people are to give information and seek it from others, an order to collaborate will fail unless it aligns with the incentive structure.

A final key-concept for successful distributed collaborative work is *technology readiness*, meaning that the organization is ready to accept the technology needed to support the distributed collaborative work.

### 2.2.2 Strengthening the Case of Distributed Pair Programming

The findings of Olson and Olson [Olson and Olson, 2000] described above tend to imply that face-to-face Pair Programming teams will most likely outperform Distributed Pair Programmers with respect to sheer productivity.

Nevertheless Baheti et al showed in various studies and researches that software development involving Distributed Pair Programming is comparable to software development involving collocated Pair Programming with respect to the metrics productivity measured in lines of code per hour and quality. In their experiments Distributed Pair Programming teams did not produce statistically significant worse results than collocated teams [Baheti, Williams, et al., 2002], [Baheti, Gehringer and Stotts, 2002], [Baheti, Williams, et al., March 2002].

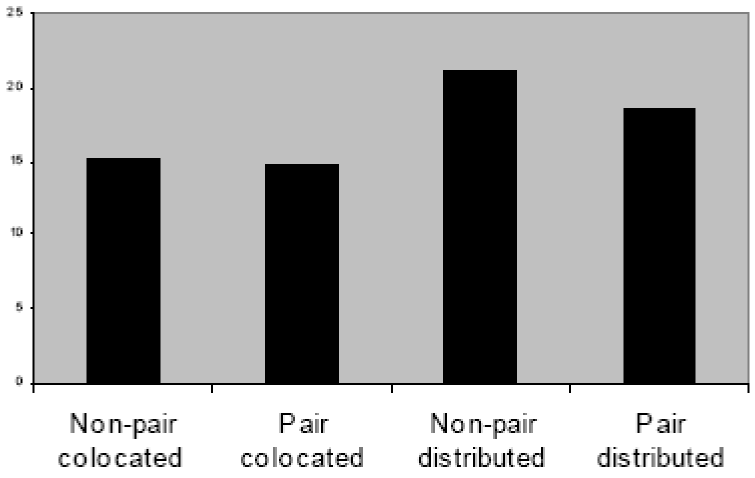


*Figure 5. Bar chart showing average lines of code written per hour in Baheti et al's experiment at the North Carolina State University [Baheti, Williams, et al., 2002]*

The data retrieved by the experiment of Baheti et al. [Baheti, Williams, et al., 2002] was analyzed in terms of productivity, measured by the average lines of code written per hour by each group, and quality, measured in term of the average grade obtained by each group. Figure 5 shows the average lines of code per hour and Figure 6 shows the average grade received by the students participating in the experiment. The result shows that those students applying the Distributed Pair Programming practice in scope of their work did perform quite similar to those students belonging to one of the other experimental groups.

This experiment therefore shows that face-to face Pair Programming does not outperform Distributed Pair Programming and therefore the practice of (Distributed) Pair Programming is quite applicable in distributed software engineering projects. This research has shown that the collocation requirement of the Pair Programming practice can be disregarded as no significant negative effects were observed in non-collocated Pair Programming teams.

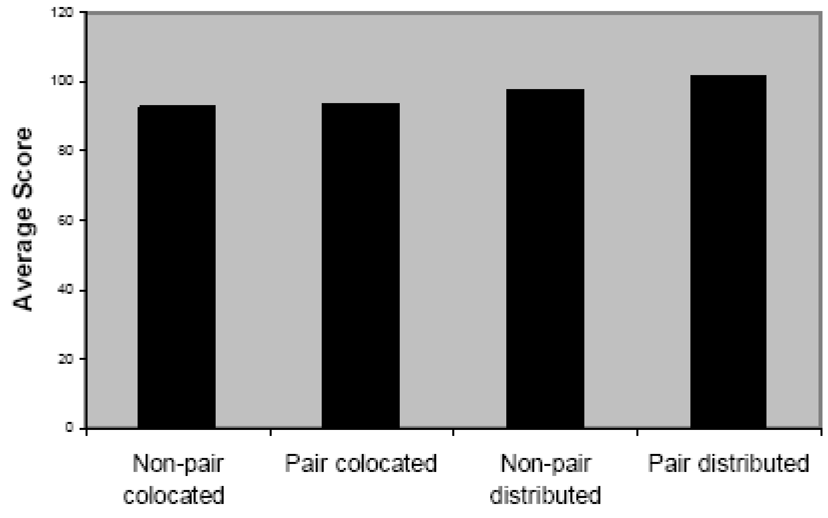


*Figure 6. Bar chart showing average grade received in Baheti et al's experiment at the North Carolina State University [Baheti, Williams, et al., 2002]*

## 2.3 Tool Support for Distributed Pair Programming

"Major corporations launch global teams, expecting that technology will make 'virtual collocation' possible." [Olson and Olson, 2000]

In the past there was no or only very limited tool support for distributed collaborative software development. Advancements in technology and the invention of groupware have changed this situation drastically throughout the last years. The rise of XP and especially the practice of Pair Programming in collocated as well as distributed environments throughout the last years initiated also research into requirements for tools supporting Distributed Pair Programming as well as research and development activities to provide such tools.

### 2.3.1 Requirements on Tools supporting Distributed Pair Programming

Previous research done by Hanks [Hanks, December 2005] in scope of his dissertation provides some ideas on requirements to be fulfilled by tools supporting the practice of Distributed Pair Programming. To get a better understanding of the behaviour a Distributed Pair Programming tool would need to support he observed collocated Pair Programmers to gain insight into their working habits. The following findings where observed by Hanks [Hanks, December 2005]:

- Pairs change roles regularly, but without any formality. Sometimes one programmer just seems to take over. Occasionally, one programmer will be using the keyboard while the second controls the mouse.
- There is frequent gesturing at the screen, associated with comments such as "Look here!" or "What's this?".
- While pairs converse regularly during pairing sessions, they do not spend very much time looking at each other. Their focus is on the computer screen.

Prior to Hanks, Cox and Greenberg [Cox and Greenberg, 2000] researched in the field of distributed collaborative environments. In their work they discuss design principles for the support of collaborating distributed teams. According to them these principles are:

- Provide a common, visually similar environment for all participants.
- Provide timely feedback of all actions within the workspace.
- Support gesturing and deictic references.
- Support workspace awareness.

These observations outline the requirements that are to be fulfilled to some extent by such tools which are used for the support of the Distributed Pair Programming practice.

#### 2.3.1.1 Floor Control

The term ‘floor control’ describes a mechanism that allows one user of a shared application to take charge of it. Only the user in charge can provide input to the application while all the other users can view the application and see its output. The availability of floor control is quite common in distributed meeting tools such as for example Microsoft’s NetMeeting. In distributed computer supported cooperative work (CSCW) applications floor control is maintained in a number of ways as discussed by Greenberg [Greenberg, 1990]:

- The current user can hand off control to another user when he is finished with his task.
- A user can take control of the floor pre-emptively.
- The system can time-slice floor control, giving it to each user in turn. Similarly, the system can hand over floor control after a certain amount of idle time.
- One of the users participates as moderator and is responsible for handing floor control to the other users.
- There can be social floor control, in which the system does not impose floor control technically but instead lets the users develop their own social mechanisms for controlling the floor. In this configuration any user can type at any time and the application merges the input streams.

In case of Distributed Pair Programming the consecutive exchange of the roles of Driver and Navigator can be reflected by the floor control mechanism provided with the software tool used in scope of the distributed collaborative work. In his observations of collocated Pair Programming Hanks [Hanks, December 2005] noticed that in the collocated appliance of the Pair Programming practice the switching of the roles between Driver and Navigator is a very informal and fluid process in which the partners will sometimes exchange roles several times within quite short time periods and thus follows the social floor control pattern defined by Greenberg and mentioned above. Hanks even observed the case that one partner was typing on the keyboard while the other was controlling the mouse. Hanks stated that an application-mediated floor control mechanism that requires the users to take a physical action such as pressing a button or selecting a menu item would potentially hinder this fluid role switching. This view is also supported by Greenberg’s [Greenberg, 1990] observation that, although social floor control tends to be chaotic, it is often effective in small groups.

#### 2.3.1.2 Gesturing

Research done by Tang [Tang, February 1991] showed that gesturing and deictic reference are important elements of group work in small collocated groups accounting for as much as 35 percent of all activities performed in collaborative work. The term ‘deictic’ that is frequently used in literature means “directly pointing at, demonstrative”. A deictic reference can be described as a pointing gesture combined with a verbal clue like “look”, “here” or “this one”. As was found in the research of Cox and Greenberg [Cox and Greenberg, 2000] as well as Hanks [Hanks, December 2005], deictic reference is an important and integral element of collocated Pair Programming.

The most common mean of supporting gesturing in software programs is the cursor or pointer which can be moved around by usage of the mouse. In distributed CSCW applications gesturing can thus be supported by the implementation of remote cursors or telepointers.

The application of cursor or pointers is not restricted to the provisioning of location information. It is common practice that cursor or pointer is also used to convey semantic information to the user as many single-user applications change the shape of the cursor based on the user’s activity to provide semantic feedback to the user. Following this line of thought, the shape of a remote cursor or telepointer can provide semantic information to the other user of the distributed CSCW application as well [Greenberg, Gutwin and Roseman, 1996]. For example, the users name can be attached to each remote pointer to inform the partner of the Distributed Pair Programming pair where his partner is currently focused and working on, the shape of the remote cursor (as well as the local cursor) can signal to the members of the Distributed Pair Programming team who is in which role or, as is the case in single-user applications, the shape of the remote cursor might vary depending on the current activity of its user.

Remote pointers, though they would be quite useful in distributed CSCW applications where collaborating team members work together synchronously are not problem-free. As Gutwin and Penner [Gutwin and Penner, 2002] discuss, network congestion, delay and data loss can lead to telepointer jitter, making it difficult to accurately follow gesturing movements. Dyck et al. [Dyck, Gutwin, et al., 2004] suggest that the cause for telepointers not being implemented in most commercial groupware applications is that the effects of network latency and traffic in realistic situations lead to remote cursors being jumpy, unresponsive and delayed. Nevertheless they stated that this is not as much a problem in distributed applications used by only two people as is the case in Distributed Pair Programming, as coordination is here less of an issue than it is in heavy multi-user groupware environments.

As state by Hanks [Hanks, December 2005] the main focus of the appliance of gestures in Pair Programming is pointing gestures as in most cases the collaborating pair is looking at text based artefacts. They rarely need to trace figures, identify a set of objects, or draw a path through a set of objects. Such tasks would certainly be much more important if the Pair Programming pair would be engaged in other tasks such as developing UML diagrams.

#### 2.3.1.3 Workspace Awareness

In the case of collocated collaborating teams applying the Pair Programming practice each participant is aware of the presence and activities of his partner. When performing Distributed Pair Programming this awareness is not automatically available but needs to be replicated by the application through communicating workspace awareness in some way.

Gutwin and Greenberg [Gutwin and Greenberg, September 1999] define workspace awareness as the “up-to-the-moment understanding of another person’s interaction with the shared space”. In a case where multiple people are working on the same artefact in a collaborating way, each of the collaborating users needs to know who else is participating in the collaborative work, on which piece of the artefact they are working on and what they are doing. As stated by Hanks [Hanks, December 2005] this is of particular importance when using tools that make it possible for the collaborating team members to view different sections of the artefact they are working on. Workspace awareness is the basis for efficient collaboration and communication because the groupware users know what each other can see.

Hanks [Hanks, December 2005] discuss’ the issue of workspace awareness in his studies. He states that issues surrounding workspace awareness are not as important in screen-sharing application because every user is presented with the same view. He states that the highly collaborative nature of Pair Programming also increases workspace awareness as the partners implicitly know where the other one is working. The Navigator can immediately see any changes made by the Driver, and the Driver knows what the Navigator is doing because of the ongoing dialog between them. According to Hanks the availability of a gesturing cursor enabling the partners to use means of deictic references also promotes workspace awareness. Hanks further states that in a distributed environment the Driver is not as aware of the Navigator as he would be if they were collocated. Due to their separation the Driver lacks many of the cues that are available when working collocated. For example the Driver can not tell whether the Navigator is paying close attention to the Driver’s work. An ongoing verbal dialog between them is needed to mitigate this behaviour.

#### 2.3.1.4 Communication channels

Collocated Pair Programming is a highly conversation-intense task [Williams and Kessler, 2002]. From this it is obvious that it is essential for partners joining in a team performing Distributed Pair Programming to be able to speak with each other. Canfora et al. [Canfora, Cimitile and Visaggio, 2003] found that separated Pair Programming team members who are limited to text-based chat tended to stop collaborating and started to work as if they were solo programmers.

The observations made by Hanks [Hanks, December 2005] in his studies suggest that video is no necessary communication channel in Distributed Pair Programming. Similar conclusions were made by other researchers. Baheti et al. [Baheti, Gehringer and Stotts, 2002] studied distributed student teams and provided them with webcams for their distributed projects. They found that the cameras were not used by the students. Olson and Olson [Olson and Olson, 2000] reported that video does not improve the performance of people engaged in a variety of distributed tasks. Schummer and Schummer [Schummer and Schummer, 2001] state that although it is essential for the members of Distributed Pair Programming teams to be able to speak to and hear each other it is not necessary for them to see each other. They justify this with the fact that the focus of the partners lies with the code they are working on and not with each other.

### 2.3.2 Tools supporting the Distributed Pair Programming Practice

Distributed Pair Programming by all means belongs amongst the large field of computer supported cooperative work. This makes the need for software tool support to enable two non-collocated software developers to join in a Distributed Pair Programming session and provide them with similar means to collaborate like in collocated Pair Programming quite obvious.

Seeing the need for software tool support for the Distributed Pair Programming practice, a number of scientists focused their research work around this topic. In this work primary two approaches have been taken with respect to tools supporting Distributed Pair Programming.

The first approach taken is to use commercial off-the-shelf groupware or screen-sharing applications like Microsoft's NetMeeting [NetMeeting] or Virtual Network Computing (VNC) [Richardson, Stafford-Fraser, et al., February 1998], [VNC-Wikipedia], [RealVNC]. All of the tools used for supporting Distributed Pair Programming by this approach share the characteristic that they are screen-sharing applications duplicating the desktop of one user to all the other users connected and thus enabling the remote partner of a Pair Programming team to see exactly what the other partner is doing on his screen. Some research work was done in extending such tools with features specifically designed for means of Distributed Pair Programming. Hanks [Hanks, December 2005] extended VNC with a second pointer to be used by the Navigator for deictic references.

The second approach taken is to develop tools specifically designed for the usage in scope of the practice of Distributed Pair Programming. Among this work are Schummer and Schummer with their TUKAN system [Schummer and Schummer, 2001], Moomba developed by Reeves and Zhu [Reeves and Zhu, 2004], Hypervideo [Stotts, Smith and Williams, March 2002] and Transparent Video Facetop [Stotts, Smith and Gyllstrom, 2004] developed by Stots et al., COPPER developed by Natsu et al. [Natsu, Favela, et al., 2003], or SubEthaEdit developed by Coding Monkeys [SubEthaEdit]. Also to be listed with this second approach are extensions or plug-ins to existing established, commercial off-the-shelf Integrated Development Environments (IDEs) specifically designed for the support of distributed software engineering and thus potentially applicable in Distributed Pair Programming like collab.Netbeans [collab.Netbeans] or various Eclipse plug-ins like PEP - Pair Eclipse Programming [PEP] or Sangam [Sangam].

Please note that all the tools found in scope of the research of this diploma thesis to be potentially useful in supporting the practice of Distributed Pair Programming will be discussed in detail within chapter 4. Therefore this chapter gives only a very rough overview about the available software tool support for Distributed Pair Programming.

# 3 Research Focus of this Work

As discussed in the previous chapter there was already some prior research work done in the area of tool support for the appliance of the Distributed Pair Programming practice. Nevertheless there is still quite a gap to be filled especially when it comes to the provisioning of consolidated lists of available tools that are either specifically designed for or potentially usable for the support of the Distributed Pair Programming practice as well as of desirable requirements on such tools. Another open issue is the possibility to evaluate such tools with respect to their usability for supporting the Distributed Pair Programming practice in a comparable way.

The questions that are addressed in the scope of this diploma thesis to fill the existing research gaps are addressed in this chapter. Further on this chapter describes the approach taken to fill this research gap and provide answers to the addressed research questions.

## 3.1 Research Questions

Distributed Pair Programming would not be possible without software tools that support non-collocated collaboration in a way that simulates the collaborative cooperation of non-distributed Pair Programming. This means that the members of a Pair Programming pair experience the actual Pair Programming activity in a similar way when doing Distributed Pair Programming supported by software tools as if they were sitting side by side in front of one screen.

Such tools are already available as otherwise the practice of Distributed Pair Programming would be a solely theoretical issue. As was already discussed in the chapter dealing with related work there is already some scientific research available that mentions a number of such tools, most of them being specifically designed or enhanced for the support of the Distributed Pair Programming practice though some of them are commonly available screen-sharing applications like Microsoft’s NetMeeting [NetMeeting] or Virtual Network Computing (VNC) [Richardson, Stafford-Fraser, et al., February 1998], [VNC-Wikipedia], [RealVNC].

The range of available tools potentially usable in the appliance of the Distributed Pair Programming practice is larger than those tools specifically designed for the support of the Distributed Pair Programming practice along with some commonly available screen-sharing applications. Among those tools potentially usable for supporting the Distributed Pair Programming practice are to be found tools like commercial and free collaborative editors, Integrated Development Environments (IDEs) that might be specifically enhanced for the support of the Distributed Pair Programming practice, software providing means for online meetings as well as even software that is specifically designed for other kinds of distributed computer supported collaborative work like online review or design tools that also shows potential to be supportive of the Distributed Pair Programming activities.

An extensive list subsuming all tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice that are currently available either from research centres, from commercial side or from free open source software side is missing in the previous research. This diploma thesis aims at providing such an extensive list though it is apparent that it is almost impossible for such a list to be complete.

Evidently there are a number of requirements or properties that are desirable to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice. Though prior research discusses quite a number of eligible requirements for such tools like the support of workspace awareness, the support of floor control for the role changes between Driver and Navigator, the support of deictic referencing by both of the collaborating partners at the same time and the need of various communication channels to enable the Pair Programming pair to effectively and efficiently collaborate, previous research lacks to formulate these general requirements into specific properties of a tool that can be evaluated whether being supported by a selected tool or not.

Besides the desirable requirements resulting from the need to simulate the collocated situation of Pair Programming in a distributed environment there is also a number of other properties that are preferable to be fulfilled by tools used for the support of the Distributed Pair Programming practice as well. Among those are properties like usability, availability of documentation, platform independence and so on. Such properties are completely omitted in the previous research with respect to requirements for tools for the support of Distributed Pair Programming.

Thus the question which requirements are eligible to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice arises. The consolidation of a list of requirements that are desirable to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice is among the aims of this diploma thesis.

As soon as a list of the tools currently available that are either specifically designed for or potentially usable in the support of the Distributed Pair Programming practice is on-hand it comes to mind that some of the tools might be better suited for the support of the Distributed Pair Programming practice than others depending on how good they fulfill the desired requirements of such tools. To find out which tools are better suited for the support of the Distributed Pair Programming practice than others it is necessary to find ways to evaluate the tools potentially usable in the support of the Distributed Pair Programming practice against the consolidated list of requirements that are desirable to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice in a comparable way.

Though there are some evaluations of certain tools in the previous research work as also mentioned in the chapter discussing related work there is no research available until now on providing a comparable evaluation of tools specifically designed for or potentially usable in the appliance of the Distributed Pair Programming with respect to their usability in a Distributed Pair Programming environment. In the scope of this diploma thesis a method for doing a comparable evaluation of tools specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice shall be provided.

The availability of a comparable method of evaluating the usability of today's available tools leads to the question of how good the requirements eligible to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice are covered by today's available tools as well as to the question whether there are any requirements desirable to be fulfilled by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice that are not supported in the currently available tools. This diploma thesis intends to provide answers to these questions as well.

Summarizing this subchapter the following research questions are addressed in scope of this diploma thesis:

- *Available Tools for the Support of Distributed Pair Programming*
  Which tools are available being either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice?
- *Eligible Properties of Tools supporting the Distributed Pair Programming Practice*
  Which requirements or properties are eligible to be supported by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice?
- *Comparative Evaluation of Tools supporting the Distributed Pair Programming Practice*
  How can tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice be evaluated in a comparative way?
- *Availability of Support of eligible Properties of Tools supporting Distributed Pair Programming*
  How good are the requirements or properties eligible to be supported by tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice covered by today's available tools and are there any such requirements that are not supported in the currently available tools?

## 3.2 Research Approach

This chapter describes the research work done to provide answers to the research questions that are to be addressed in scope of this diploma thesis which are mentioned in the previous subchapter. For an overview of the research approach see Figure 7.

### 3.2.1 Candidate Tools for the support of Distributed Pair Programming

As was already stated a first aim of this diploma thesis is to consolidate a list of available software tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice. Such a consolidated list obviously has to cover all those tools already mentioned in previous research work to be supportive in doing Distributed Pair Programming. These tools were found by doing a research of the available literature dealing with the topic of tool support for Distributed Pair Programming. Naturally the tools mentioned in previous scientific work are rather tools that were created in scope of research work, especially when it comes to the category of collaboration-aware applications. There is a significant lack of mentioning commercial or open source tools developed for or potentially usable in doing Distributed Pair Programming in previous scientific work. Therefore besides the research done in available literature additional research was done on the Internet using search engines like Google to complement the list of tools usable for the support of the Distributed Pair Programming practice with commercial and open source software tools that show potential to be usable in scope of the Distributed Pair Programming practice. Furtheron the experience of student apprentices with respect to Distributed Pair Programming and software tools that could be used for this practice was used to finally complete the list of tools that are either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice by discussing this topic with them in a workshop.

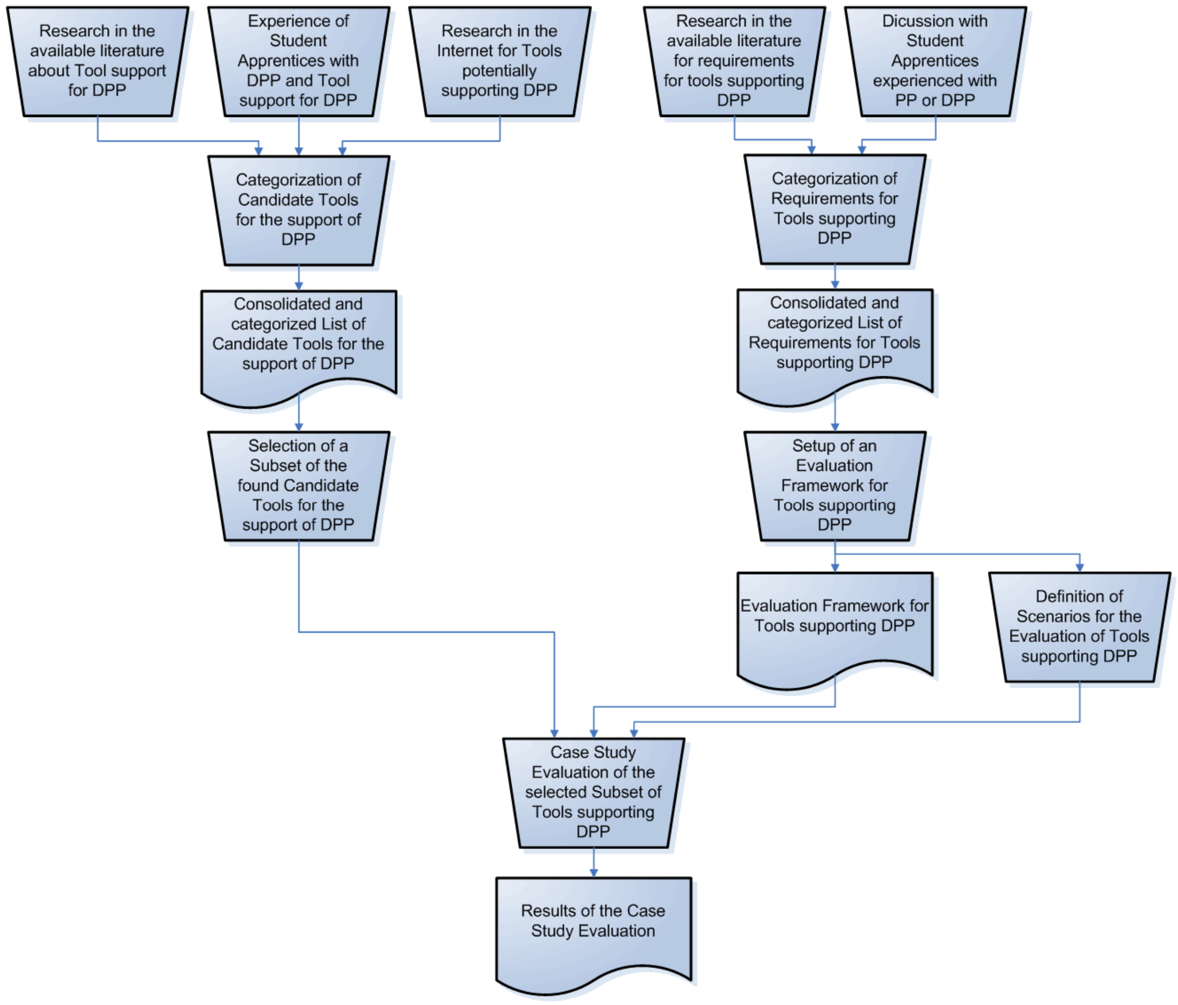

Research in the available literature about Tool support for DPP
Experience of Student Apprentices with DPP and Tool support for DPP
Research in the Internet for Tools potentially supporting DPP
Research in the available literature for requirements for tools supporting DPP
Dicussion with Student Apprentices experienced with PP or DPP
Categorization of Candidate Tools for the support of DPP
Categorization of Requirements for Tools supporting DPP
Consolidated and categorized List of Candidate Tools for the support of DPP
Consolidated and categorized List of Requirements for Tools supporting DPP
Selection of a Subset of the found Candidate Tools for the support of DPP
Setup of an Evaluation Framework for Tools supporting DPP
Evaluation Framework for Tools supporting DPP
Definition of Scenarios for the Evaluation of Tools supporting DPP
Case Study Evaluation of the selected Subset of Tools supporting DPP
Results of the Case Study Evaluation

*Figure 7. Research Approach Overview*

The list created in this way includes a wide range of tools, covering screen-sharing applications along with a variety of approaches of collaboration-aware applications. Thus there is the need to provide some structure or categorization of the tools to make it easier to compare the usability of various categories of tools supporting the appliance of the Distributed Pair Programming practice as well as the usability of tools of the same category. As already discussed there is the obvious distinction between screen-sharing and collaboration-aware tools. But although this distinction provides a major classification of the tools supporting the Distributed Pair Programming practice into two categories that are difficult to compare with each other as the approaches taken are completely different, even within these two categories it is possible to find groups of tools sharing specific similarities that provide a distinction from other groups of tools and therefore a sub-categorisation within these two big categories. This categorisation was done by analysing the available descriptions of the tools found in the consolidated list of tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice. The sub-categories used for the categorisation beyond the main categorisation into screen-sharing and collaboration-aware tools were also found by analysing the documentation of the tools found for peculiar similarities of the tools found.

The result of these activities is a consolidated extensive list of available tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice that is categorized in the two categories of screen-sharing and collaboration-aware tools known from previous work and within these two categories further sub-categorized into categories of tools sharing specific similarities.

### 3.2.2 Desirable Requirements for Tools supporting Distributed Pair Programming

To be able to decide whether a tool that showed potential to be usable for the support of the Distributed Pair Programming practice in the research done into available tools is in fact supportive of effective and efficient Distributed Pair Programming it is necessary to find out which requirements or properties are eligible to be fulfilled by such tools. There are already some ideas on such requirements available in the previous work as also mentioned in the corresponding chapter. These requirements were collected by doing research of the available scientific literature and build up the basis for a consolidated list of desirable requirements for tools supporting the Distributed Pair Programming practice.

Additionally the topic of eligible properties of tools that are to be used to support the Distributed Pair Programming practice and the findings from the research in the literature was discussed with a group of student apprentices with experience in Pair Programming or even Distributed Pair Programming during a workshop. This made it possible to find further requirements not yet mentioned in previous work and extend the list of desirable requirements for tools to be used for the support of Distributed Pair Programming. In scope of this workshop it was also found that some desirable properties that are common for software tools in general like for example usability requirements are also important to be fulfilled by tools used to support the Distributed Pair Programming practice.

The list of desirable requirements for tools supporting the Distributed Pair Programming practice created in this way was a quite long one and needed some structuring. Thus the requirements were subsumed into a number of categories. The definition of the categories was taken over from previous work for those requirement categories intrinsic to the support of the Pair Programming practice in a distributed environment. The other categories were defined in a workshop with student apprentices by analysing and grouping together the found eligible properties of tools that are to be used to support the Distributed Pair Programming practice.

In this way a consolidated list of desirable requirements for tools supporting Distributed Pair Programming evolved along with a categorisation of these requirements to get them better organized.

### 3.2.3 Evaluation Framework for Tools supporting Distributed Pair Programming

Having a consolidated list of tools with potential to support the Distributed Pair Programming practice as well as a consolidated list of desirable requirements for such tools available, the question arises which of the available tools are better in supporting certain requirements as well as the practice of Distributed Pair Programming in general. To provide an answer to this question it is necessary to evaluate the available tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice in a comparable way.

A reasonable approach to provide means for a comparable evaluation of a number of tools is to use the list of requirements desirable for such tools to set up an evaluation framework. The same was done in this diploma thesis to provide a comparable way of evaluating the tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice against the properties found to be eligible to be bolstered by such tools. As a basis for the setup of the evaluation framework the abstract model to select software development tools as defined by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] which is based on the DESMET project [Kitchenham, Linkman and Law, 1997] is used. This model is based on a qualitative analysis of tools and their properties and provides a visualization of an evaluation of several tools. The same is valid for the evaluation framework that is to be set up for a comparable evaluation of the appropriateness with respect to usability, effectiveness and efficiency of tools supporting the Distributed Pair Programming practice. The model described by Zwartjes and van Geffen consists of a tool matrix that is built up by executing five steps [Zwartjes and van Geffen, 2005]:

- Find or refine desired categories of tool support.
- Specify the required tool properties and their importance in the tool matrix.
- Populate the rows of the tool matrix with tools and its cells with indications of compliance.
- Analyze the tool matrix.
- Select a set of tools.

Though the definition of some of these steps needs a little adaptation from the definition as given by Zwartjes and van Geffen the same steps were used for the set up of an generic evaluation framework for tools supporting the Distributed Pair Programming practice. A detailed description on how the generic evaluation-framework for tools supporting the Distributed Pair Programming practice was set up using the steps mentioned above as well as the distinction between the definition of these steps as given by Zwartjes and van Geffen and as applied in this diploma thesis can be found in chapter 5.2. This was done based on the results of a workshop held with student apprentices.

The generic evaluation-framework set up in scope of this diploma thesis might as well be usable for the evaluation of other software tools for computer supported cooperative work with some adaptations with respect to some properties specific to the use in scope of the Distributed Pair Programming practice, the importance of some properties and the description how they are to be evaluated. Nevertheless the generic evaluation-framework as described in this diploma thesis without any adaptations is limited to the usage in evaluating tools for their usability in supporting the Distributed Pair Programming practice.

### 3.2.4 Evaluation of Tools supporting Distributed Pair Programming

Having set up an evaluation-framework for tools supporting the Distributed Pair Programming practice it can be used to evaluate the grade of fulfilment of the eligible requirements of tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice. The result of this evaluation will also show whether there are any requirements desirable to be fulfilled by such tools that are not supported in the currently available tools.

Due to the fact that an empirical evaluation of a number of the found tools with potential to be supportive or which are specifically designed for Distributed Pair Programming was not possible the evaluation in scope of this diploma thesis was done in the way of a series of case studies of selected tools in which each of the tools evaluated was evaluated by a single pair of student apprentices. Due to time and effort constraints, a selection of a subset of the found tools supporting or specifically designed for the Distributed Pair Programming practice was necessary. The target of the evaluation was to cover as high a number of the tools found as possible and to cover the whole variety of approaches to the support of the Distributed Pair Programming practice in the found tools as far as possible. Therefore the following approach was taken to select the tools that were to be included in the actual evaluation.

The categorization of the tools found in scope of the research for this diploma thesis into sub-categories as discussed above provided a basis for the selection of tools in a way that, as far as possible, all these categories are covered by the evaluation. Nevertheless there are other criteria that affect the tool selection for the tool evaluation as well. In general this can be described as two topics, the availability of the tool that is a candidate for evaluation itself as well as the availability of resources needed to operate the tool.

The criterion of the availability of the tool that is a candidate for evaluation itself means simply whether a tool found in scope of the research for this diploma thesis is available for the evaluation or not. There can be a number of reasons why a tool that was found during the research could possibly be unavailable for the evaluation. In some cases it was simply not possible to get hand on the actual tool as it was either not developed yet or the contact found rejected the request to provide access to the tool to include it in the evaluation or ignored the request at all. In other cases there was no trial version but only an expensive licensed version available that made the inclusion of the tool in the evaluation in scope of this diploma thesis impossible.

The availability of resources needed to operate the tool deals with such issues like whether the tool is available for the operating systems available to the evaluation team which were Microsoft's Windows XP and partly Linux, whether the tool's requirements on available hardware and software could be met. Some of the tools found for example rely on the availability of video projectors which were not available to the evaluation team.

Having selected a certain subset of the found tools that are either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice that is evaluated in scope of the series of case studies it is also necessary to think about how the evaluation itself is done. It would be easy to define a special programming example with some hints on what to do and what Pair Programming is about. Nevertheless as in the evaluation each pair of the student apprentices would evaluate a number of tools the evaluation would not be comparable if the same programming example would be developed over and over again by the same people with different tools. Thus in scope this diploma thesis the approach to define a number of scenarios that are to be gone through in each tool evaluation was taken. These scenarios do not define what code or functionality has to be developed in scope of the evaluation but rather define which actions have to be performed to collect the needed information to fill out the evaluation framework for a specific tool. The definitions of the scenarios can be found in the appendix of this diploma thesis.

The result of the evaluation of tools supporting the Distributed Pair Programming practice done in scope of this diploma thesis is the filled out evaluation framework that provides the grade of fulfilment of the eligible requirements of tools either specifically designed for or potentially usable in the appliance of the Distributed Pair Programming practice for the tools evaluated in scope of the case studies. Based on this it can be discussed whether there are any requirements desirable to be fulfilled by such tools that are not supported in the currently available tools.

# 4 Candidate Tools for Distributed Pair Programming

In scope of the work on this diploma thesis a research for all kind of software tools available that provide support to enable two software developers to join in the Distributed Pair Programming practice was done. The result of this research is listed and discussed below.

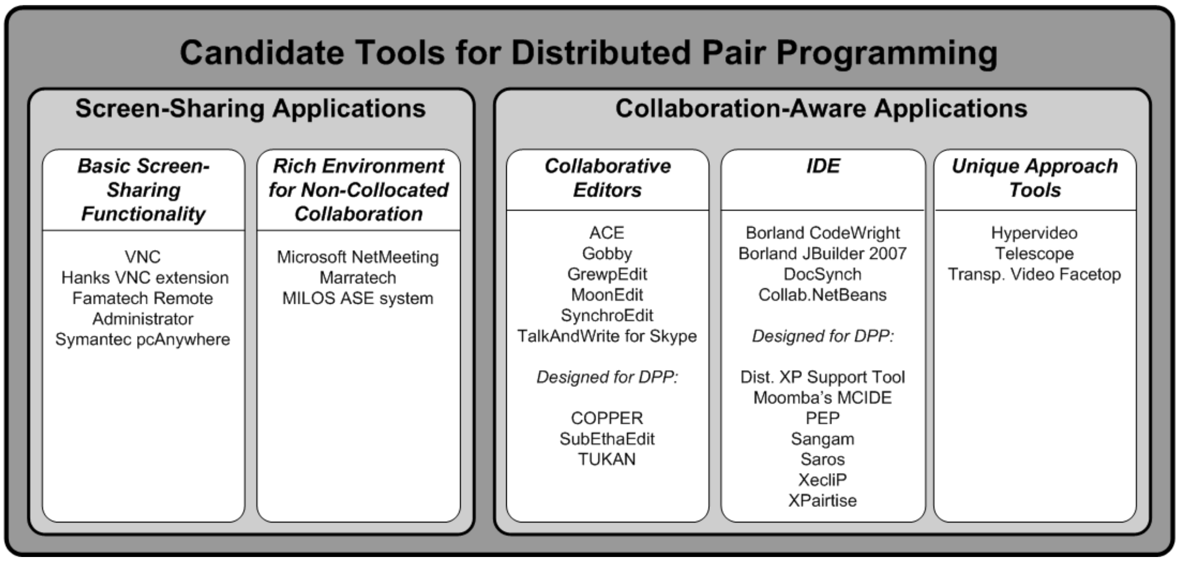


*Figure 8. Overview of Candidate Tools for Distributed Pair Programming*

All the software tools found throughout the research can be categorized into either screen-sharing applications or collaboration-aware applications as already discussed in previous work. Although this distinction provides a major classification of the tools supporting the Distributed Pair Programming practice into two categories that are difficult to compare to each other as the approaches taken are completely different, even within these two categories it is possible to find groups of tools sharing specific similarities that provide a distinction from other groups of tools and therefore a sub-categorisation within these two big categories.

## 4.1 Screen-Sharing Applications

Screen-sharing applications allow multiple users at separate locations to view a common desktop environment. Most of these applications show a client-server like behaviour. The desktop of one user whose application is acting as a server is replicated to all other users' screens which have their applications act as clients and are connected to the server. In this way the situation of the collocated Pair Programming in which both partners sit in front of the same screen is simulated. Screen-sharing applications are usually not specifically designed for distributed software development or the support of the Distributed Pair Programming practice but are rather developed to either provide means for net based meetings or to allow remote system access for system administration and other tasks.

Analyzing the tools found in the screen-sharing tool category and looking onto VNC on one hand and Microsoft's NetMeeting on the other hand the difference between the two tools catches the eye. VNC provides nothing but the basic screen-sharing functionality meaning that the user's screen of the machine acting as VNC server is replicated to the user's screen of the machine acting as client.

Nothing else which has any potential to be an asset in scope of the appliance of the Distributed Pair Programming practice is provided by VNC. When looking at Microsoft's NetMeeting besides the actual screen-sharing there is a far richer environment provided by this tool covering things like a textual chat, a whiteboard, a quite rudimentary but nevertheless existing implementation of a floor control, voice and video transmission and so on. This is due to the fact that NetMeeting is actually not designed as a tool for simple screen-sharing or remote operation of a desktop but rather for the scope of online meetings in which the richer environment which provides tools that would be available in a collocated meeting that takes place in a meeting room is needed to enable the non-collocated meeting partners to successfully collaborate in a similar way as they would in a collocated meeting. These two approaches in screen-sharing applications now provide an additional sub-categorization of this category. One sub-category covers tools that only provide basic screen-sharing functionality like VNC along with its extension for Distributed Pair Programming by Hanks, the Famatech Remote Administrator or Symantec's pcAnywhere. The other sub-category deals with such tools that provide a rich environment for non-collocated collaboration besides the actual screen-sharing functionality. Among this second sub-category belong Microsoft's NetMeeting, the Marratech online conferencing system as well as the MILOS ASE system that actually relies on NetMeeting for its support of the Distributed Pair Programming practice.

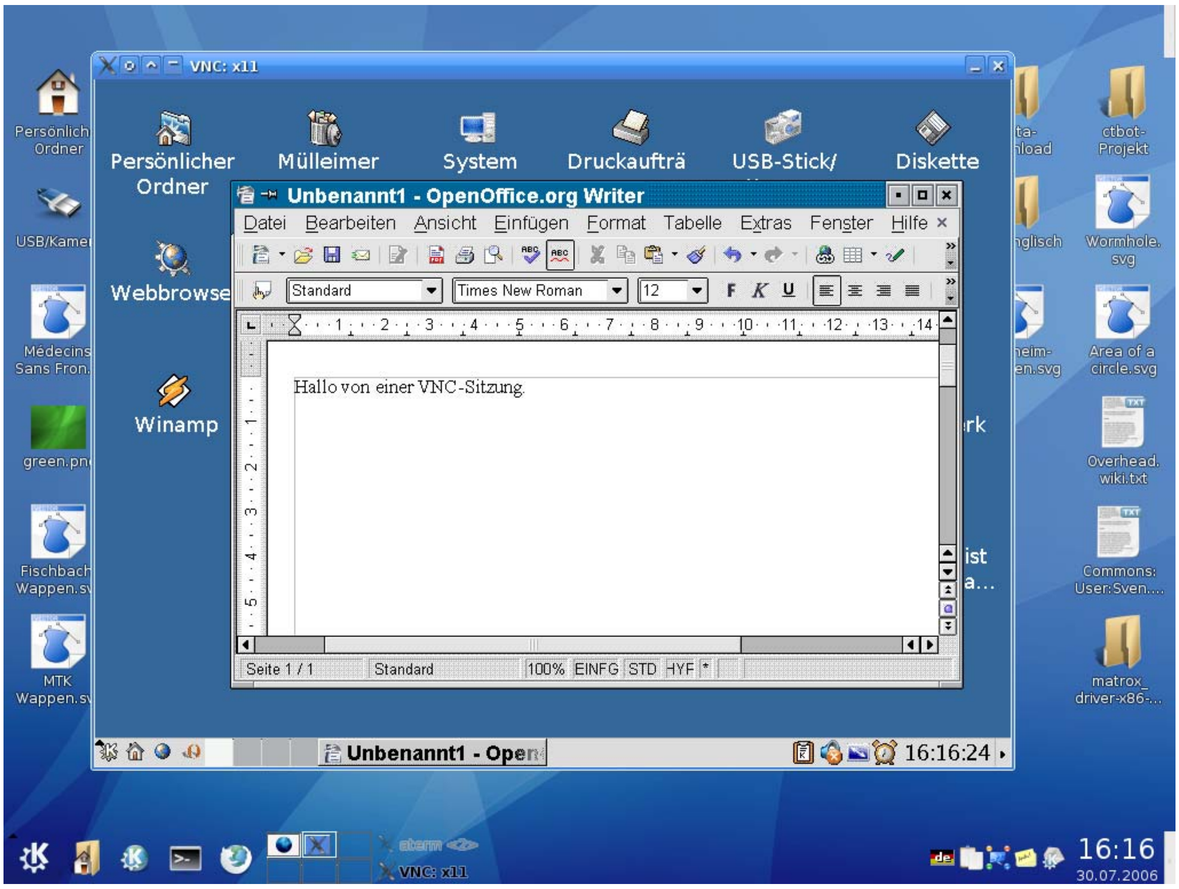
There is quite a range of commercial products that support screen-sharing available on the market. Most of these applications provide a similar functionality belonging to one of the two sub-categories to a more or less extent. The following sections summarize the most prominent and widely known of these screen-sharing tools.

### 4.1.1 Virtual Network Computing (VNC)

Virtual Network Computing (VNC) [Richardson, Stafford-Fraser, et al., February 1998], [VNC-Wikipedia], originally developed by Olivetti & Oracle Research Laboratory and later at AT&T, is a graphical desktop sharing system which displays the desktop of a remote computer on which the VNC-server applications is running on the local computer running the VNC-client application and respectively transmits keyboard inputs and mouse movements of the local computer to the remote one. In this way it is possible to remotely control another computer in the same way as if sitting directly in front of it. Alternately there is also the possibility to use a read-only mode in which it is possible to view the remote computers desktop but no inputs are possible.

A VNC system consists of the VNC-client, the VNC-server and the VNC-communication protocol [VNC-Wikipedia]:

- The VNC-server application is the program that needs to be run on the machine that shares its desktop.
- The VNC-client or VNC-viewer application is the program that watches and interacts with the VNC-server.
- The VNC-protocol is very simple, based on one graphic primitive: "Put a rectangle of pixel data at the specified X, Y position".


VNC: x11
Persönlicher Ordner
Mülleimer
System
Druckaufträ
USB-Stick/
Diskette
Webbrowse
Winamp
Unbenannt1 - OpenOffice.org Writer
Datei Bearbeiten Ansicht Einfügen Format Tabelle Extras Fenster Hilfe
Standard
Times New Roman
12
Seite 1 / 1 Standard 100% EINFG STD HYF
16:16:24
16:16
30.07.2006

Hallo von einer VNC-Sitzung.

*Figure 9. Screenshot: Virtual Network Computing (VNC) [VNC-Wikipedia]*

As stated on RealVNC's homepage [RealVNC], VNC has a wide range of applications including system administration, IT support and helpdesks. It can also be used to support the mobile user, both for hot desking within the enterprise and also to provide remote access at home, or on the road. The system allows several connections to the same desktop, providing an invaluable tool for collaborative or shared working in the workplace or classroom.

VNC is platform-independent. VNC-client and -server applications are available for almost all GUI operating systems. Newer versions of VNC also feature kind of a web server providing a Java-Applet that enables remote access via Java-supporting web browsers even without installed VNC-client software. VNC-viewers of any operating system can usually connect to VNC-servers on any other operating system. It is possible to connect multiple VNC-clients to the same VNC-server at the same time.

The original VNC source code and many modern derivatives are open-source software under the GNU General Public License. RealVNC [RealVNC] provides the official advancement of AT&T's VNC and is as well open-source software under the GNU General Public License. Nevertheless there is only a very limited "Free Edition" available free of charge. Other modern implementations of VNC are TightVNC [TightVNC], UltraVNC [UltraVNC] and x11vnc [x11vnc].

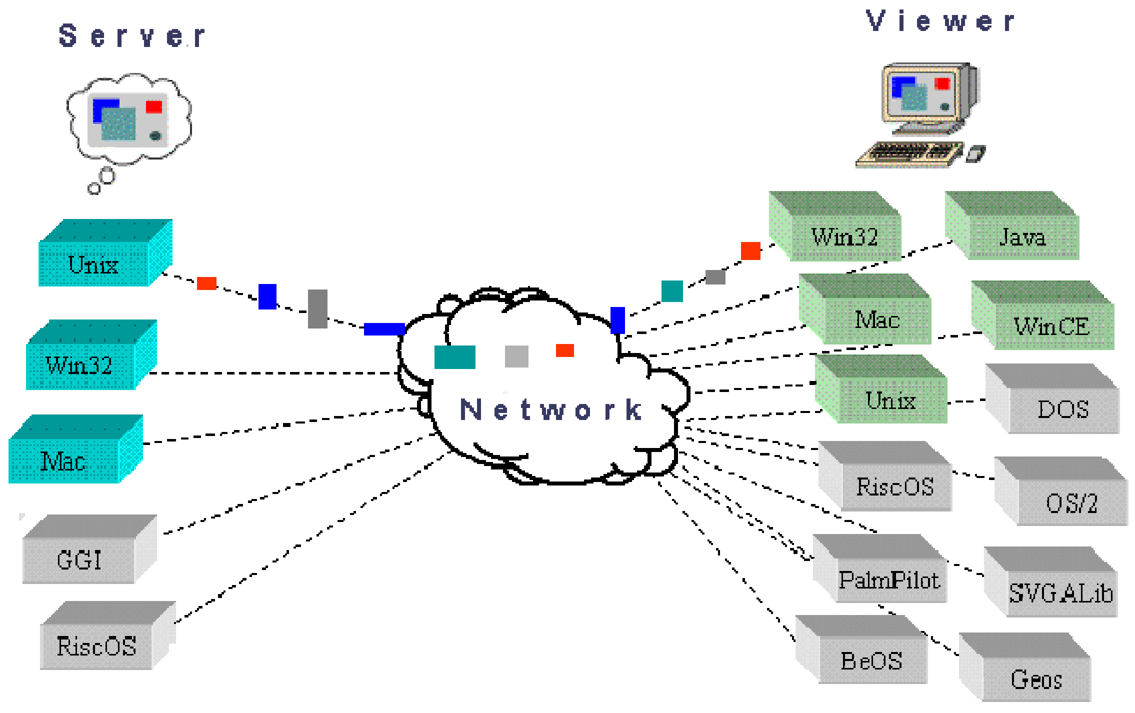


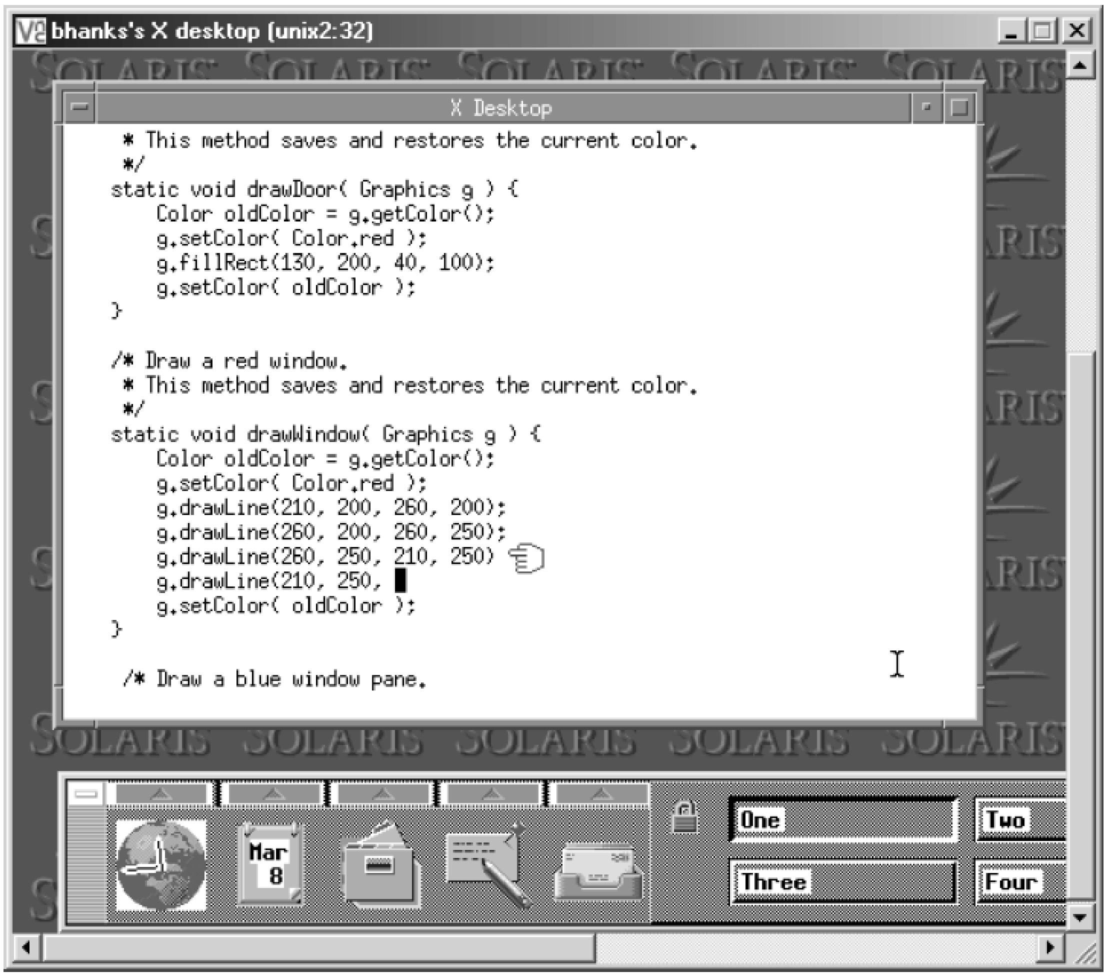
*Figure 10. Virtual Network Computing (VNC) connection possibilities [RealVNC]*

#### 4.1.1.1 VNC Extension for Distributed Pair Programming by Hanks

In his work on empirical studies of Distributed Pair Programming Hanks [Hanks, December 2005] decided to support Distributed Pair Programming by modifying the open-source screen-sharing application VNC. He states that while VNC can be used as-is for Distributed Pair Programming it is not ideal for this as both users have active keyboards and mice and if both partners use their input devices simultaneously their keystrokes are interlaced into an unintelligible stream. Further on, there is only one pointer available making it difficult for the Navigator to point at areas of the screen.

Hanks, feeling that it was critical for a Distributed Pair Programming Tool to support gesturing by the Navigator modified VNC by adding a second telepointer for the Navigator's use in the form of a red hand with a pointing index finger. The speciality of this gesturing telepointer for the Navigator is that it is only present if the Navigator activates it by entering a 'gesturing mode'. This is done by pressing a function key which causes the second pointer to appear on both of the partners' screens. The Navigator can then use his mouse to control the gesturing pointer while the Driver keeps control of the standard cursor. Keyboard and mouse button events of the Navigator are ignored while he is in gesturing mode. When the Navigator exits the gesturing mode by pressing the same function key again, the gesturing cursor disappears and both partners regain full control over the shared standard pointer.

Hanks relied on social floor control when he extended VNC for support of Distributed Pair Programming, motivated by his observations of collocated pair programmers' interaction with respect to floor control. Thus no floor control mechanism is implemented in VNC's extension by Hanks. To enable the partners of a Distributing Pair Programming team to speak to each other, telephones or voice-over-IP applications are to be used. A further modification to VNC done by Hanks is an extension to unobtrusively collect data about the interaction and pairing process in Distributed Pair Programming.



```
 * This method saves and restores the current color.
 */
static void drawDoor( Graphics g ) {
    Color oldColor = g.getColor();
    g.setColor( Color.red );
    g.fillRect(130, 200, 40, 100);
    g.setColor( oldColor );
}

/* Draw a red window.
 * This method saves and restores the current color.
 */
static void drawWindow( Graphics g ) {
    Color oldColor = g.getColor();
    g.setColor( Color.red );
    g.drawLine(210, 200, 260, 200);
    g.drawLine(260, 200, 260, 250);
    g.drawLine(260, 250, 210, 250)
    g.drawLine(210, 250,
    g.setColor( oldColor );
}

 /* Draw a blue window pane.
```

*Figure 11. Gesturing mode of the VNC Extension for Distributed Pair Programming [Hanks, December 2005]*

Hanks VNC enhancement consists of two parts: A server application that runs on UNIX and a client application that runs on Windows. The client connects to the server via Internet and displays a copy of the servers UNIX desktop on the local computer. When two users connect to the same server, both users see the same UNIX desktop. Keystrokes, mouse actions and command output will be displayed on both clients. Both users see the results of the actions by either of them. In this way it is possible to pair program from separate locations since both partners are able to see and interact with the same desktop.

### 4.1.2 Famatech Remote Administrator

Remote Administrator [RemoteAdministrator] is a remote control application for Microsoft Windows developed by Famatech.

Remote Administrator consists of two modules:

- The remote administrator viewer module needs to be installed on the local computer which is used to access the remote computer.

- The remote administrator server module needs to be installed on the remote computer which is to be accessed remotely.

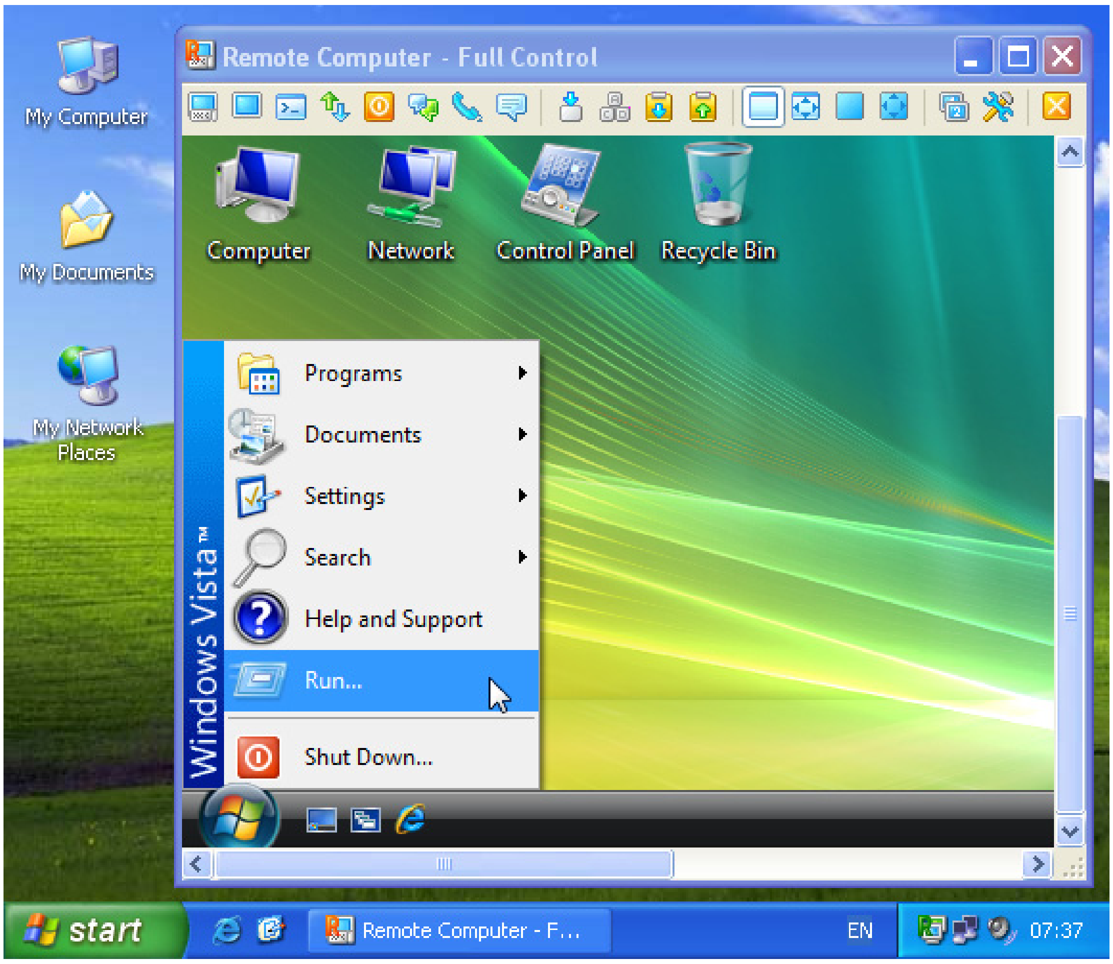


*Figure 12. Screenshot: Remote Administrator [RemoteAdministrator]*

All mouse movements and keyboard signals are transferred from the local computer directly to the remote computer over the network, relaying the graphical screen updates back in the other direction. Remote Administrator makes it possible to remotely access the same computer from multiple places and provides means of advanced file transferring, text and voice chats, remote shutdown, Telnet and other useful features.

### 4.1.3 Symantec pcAnywhere

Symantec's pcAnywhere [pcAnywhere] provides secure, remote access to computers and servers. It provides means to quickly resolve helpdesk and server support issues or stay productive while working away from your office. It is possible to use a desktop computer, laptop or even a mobile device to work across multiple platforms, including Microsoft Windows, Linux and Mac OS X. Further on it is also possible to deploy a limited-functionality, single-user host to computers that do not have a host running.

pcAnywhere's connectivity features help facilitate connections through firewalls, routers, and other types of Network Address Translation devices. Robust security features help to protect computers and servers from unauthorized access. Symantec pcAnywhere is advertised to be able to be used in the following ways:

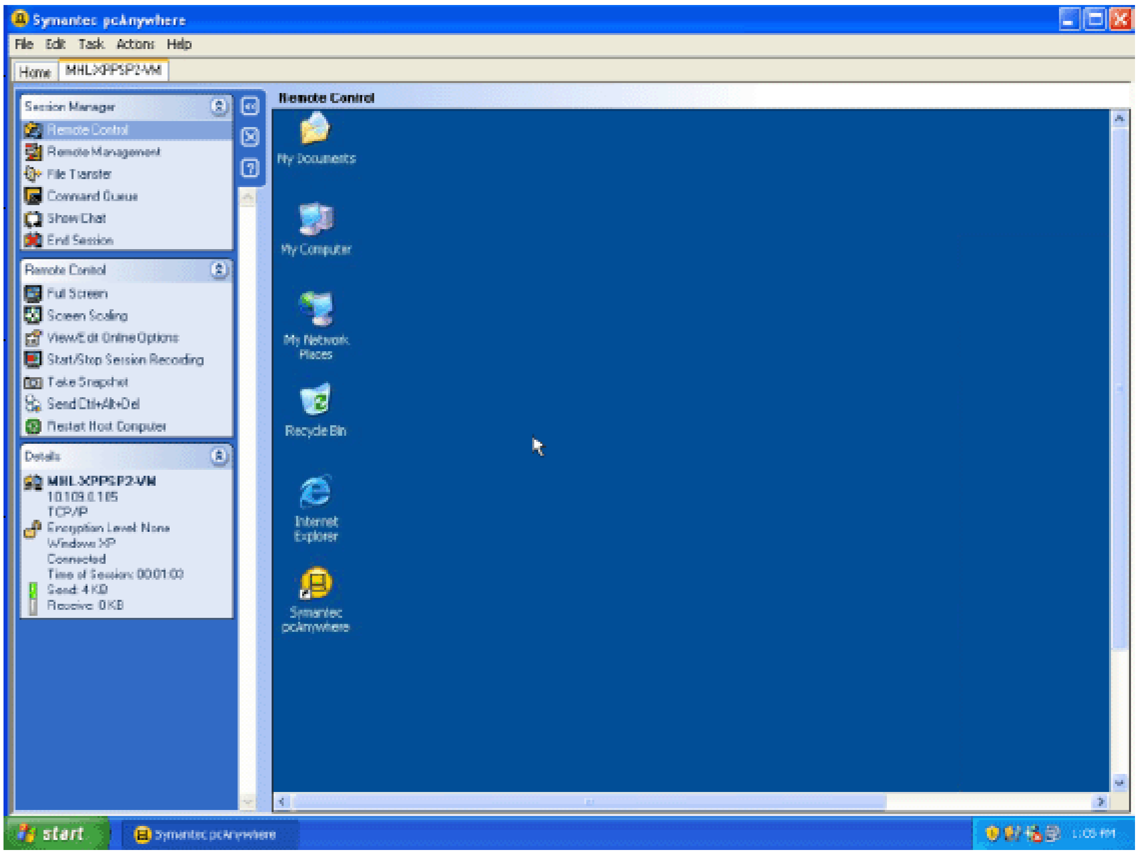
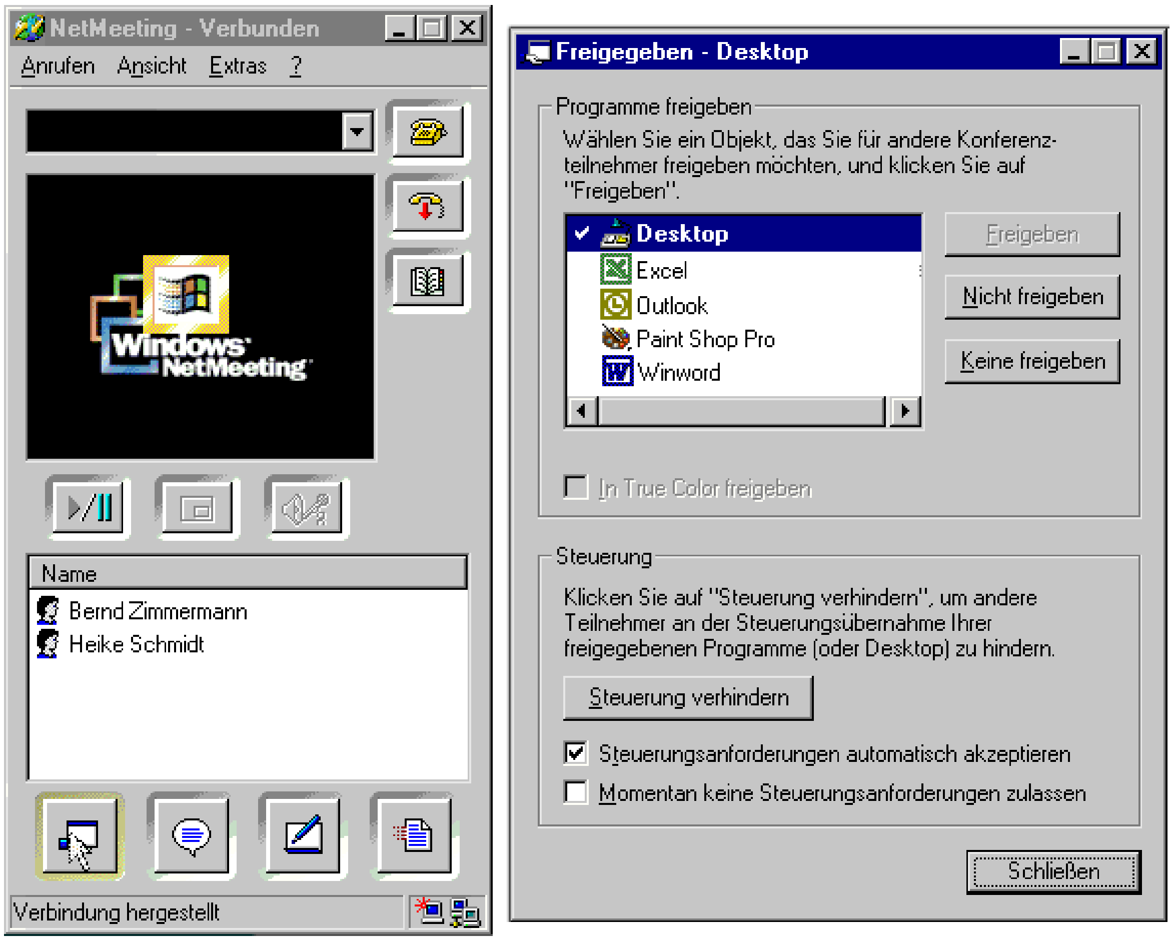


*Figure 13. Screenshot: pcAnywhere [pcAnywhere]*

- Manage computers remotely meaning that it lets helpdesk providers and administrators troubleshoot and quickly resolve computer problems. You can remotely perform diagnostics, check and modify settings, and deploy and install software.
- Support and maintain servers by being able to connect to servers to perform routine maintenance, deploy and install software patches and upgrades, assess performance, and troubleshoot network problems.
- To transfer files between computers. It is possible to perform automatic end-of-day file transfers from one computer to another or exchange any files at any time.
- Working from a remote location is supported by letting the user remotely connect to another computer and work as though he were sitting in front of that computer.

### 4.1.4 Microsoft NetMeeting

Microsoft NetMeeting [NetMeeting] is a voice-over-IP and multi-point videoconferencing application which is already included in Windows 2000 and XP. Unfortunately it seems that Windows XP is the last version of Microsoft Windows to include NetMeeting. With the release of Windows Vista, NetMeeting is no longer included and has been replaced by Windows Meeting Space which provides similar collaboration features but lacks NetMeeting's conferencing features. Nevertheless NetMeeting can still be installed and run on computers running Windows Vista.


NetMeeting - Verbunden

Anrufen Ansicht Extras ?

Name

Bernd Zimmermann

Heike Schmidt

Verbindung hergestellt

Freigegeben - Desktop

Programme freigeben

Wählen Sie ein Objekt, das Sie für andere Konferenzteilnehmer freigeben möchten, und klicken Sie auf "Freigeben".

✓ Desktop

Excel

Outlook

Paint Shop Pro

Winword

Freigeben

Nicht freigeben

Keine freigeben

☐ In True Color freigeben

Steuerung

Klicken Sie auf "Steuerung verhindern", um andere Teilnehmer an der Steuerungsübernahme Ihrer freigegebenen Programme (oder Desktop) zu hindern.

Steuerung verhindern

☑ Steuerungsanforderungen automatisch akzeptieren

☐ Momentan keine Steuerungsanforderungen zulassen

Schließen

*Figure 14. Screenshot: Microsoft NetMeeting [NetMeeting-Screenshot]*

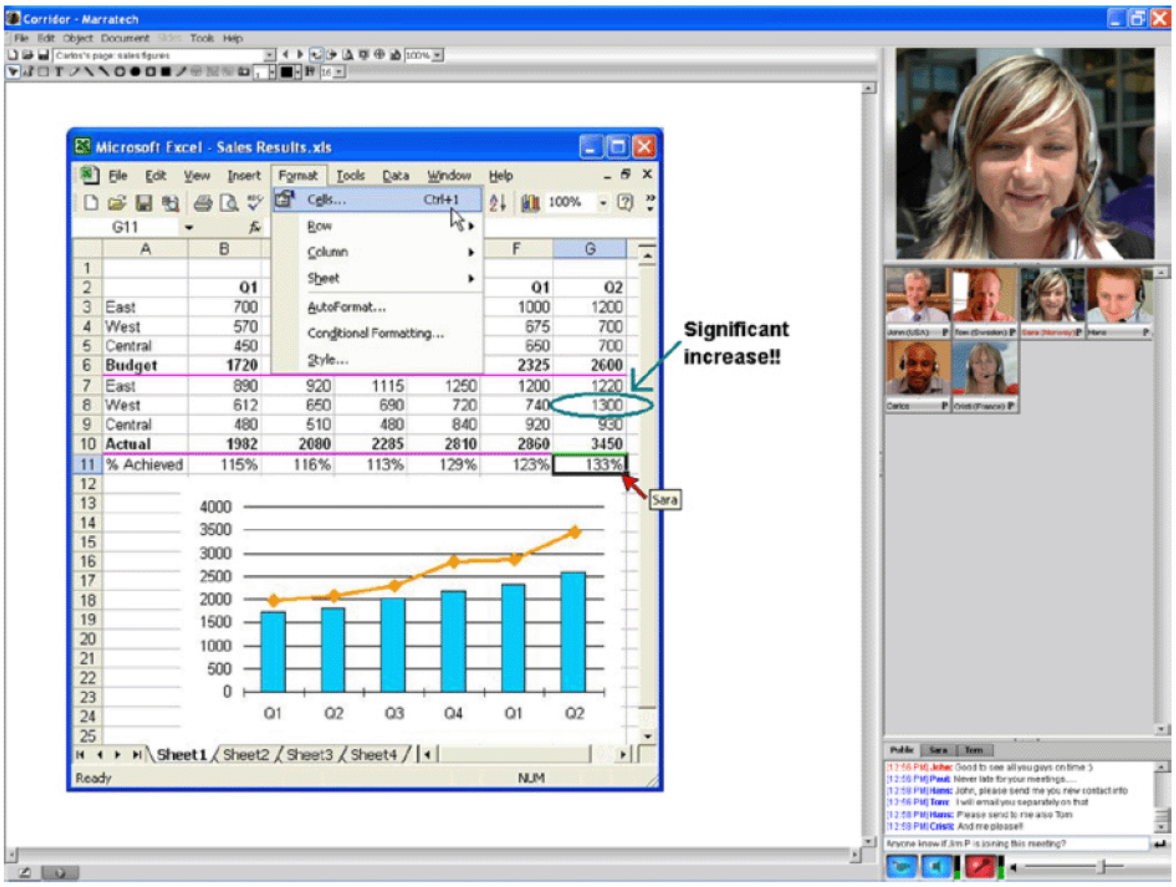
The main features of NetMeeting are:

- Audio and video conferencing.
- A chat that lets the user conduct text-based real-time conversation with all or only selected meeting participants. The chat possibilities are very rudimentary and restricted to the essential.
- A whiteboard that provides means to collaborate via graphic information. Its user interface is quite similar to that of the Windows Paint application but unlike in Paint all objects stay separately move- and removable. The drawing possibilities are restricted to vector graphics and free position able text although it is also possible to paste screenshots.
- A file transfer functionality that makes it possible to send files in the background during a NetMeeting session to all participants.
- Remote desktop sharing makes it possible to view and even operate on a desktop from a remote location.
- Application sharing makes it possible to flexibly share multiple programs during a conference. The user who has the application installed on his computer can share it via NetMeeting. Although all participants of the conference can view what is done with the shared application only one can be in control of it at any given time. In the case that another user wants to take over control of the shared application he needs to request it and can be granted control by the

participant sharing the application who can take back control by simply clicking on the application.

### 4.1.5 Marratech

Marratech [Marratech] is developed by Marratech AB, a global company based in Sweden that focuses its development activities on software solutions which enable remote groups and individuals to collaborate and interact over the Internet. Marratech video conferencing software was recently acquired by Google. It provides features like an easy-to-use interactive whiteboard, application sharing, high quality audio as well as real-time video and messaging on every participant's computer. The vision of Marratech is to put its users into a fully collaborative working environment without the necessity to leave each users desk.


Corridor - Marratech
Microsoft Excel - Sales Results.xls
File Edit View Insert Format Tools Data Window Help
Cells... Ctrl+1
Row
Column
Sheet
AutoFormat...
Conditional Formatting...
Style...
G11
Significant increase!!
Sara
Sheet1 Sheet2 Sheet3 Sheet4
Ready
NUM
Public Sara Tom

| | A | B | | | | F | G |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |
| 2 | | Q1 | | | | Q1 | Q2 |
| 3 | East | 700 | | | | 1000 | 1200 |
| 4 | West | 570 | | | | 675 | 700 |
| 5 | Central | 450 | | | | 650 | 700 |
| 6 | **Budget** | **1720** | | | | **2325** | **2600** |
| 7 | East | 890 | 920 | 1115 | 1250 | 1200 | 1220 |
| 8 | West | 612 | 650 | 690 | 720 | 740 | 1300 |
| 9 | Central | 480 | 510 | 480 | 840 | 920 | 930 |
| 10 | **Actual** | **1982** | **2080** | **2285** | **2810** | **2860** | **3450** |
| 11 | % Achieved | 115% | 116% | 113% | 129% | 123% | 133% |

[12:56 PM] John: Good to see all you guys on time :)
[12:56 PM] Paul: Never late for your meetings.....
[12:58 PM] Hans: John, please send me you new contact info
[12:56 PM] Tom: I will email you separately on that
[12:58 PM] Hans: Please send to me also Tom
[12:58 PM] Cristi: And me please!!
Anyone knew if Jim P is joining this meeting?

*Figure 15. Screenshot: Marratech [Marratech]*

Marratech comes in two parts, a server part called Marratech Manager and a client part called Marratech client. Marratech Manager is a server software solution providing ‘virtual’ rooms in which the collaborative meetings are hosted. It makes them available through a simple web-based interface. The latest version of Marratech Manager supports users accessing meetings hosted by Marratech Manager directly via ‘MeetNow!’, which is a web browser based client available at Marratech. Marratech Manager is a multiple platform tool and runs on a wide variety of server platforms like Windows XP, 2000 and 2003, Mac OS X as well as RedHat and SuSe Linux. Marratech Manager also provides a moderator function. With this functionality the moderator of a meeting is able to either restrict participants to be only receiving audio, video and slides, or allow them to send video, talk and make use of the telepointer. Also full capability including the uploading of slides into the whiteboard, leading the presentation and sharing applications can be assigned to multiple participants.

Marratech client, as already mentioned, can either be launched directly from the web-browser or can be downloaded from Marratech's website to the workstation of the user. By using either of these client versions to connect, any number of users can join web meetings hosted by Marratech Manager. All that is needed are a headset and an optional webcam.

Key features of Marratech's video conferencing software are the possibility to upload a wide range of things like images, documents, live camera snapshots, presentations and other application windows to the Marratech whiteboard while talking to and viewing the other participants of the meeting. Marratech provides telepointers, highlighters, colours and other tools to support its users in the collaborative work. The support of application sharing and working dynamically in the shared applications is a great asset for the collaborative work. It makes sharing information and collaboration possible in real-time across multiple platforms like Windows, Mac and Linux systems. Marratech provides video support enabling its users to see each other in real-time which dramatically increases the sense of presence and the collaboration awareness of the collaborating session participants. Marratech also provides a chat-functionality to enable its users to exchange short textual information. It is possible to chat with all meeting participants as well as to have a private chat with an individual participant only. Additionally Marratech provides the functionality to record and play back various kinds of media like voice, video, whiteboard and chat.

#### 4.1.6 MILOS ASE

MILOS ASE is a project that tries to overcome the XP constraint of collocation by introducing a process-support environment (called MILOS for Agile Software Engineering - MILOS ASE) that helps software development teams to maintain XP practices in a distributed setting. MILOS ASE supports project coordination using the planning game, user stories, information routing, team communication, and Pair Programming [Maurer, 2002].

As also stated in Maurer's work [Maurer, 2002] MILOS ASE actually integrates the screen-sharing tool Microsoft NetMeeting into its approach for means of support of Distributed Pair Programming. Therefore there is actually no new tool for the support of the Distributed Pair Programming practice available with MILOS ASE. Any screen-sharing tool with similar features as provided in Microsoft's NetMeeting, for example Marratech, could be integrated in the MILOS ASE approach for the support of Distributed Pair Programming.

## 4.2 Collaboration-Aware Applications

The most straightforward way to support distributed work is certainly to develop a tool specifically designed for that kind of work. Many researchers as well as a number of commercial developments followed this approach and developed either tools specifically designed for distributed collaborative work or plug-ins for existing IDEs enhancing the same to support distributed collaborative work. Some of these tools are even specifically designed and developed for the appliance in the Distributed Pair Programming practice.

Collaboration-aware applications usually do not support screen-sharing, though for some a screen-sharing functionality might be part of them, but rather support different views on the common artefact worked on based on the roles of the individual users. The common workspace is not simulated by sharing one desktop but rather by providing the common workspace within the collaboration-aware application. Often such tools come with a richer user interface that more effectively supports collaboration than that of screen-sharing applications. Collaboration-aware applications can either be collaborative editors or IDEs specifically developed or enhanced for the support of collaboration of distributed developers.

As there were even more collaboration-aware applications with potential to support the Distributed Pair Programming practice found during the research for this diploma thesis, a further sub-categorization is even more important for this category of tools. Analyzing the descriptions of the tools found, it is possible to sub-categorize collaboration-aware applications in such tools providing only the possibility to edit text or code in a collaborative way, such tools that provide a whole integrated development environment and therefore are specifically designed for the art of software development and such tools or toolsets, that take a completely unique approach onto the support of the Distributed Pair Programming practice.

Those tools providing only the functionality commonly available in text editors for non-collocated collaboration will be termed collaborative editors. It is possibly to sub-categorize the collaborative editors even further into such collaborative editors that are designed for collaborative authoring or review of textual artefacts in general and such collaborative editors that are specifically designed for the support of the practice of Distributed Pair Programming. Collaborative editors that are not specifically designed for the support of the Distributed Pair Programming practice are ACE, Gobby, GrewpEdit, MoonEdit and SynchroEdit as well as the TalkAndWrite plug-in for Skype which all provide fully fledged editor functionality in collaborative environments. Those specifically designed for the support of the Distributed Pair Programming practice are COPPER, SubEthaEdit as well as the TUKAN environment for the support of distributed XP.

In a similar way as done for collaborative editors it makes sense to sub-categorize the IDEs found in the research for this diploma thesis into such designed for non-collocated collaboration but not specifically for the Distributed Pair Programming practice and such that are specifically designed for this practice as well. Collaborative IDEs found that are not specifically designed to be used in Distributed Pair Programming environments are Borland’s CodeWright as well as Borland’s JBuilder 2007. Among those specifically designed for the Distributed Pair Programming practice is actually only one fully fledged IDE, namely Moomba’s MCIDE. All the other tools found that belong into this sub-category are actually plug-ins to existing integrated development environment which provide enhancements so that these IDEs can be used in a Distributed Pair Programming setting. PEP, Sangam, Saros, XecliP and XPairtise are plug-ins to the Eclipse integrated development environment specifically designed for the support of the Distributed Pair Programming practice and therefore belong into this sub-category. Extensions for other IDEs specifically designed for the practice of Distributed Pair Programming are the Distributed XP Support Tool which enhances Microsoft’s Visual Studio.NET for Distributed Pair Programming. Not specifically designed for Distributed Pair Programming are the plug-ins DocSynch which currently is only available for the jEdit development environment but is planned to be made available for other IDEs as well and the NetBeans Collaboration Project that enhances Netbeans with functionality needed in collaboration of distributed teams.

Finally there is the sub-category of collaboration-aware applications the covers such tools or tooling environments that take a completely unique approach to the support of the Distributed Pair Programming practice. Hypervideo, that consists of a dual video projector setup with NetMeeting running on a single PC and supports the collaboration in scope of Distributed Pair Programming in the way that one projector displays the shared desktop and another projector displays a life-sized image of each collaborator to the other, Telescope, which is an Eclipse based audio and video conferencing software, and Transparent Video Facetop, that uses a screen-sharing application for the replication of the desktop used for the collaborative work to both users' workstations and overlays this with semi-transparent images of both of them as if they were sitting side by side so that any gesturing and deictic activity can be seen on the screen by the actions displayed in the image of ones partner.

The following sections describe the collaboration-aware applications and plug-ins for IDEs found in the research done in scope of this diploma thesis that are either candidates or are specifically designed for support of the practice of Distributed Pair Programming.

### 4.2.1 ACE - A Collaborative Editor

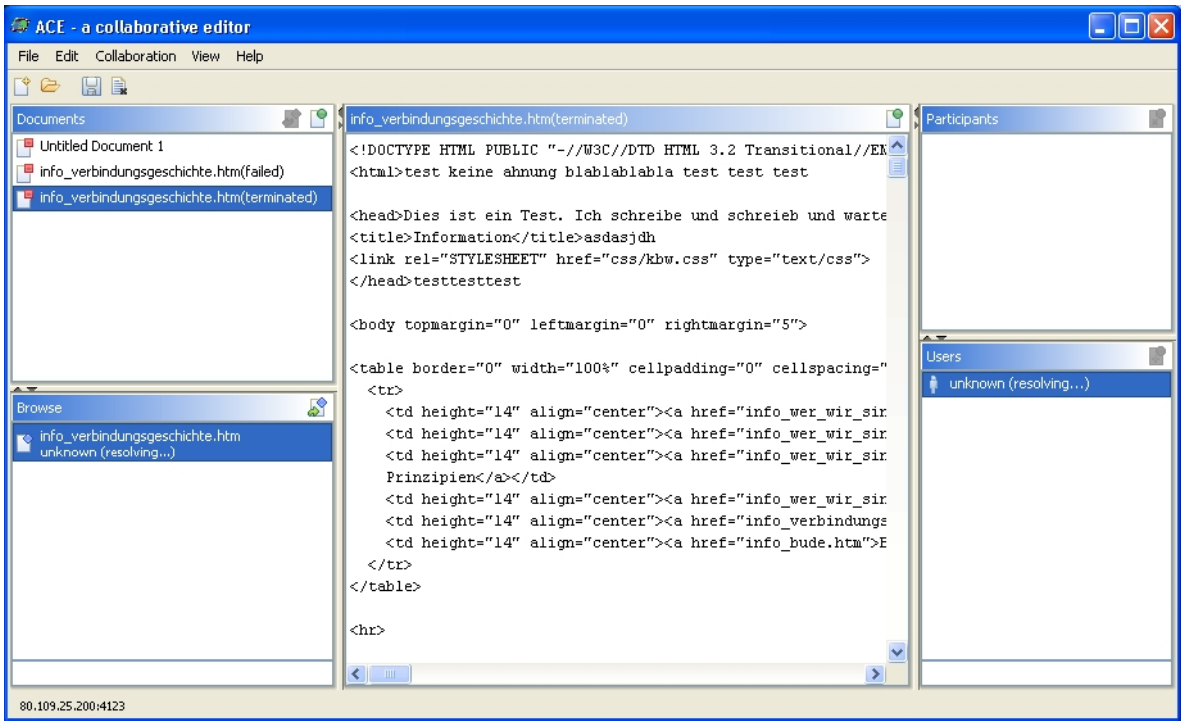
On the homepage of ACE it is stated, that ACE is a platform-independent, collaborative text editor. It is a real-time cooperative editing system that allows multiple geographically dispersed users to view and edit a shared text document at the same time [ACE].

ACE was developed at the Berne University of Applied Sciences, School of Engineering and Information Technology, in scope of a diploma thesis in 2005. ACE is available as open source software at Sourceforge [Sourceforge]. The vision behind ACE, as stated on its homepage [ACE], is that it is the first step on the way to software that allows people to collaboratively edit advanced text processing documents, create a presentation, edit a graphics document or write software in a useful manner.

ACE enables multiple, geographically distributed users to view and collaboratively edit a shared text document at the same time. The users of ACE are not even required to work on the same operating system platform as ACE runs across all major platforms, including Windows, Linux, and Mac OS. A good overview of how ACE works and what can be done with ACE can be found in ACE's User Manual [ACE Usermanual].

For all intents and purposes ACE is basically a simple text editor providing standard features known from such editors like copy and paste or the possibility to store and load text artefacts. It is possible to have opened and edit multiple documents at the same time within separate windows within ACEs GUI.

What enables ACE to be seen as a tool potentially usable for the practice of Distributed Pair Programming is the fact that with ACE it is possible to share documents with other users on different computers connected by communication networks by using its publishing functionality. ACE also provides means to discover remote users and their shared documents automatically in a local area network. Each user of ACE can decide to join any discovered shared document. All of this is possible without any configuration-necessities as ACE is based on zero-configuration networking also known as Bonjour or Rendezvous.


ACE - a collaborative editor
File
Edit
Collaboration
View
Help
Documents
Untitled Document 1
info_verbindungsgeschichte.htm(failed)
info_verbindungsgeschichte.htm(terminated)
Browse
info_verbindungsgeschichte.htm
unknown (resolving...)
info_verbindungsgeschichte.htm(terminated)
Participants
Users
unknown (resolving...)
80.109.25.200:4123

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Transitional//EN
<html>test keine ahnung blablablabla test test test

<head>Dies ist ein Test. Ich schreibe und schreieb und warte
<title>Information</title>asdasjdh
<link rel="STYLESHEET" href="css/kbw.css" type="text/css">
</head>testtesttest

<body topmargin="0" leftmargin="0" rightmargin="5">

<table border="0" width="100%" cellpadding="0" cellspacing="
  <tr>
    <td height="14" align="center"><a href="info_wer_wir_sin
    <td height="14" align="center"><a href="info_wer_wir_sin
    <td height="14" align="center"><a href="info_wer_wir_sin
    Prinzipien</a></td>
    <td height="14" align="center"><a href="info_wer_wir_sin
    <td height="14" align="center"><a href="info_verbindungs
    <td height="14" align="center"><a href="info_bude.htm">E
  </tr>
</table>

<hr>
```

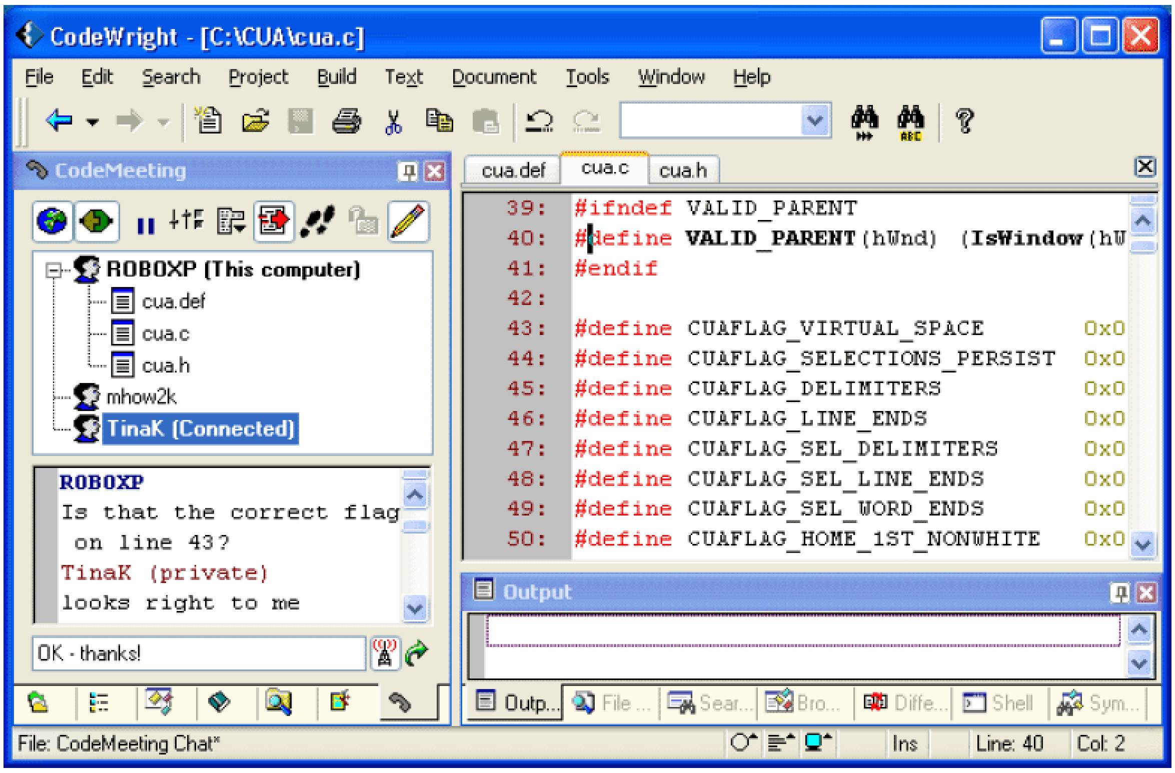
*Figure 16. Screenshot: ACE - A Collaborative Editor*

All of the users that have joined a shared document can freely edit the document at the same time, making all the participants a virtual team. ACE provides collaboration awareness to its users helping to avoid unnecessary conflicts which might otherwise occur when two users edit the document at the same time and text location. ACEs support of collaboration awareness is done by means of showing the cursor and the currently selected text of any of the other users marked with the colour of the respective user. The alignment between user and colour is made by showing the colour next to the users name in the window listing all the participants collaborating on a shared document.

According to ACEs homepage [ACE], the heart of the application is a concurrency control algorithm based on the concept of Operational Transformation which allows for lock-free editing of a document by multiple users, imposes no editing constraints and resolves all conflicts automatically. This algorithm overcomes one of the most significant challenges in designing and implementing real-time collaborative editing systems, namely consistency preservation. That is to ensure that at quiescence, meaning when no messages are in transit, the shared document is identical for all participants.

### 4.2.2 Borland CodeWright

CodeWright is an editor designed for use by software programmers and web content providers. It supports multiple programming languages. Its user manual [CodeWright Usermanual] states that CodeWright is not a portation of a program from DOS, UNIX, or any other operating system but the first professional-quality, extensible programmer's editor written specifically for Windows. It was developed by Borland from 1991 until 2003 but is not supported by Borland any more since the middle of 2005.


CodeWright - [C:\CUA\cua.c]
File Edit Search Project Build Text Document Tools Window Help
CodeMeeting
ROBOXP (This computer)
cua.def
cua.c
cua.h
mhow2k
TinaK (Connected)
cua.def cua.c cua.h
Output
Outp... File ... Sear... Bro... Diffe... Shell Sym...
File: CodeMeeting Chat*
Ins Line: 40 Col: 2

```
39: #ifndef VALID_PARENT
40: #define VALID_PARENT(hWnd)  (IsWindow(hW
41: #endif
42:
43: #define CUAFLAG_VIRTUAL_SPACE          0x0
44: #define CUAFLAG_SELECTIONS_PERSIST     0x0
45: #define CUAFLAG_DELIMITERS             0x0
46: #define CUAFLAG_LINE_ENDS              0x0
47: #define CUAFLAG_SEL_DELIMITERS         0x0
48: #define CUAFLAG_SEL_LINE_ENDS          0x0
49: #define CUAFLAG_SEL_WORD_ENDS          0x0
50: #define CUAFLAG_HOME_1ST_NONWHITE      0x0
```

```
ROBOXP
Is that the correct flag
 on line 43?
TinaK (private)
looks right to me
```

OK - thanks!

*Figure 17. Screenshot: Borland CodeWright [CodeWright Usermanual]*

CodeWright is rather an IDE than a simple editor including features to administer development projects. It supports quite a number of programming languages like C, C++, C#, Pascal, Assembly, xBase, Visual Basic, COBOL, ADA, Java, HTML, XML and more. Based on language templates CodeWright supports language specific features like syntax highlighting.

CodeWright provides instant messaging and remote file editing within its editor, called CodeMeeting, making it potentially usable for the Distributed Pair Programming practice. CodeMeeting makes it possible to share documents with multiple CodeWright users as well as to grant other users the ability to view or edit the shared document, or review remote files.

Further on there is kind of a formal Floor Control mechanism implemented in Borlands CodeWright. By doing corresponding settings in the configuration of CodeWright it is possible to have CodeWright prompt before allowing remote editing. This option displays a dialog prompt when a connected user requests control of a document. If the corresponding configuration setting is not done, the control change is granted without approval. The control over editing in CodeWright is done on a per-document basis. The document owner initially has control. For the document owner as well as any remote user who is in control, shared editing is just like normal editing. Changes will be reflected on the systems of all users who are sharing the document. The document owner can abandon another user's changes by reloading the file. Conversely, none of the non-owners of a document can reload the file to abandon changes.

Another feature of CodeWright which is in support of CodeWrights usability in Distributed Pair Programming is the possibility to make CodeWright follow along any user in a shared session within the same document as well as when switching between documents. This functionality grants that in a Distributed Pair Programming session Driver and Navigator always have the same view on the documents worked on and see the same section of the same document as their respective partner.

### 4.2.3 Borland JBuilder 2007

Borland is advertising its JBuilder 2007 as making collaborative development fast and reliable for Java open source and the web [JBuilder]. It is an application server independent enterprise class IDE built on open source Eclipse.

According to [InfoWorld, March 2007] JBuilder offers impressive collaborative features. Among those it sports a developer-oriented messaging system, which helps with code reviews as well as developer communication. According to [JBuilder Reviewer’s Guide], JBuilder 2007 includes collaboration features that allow local and remote developers to jointly design, edit and debug applications in real time. JBuilder 2007 automatically discovers other clients on the network, allowing users to easily form ad-hoc collaboration sessions.

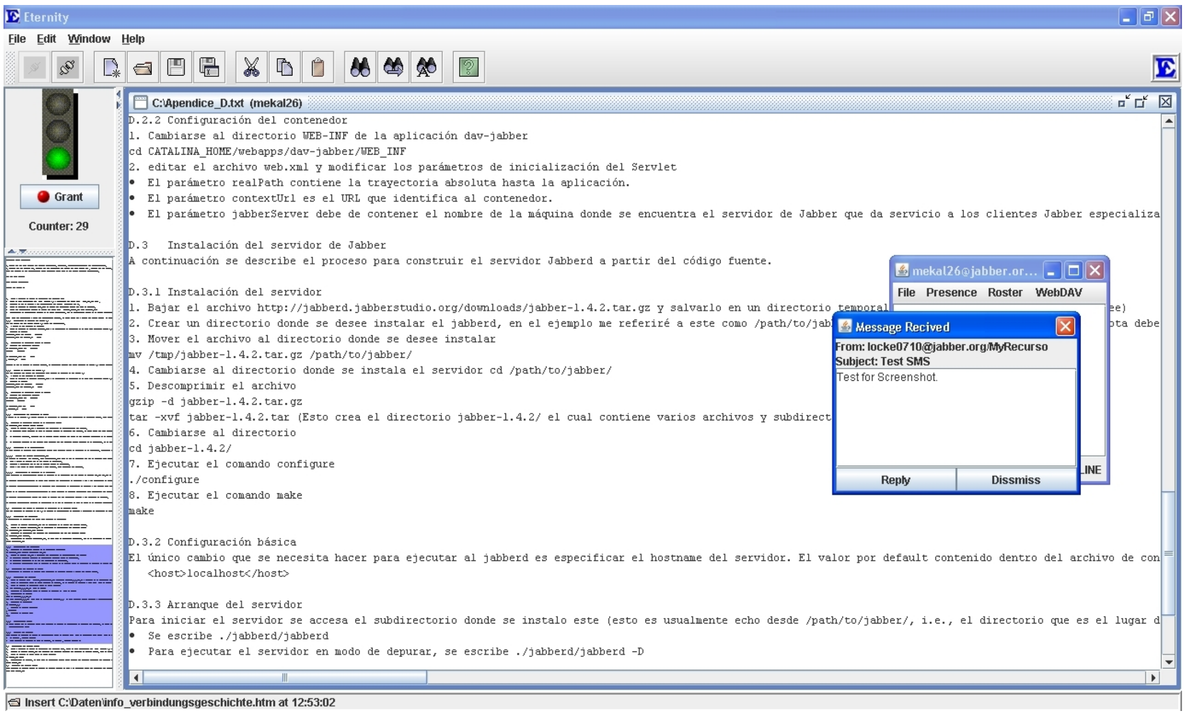
As it was not possible to get a more detailed description of the collaborative features of Borland JBuilder 2007, it can not be clearly stated whether it would be usable in supporting the Distributed Pair Programming practice. Nevertheless it is listed here as the little information available suggests that it could be used in this content.

### 4.2.4 COPPER

COPPER (COllaborative Pair Programming EditoR) was specifically developed to support the practice of Distributed Pair Programming by Natsu, Favela, Morán, Decouchant and Martinez-Enriquez in 2003 [Natsu, Favela, et al., 2003].

The COPPER system follows the client-server architecture and consists of a collaborative editor, a system taking care of user and document presence and an audio subsystem to provide means of verbal communication. On the client side the three subsystems are integrated into an application used to write programs, to access document services and to communicate with peers. On the server side, COPPER consists of a so called instant messaging and presence server, which is responsible for the distribution of messages among the users and maintains user and document presence and a Document server providing access control to document storage and editing.

The editor provided by COPPER can be used either individually or in synchronous collaborative mode. Users can be either collocated or geographically distributed. There is a formal way of floor control implemented in COPPER. It is represented by a traffic light which is activated when working in collaborative mode and includes an action button to request and grant floor control. As COPPER allows the collaborating partners to scroll through the document independently, there is a radar view that provides a general overview of the document being edited to both users. The section of the document in focus of the local user is shown in one colour the section of the document in focus of the collaborating user is shown in a different colour. When these two regions overlap, this is marked using a third colour.


Eternity
File Edit Window Help
Grant
Counter: 29
C:\Apendice_D.txt (mekal26)
mekal26@jabber.or...
File Presence Roster WebDAV
Message Recived
Reply
Dissmiss
LINE
Insert C:\Daten\info_verbindungsgeschichte.htm at 12:53:02

```
D.2.2 Configuración del contenedor
1. Cambiarse al directorio WEB-INF de la aplicación dav-jabber
cd CATALINA_HOME/webapps/dav-jabber/WEB_INF
2. editar el archivo web.xml y modificar los parámetros de inicialización del Servlet
•  El parámetro realPath contiene la trayectoria absoluta hasta la aplicación.
•  El parámetro contextUrl es el URL que identifica al contenedor.
•  El parámetro jabberServer debe de contener el nombre de la máquina donde se encuentra el servidor de Jabber que da servicio a los clientes Jabber especializa

D.3   Instalación del servidor de Jabber
A continuación se describe el proceso para construir el servidor Jabberd a partir del código fuente.

D.3.1 Instalación del servidor
1. Bajar el archivo http://jabberd.jabberstudio.org/downloads/jabber-1.4.2.tar.gz y salvarlo en un directorio temporal                  ee)
2. Crear un directorio donde se desee instalar el jabberd, en el ejemplo me referiré a este como /path/to/jab                          ota debe
3. Mover el archivo al directorio donde se desee instalar
mv /tmp/jabber-1.4.2.tar.gz /path/to/jabber/
4. Cambiarse al directorio donde se instala el servidor cd /path/to/jabber/
5. Descomprimir el archivo
gzip -d jabber-1.4.2.tar.gz
tar -xvf jabber-1.4.2.tar (Esto crea el directorio jabber-1.4.2/ el cual contiene varios archivos y subdirect
6. Cambiarse al directorio
cd jabber-1.4.2/
7. Ejecutar el comando configure
./configure
8. Ejecutar el comando make
make

D.3.2 Configuración básica
El único cambio que se necesita hacer para ejecutar al jabberd es especificar el hostname del servidor. El valor por default contenido dentro del archivo de con
   <host>localhost</host>

D.3.3 Arranque del servidor
Para iniciar el servidor se accesa el subdirectorio donde se instalo este (esto es usualmente echo desde /path/to/jabber/, i.e., el directorio que es el lugar d
•  Se escribe ./jabberd/jabberd
•  Para ejecutar el servidor en modo de depurar, se escribe ./jabberd/jabberd -D
```

From: locke0710@jabber.org/MyRecurso
Subject: Test SMS
Test for Screenshot.

*Figure 18. Screenshot: COPPER*

For means of communication between the collaborating partners COPPER provides audio as well as textual chat communication possibilities.

### 4.2.5 Distributed XP Support Tool

The Distributed XP Support Tool [Atsuta and Matsuura, 2004] is an extension to Microsoft's Visual Studio.NET IDE specifically designed to support the Distributed Pair Programming practice by Atsuta and Matsuura in 2004. As such it provides a number of features needed to enable Distributed Pair Programming activities.

As a most basic requirement for supporting of the Distributed Pair Programming practice the Distributed XP Support Tool features real time synchronization of the content shared between the Pair Programming partners. If the partner acting as Driver performs edit work this is directly displayed on the Navigators screen as well.

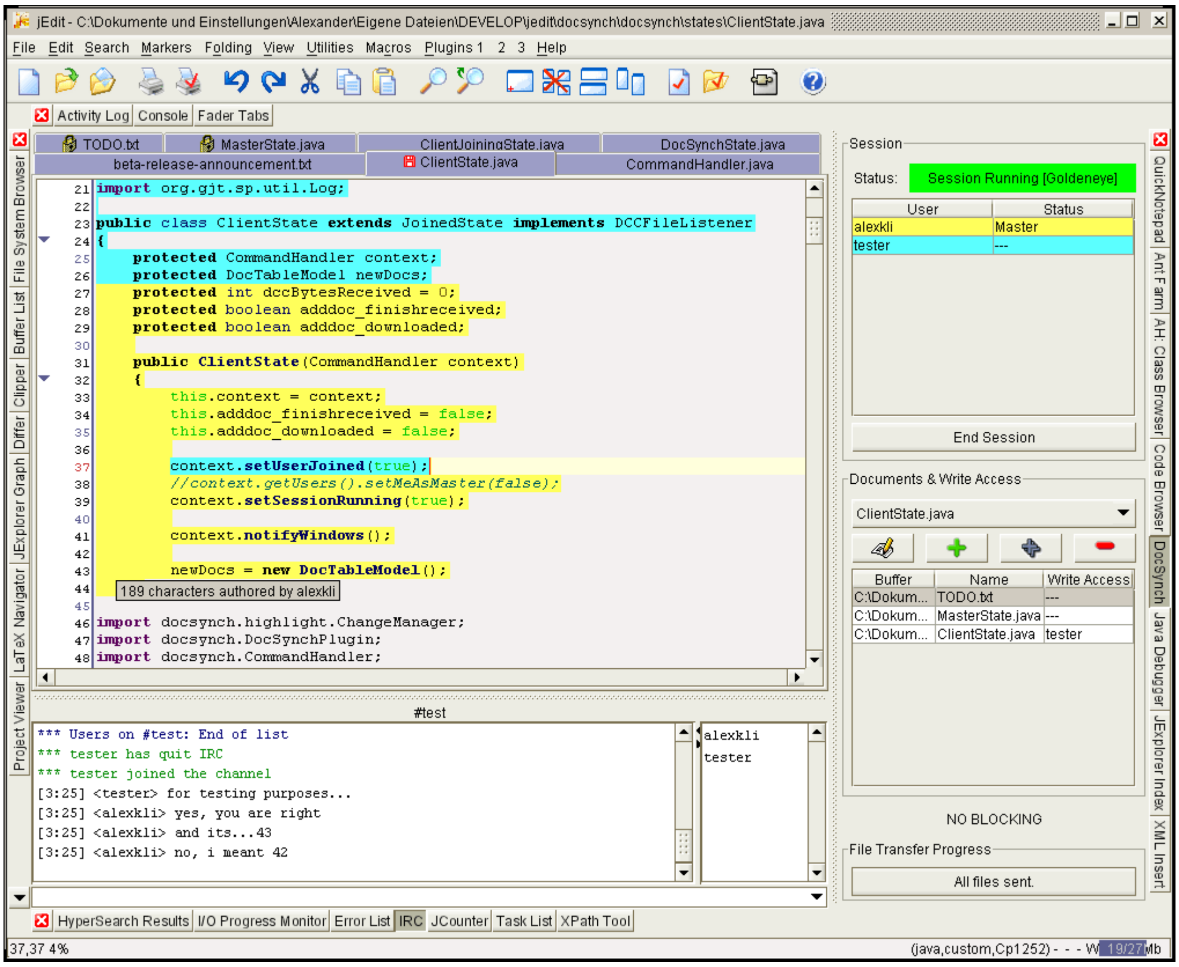
From what is stated in [Atsuta and Matsuura, 2004] it seems that the Distributed XP Support Tool supports both formal as well as informal floor control. In the case of formal floor control a request for a role change is sent by one client to the other and the other responds to it by either granting or denying the role change. An informal role change during the collaborative work seems to be possible by simply taking over the driving role as former Navigator. Unfortunately [Atsuta and Matsuura, 2004] gives no detailed information on how this is implemented in the Distributed XP Support Tool.

The Distributed XP Support Tool provides an interesting functionality called "Notification of the state". What is meant by this term is that if either of the pair drops out of the Distributed Pair Programming session for a while, he can notify his partner about this. If one of the partners set this 'leaving state' it seems that any work of the other partner on the shared artefacts is locked as long as the absence of his partner lasts. Upon rejoining the Pair Programming session the returning partner has to cancel his ‘leaving state’ to enable both partners to continue their collaborative work.

A chatting functionality as well as the possibility to communicate graphically via a whiteboard is supported by the Distributed XP Support Tool.

### 4.2.6 DocSynch

DocSynch [DocSynch] itself is actually not a collaborative editor but rather a system to transform single-user editors into multi-user editors. In this way it allows to remotely edit text documents together and thus is a tool potentially useable in scope of the Distributed Pair Programming practice. Implementations of DocSynch are extensions to existing text editors and IDEs. It is developed as an open source project and available via Sourceforge [Sourceforge].


jEdit - C:\Dokumente und Einstellungen\Alexander\Eigene Dateien\DEVELOP\jedit\docsynch\docsynch\states\ClientState.java
File Edit Search Markers Folding View Utilities Macros Plugins 1 2 3 Help
Activity Log Console Fader Tabs
TODO.txt MasterState.java ClientJoiningState.java DocSynchState.java
beta-release-announcement.txt ClientState.java CommandHandler.java
189 characters authored by alexkli
#test
HyperSearch Results I/O Progress Monitor Error List IRC JCounter Task List XPath Tool
37,37 4%
(java,custom,Cp1252) - - - W 19/27Mb
Session
Status: Session Running [Goldeneye]
End Session
Documents & Write Access
ClientState.java
NO BLOCKING
File Transfer Progress
All files sent.

```
import org.gjt.sp.util.Log;

public class ClientState extends JoinedState implements DCCFileListener
{
    protected CommandHandler context;
    protected DocTableModel newDocs;
    protected int dccBytesReceived = 0;
    protected boolean adddoc_finishreceived;
    protected boolean adddoc_downloaded;

    public ClientState(CommandHandler context)
    {
        this.context = context;
        this.adddoc_finishreceived = false;
        this.adddoc_downloaded = false;

        context.setUserJoined(true);
        //context.getUsers().setMeAsMaster(false);
        context.setSessionRunning(true);

        context.notifyWindows();

        newDocs = new DocTableModel();

import docsynch.highlight.ChangeManager;
import docsynch.DocSynchPlugin;
import docsynch.CommandHandler;
```

```
*** Users on #test: End of list
*** tester has quit IRC
*** tester joined the channel
[3:25] <tester> for testing purposes...
[3:25] <alexkli> yes, you are right
[3:25] <alexkli> and its...43
[3:25] <alexkli> no, i meant 42
```

```
alexkli
tester
```

| User | Status |
|---|---|
| alexkli | Master |
| tester | --- |

| Buffer | Name | Write Access |
|---|---|---|
| C:\Dokum... | TODO.txt | --- |
| C:\Dokum... | MasterState.java | --- |
| C:\Dokum... | ClientState.java | tester |

*Figure 19. Screenshot: DocSynch jEdit plug-in [DocSynch]*

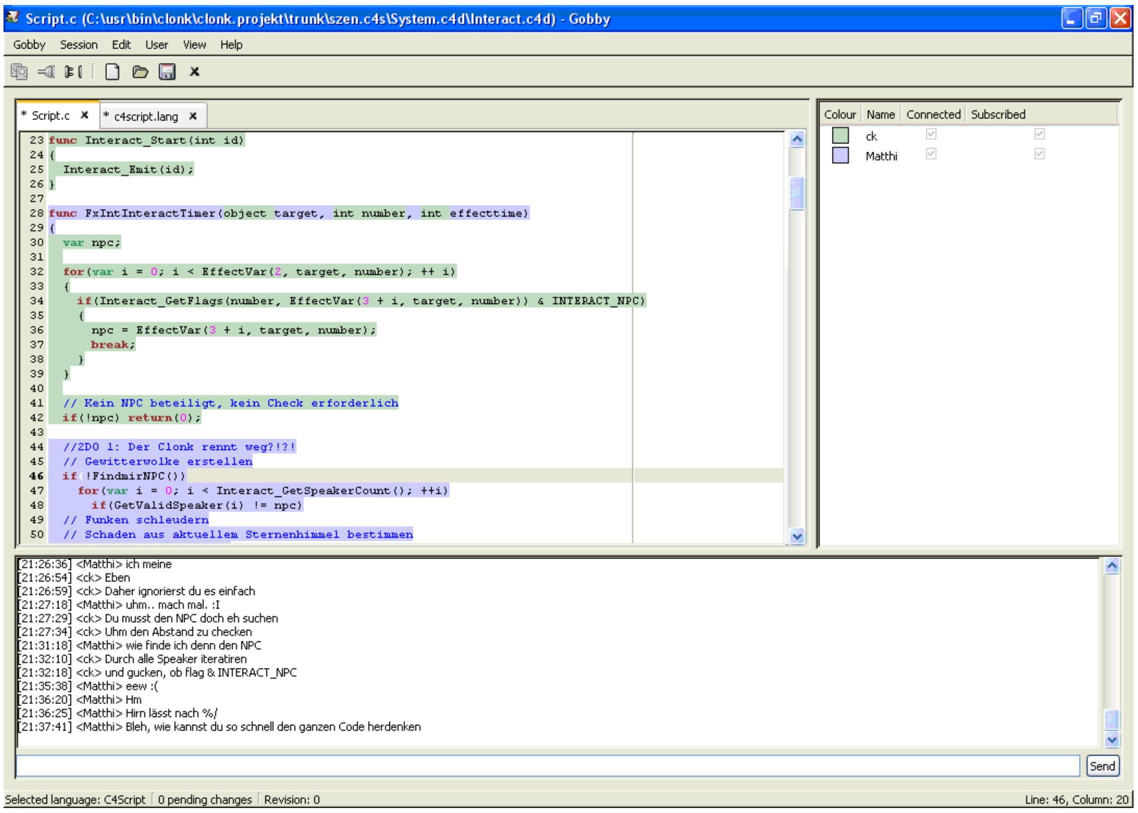
DocSynch is technically based on Internet Relay Chat (IRC). As such it needs an IRC client that fits DocSynch to run as well as and IRC server. The only implementation currently available via the DocSynch Homepage [DocSynch] is one for jEdit which is in late beta status. An add-in for Microsoft Visual Studio .NET, another java plug-in for Eclipse and an extension to VIM are planned to be developed.

DocSynch provides means of a formal floor control. DocSynch restricts write access to a shared document to one single user at any time. Write access needs to be requested or acquired prior to editing a shared document and needs to be released if the editing of the shared document is ended, enabling another collaborating user to acquire write access. It is planned to have DocSynch extended to provide other means of document locking beside the manual one described above. Those still in planning state are auto-lock mode in which the lock is acquired by starting to type, teacher lock mode in which only a user with special rights is allowed to assigns locks respectively grant write access and a real-time mode in which all can type at the same time meaning that with this mode the tool support for floor control is actually removed.

The changes made to a document are highlighted with colours signalling which user was the last to change the respective section of the document. Each user has a unique colour assigned and the text authored by him has that particular colour as background. When the pointer is moved over a coloured part and given a short rest, a tooltip will show up displaying of how many characters the block consists and who has authored it.

### 4.2.7 Gobby

According to the homepage of Gobby [Gobby], it is a free collaborative editor supporting multiple documents in one session and a multi-user chat. It runs on Microsoft Windows, Mac OS X, Linux and other Unix-like platforms. Gobby is developed as free software and licensed under the General Public License (GPL).


Script.c (C:\usr\bin\clonk\clonk.projekt\trunk\szen.c4s\System.c4d\Interact.c4d) - Gobby
Gobby Session Edit User View Help
* Script.c
* c4script.lang
Colour Name Connected Subscribed
ck
Matthi
Send

```
func Interact_Start(int id)
{
  Interact_Emit(id);
}

func FxIntInteractTimer(object target, int number, int effecttime)
{
  var npc;

  for(var i = 0; i < EffectVar(2, target, number); ++ i)
  {
    if(Interact_GetFlags(number, EffectVar(3 + i, target, number)) & INTERACT_NPC)
    {
      npc = EffectVar(3 + i, target, number);
      break;
    }
  }

  // Kein NPC beteiligt, kein Check erforderlich
  if(!npc) return(0);

  //2DO 1: Der Clonk rennt weg?!?!
  // Gewitterwolke erstellen
  if(!FindmirNPC())
    for(var i = 0; i < Interact_GetSpeakerCount(); ++i)
      if(GetValidSpeaker(i) != npc)
  // Funken schleudern
  // Schaden aus aktuellem Sternenhimmel bestimmen
```

```
[21:26:36] <Matthi> ich meine
[21:26:54] <ck> Eben
[21:26:59] <ck> Daher ignorierst du es einfach
[21:27:18] <Matthi> uhm.. mach mal. :I
[21:27:29] <ck> Du musst den NPC doch eh suchen
[21:27:34] <ck> Uhm den Abstand zu checken
[21:31:18] <Matthi> wie finde ich denn den NPC
[21:32:10] <ck> Durch alle Speaker iteratiren
[21:32:18] <ck> und gucken, ob flag & INTERACT_NPC
[21:35:38] <Matthi> eew :(
[21:36:20] <Matthi> Hm
[21:36:25] <Matthi> Hirn lässt nach %/
[21:37:41] <Matthi> Bleh, wie kannst du so schnell den ganzen Code herdenken
```

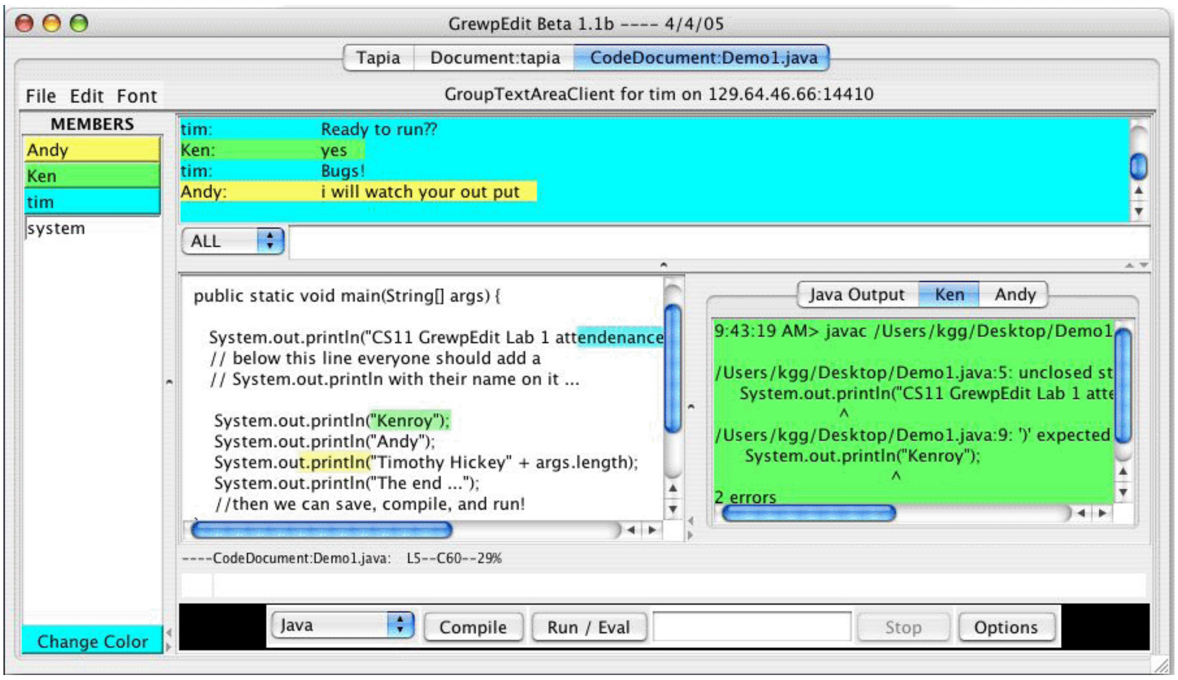
Selected language: C4Script | 0 pending changes | Revision: 0 | Line: 46, Column: 20

*Figure 20. Screenshot: Gobby [Gobby]*

Gobby enables its users to real time collaboration through encrypted channels and provides the possibility to protect collaborative sessions with passwords. Each user has its own changeable colour to be identified by others. Text sections edited by any user will be marked by giving his respective colour as a background. For communications issues with ones partners while coding an IRC-like chat is provided by Gobby. Syntax highlighting for most programming languages is supported. It is possible to edit multiple documents within one session and to drag-and-drop documents into Gobby.

### 4.2.8 GrewpEdit

GrewpEdit [Granville and Hickey, 2005] is a tool developed by the Groupware Research in Education and the Workforce Project team in the Computer Science Department at Brandeis University to support shared editing. GrewpEdit is designed to support close collaboration in software engineering classes and thus is a potential tool for the support of Distributed Pair Programming.



```
tim:      Ready to run??
Ken:      yes
tim:      Bugs!
Andy:     i will watch your out put
```

```
public static void main(String[] args) {

  System.out.println("CS11 GrewpEdit Lab 1 attendenance
  // below this line everyone should add a
  // System.out.println with their name on it ...

   System.out.println("Kenroy");
   System.out.println("Andy");
   System.out.println("Timothy Hickey" + args.length);
   System.out.println("The end ...");
   //then we can save, compile, and run!
```

```
9:43:19 AM> javac /Users/kgg/Desktop/Demo1

/Users/kgg/Desktop/Demo1.java:5: unclosed st
    System.out.println("CS11 GrewpEdit Lab 1 atte
                      ^
/Users/kgg/Desktop/Demo1.java:9: ')' expected
     System.out.println("Kenroy");
                              ^
2 errors
```

*Figure 21. Screenshot: GrewpEdit [Granville and Hickey, 2005]*

GrewpEdit is a cross-platform application that runs on several different versions of Windows, Mac OS X, and Linux platforms. It is a part of the open source project group scheme with a free open source license.

The main functionality of GrewpEdit is to provide a collaborative groupware system where users are able to join groups of other users for collaborative work. These groups can be either with or without password protection.

As soon as a user has joined a group he is able to communicate with the other members that have joined the group by means of an instant messaging like chat. Further on as soon as a document is opened or a new one started all the members that have joined the group can collaboratively edit the document. Each user has a colour assigned which is used as background colour for those sections edited by him which allows to distinct which changes and comments are made by which user and assists consequently the distributed programming process.

### 4.2.9 Hypervideo

In 2002 Stotts et al. [Stotts, Smith and Williams, March 2002] developed a hypervideo system that used video projectors to support Distributed Pair

Programming. It consists of a dual video projector setup with NetMeeting running on a single PC, and the actual hypervideo system called OvalTine. Basically this system works in the way that one projector displays the shared desktop and another projector displays a life-sized image of each collaborator to the other.

So with Hypervideo each partner of a Distributed Pair Programming team works in a specially equipped office that contains two video projectors. One of the projectors is used to project the shared screen on the wall in front of each user; the other is used to project a real-time image of the remote collaboration partner on one of the side walls. The idea behind this setup is that if the partners want to discuss something they will turn sideward to each other as in a collocated Pair Programming setup. Stotts at al. argue that this would make for more natural interactions between the separated partners as the setup simulates the same situation that is experienced in a collocated Pair Programming environment.

### 4.2.10 Moomba

Moomba, developed by Reeves and Zhu [Reeves and Zhu, 2004], is a collaborative environment designed to support distributed XP. This collaborative environment consists of a set of tools integrating needed support mechanisms to facilitate the appliance of a distributed XP process.

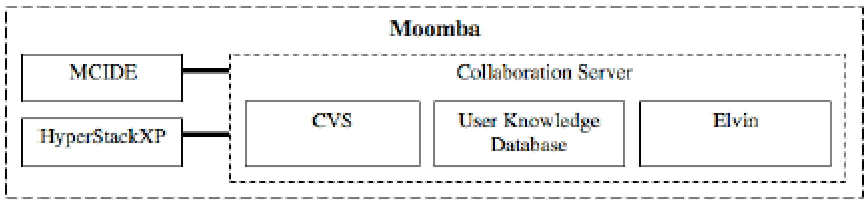


*Figure 22. Moomba's collaborative environment architecture [Reeves and Zhu, 2004]*

According to its overall architecture Moomba mainly consists of three tools:

- HyperStackXP: A web portal for project coordination and tracking.
- A collaborative server: For coordinating manipulation and management of shared artefacts.
- MCIDE: A collaborative programming environment for Distributed Pair Programming.

The web portal as well as the collaborative server is integrated with CVS code and document repositories and a user database.

The HyperStackXP web portal comes with a Pair Programming partner finder which is essentially an advanced search through the user database with the goal of finding a programming partner. It is possible to define search criteria like for example a particular skill level or coding interest. This feature can also be helpful for pairing new users to more advanced and experienced users.

Moomba's Collaborative IDE (MCIDE) is a collaborative editor that supports all the functionality usually found in today's leading development environments including syntax highlighting, code completion, find and replace functionality, indentation and a symbol finder. Two or more users can remotely collaborate on a shared artefact simultaneously. Not only source code files but also any other text based file can be shared and edited in the collaborative editor. The editing is completely unconstrained and users can insert and delete characters at any location. The functionality of Moomba also supports collaborative debugging of programs.

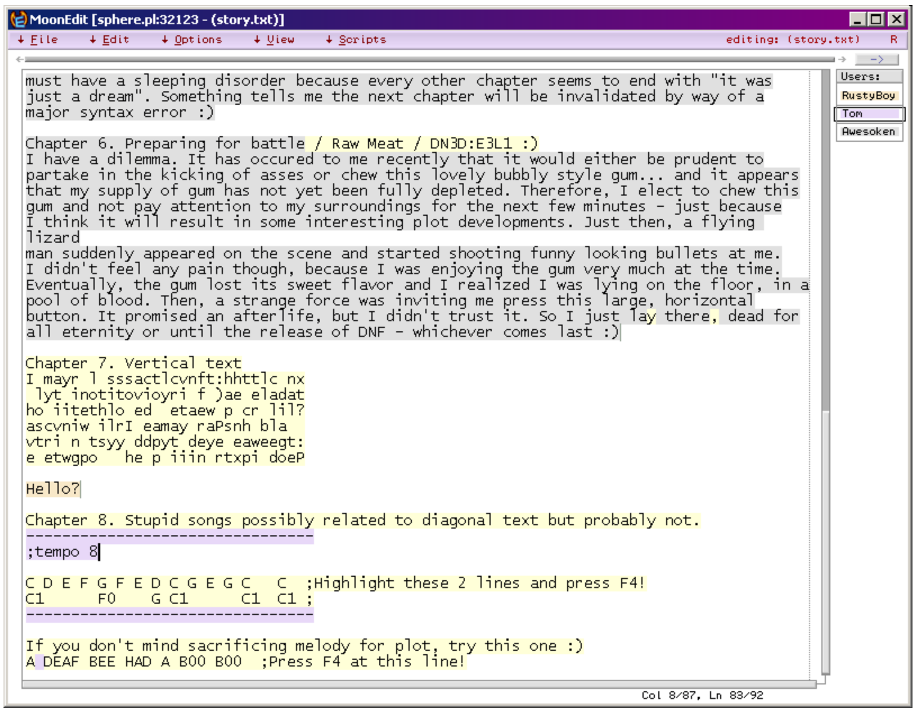
As to enable actual collaboration it is of utmost importance to make all participants aware of each other's changes to an artefact, making it necessary that input generated by each user is distributed to all participants so that consistency of the artefact displayed to each user is maintained.

To provide a sense of workspace awareness Moomba's collaborative editor visualizes contributions from various participants by using different background colours. The participants of a Distributed Pair Programming session done via Moomba's collaborative editor can scroll to any location within the document without affecting what the other participants are doing. As this freedom can lead to reduced workspace awareness it is necessary to provide means to allow participants to be aware of each other's activities. Therefore Moomba features several widgets to counter the lack of reduced awareness.

As stated in [Hanks, December 2005], although Moomba is a collaboration-aware application, Microsoft NetMeeting was used to provide audio and video support during Distributed Pair Programming sessions. Programs executed outside of the Moomba environment can be shared using the NetMeeting session, but programs executed inside of Moomba like the integrated debugger can only display text messages on the remote clients.

### 4.2.11 MoonEdit

MoonEdit [MoonEdit] is a multi-platform collaborative text editor developed by Tom Dobrowolski. It makes cooperative multi-user text editing over the Internet possible to its users.



```
must have a sleeping disorder because every other chapter seems to end with "it was
just a dream". Something tells me the next chapter will be invalidated by way of a
major syntax error :)

Chapter 6. Preparing for battle / Raw Meat / DN3D:E3L1 :)
I have a dilemma. It has occured to me recently that it would either be prudent to
partake in the kicking of asses or chew this lovely bubbly style gum... and it appears
that my supply of gum has not yet been fully depleted. Therefore, I elect to chew this
gum and not pay attention to my surroundings for the next few minutes - just because
I think it will result in some interesting plot developments. Just then, a flying
lizard
man suddenly appeared on the scene and started shooting funny looking bullets at me.
I didn't feel any pain though, because I was enjoying the gum very much at the time.
Eventually, the gum lost its sweet flavor and I realized I was lying on the floor, in a
pool of blood. Then, a strange force was inviting me press this large, horizontal
button. It promised an afterlife, but I didn't trust it. So I just lay there, dead for
all eternity or until the release of DNF - whichever comes last :)

Chapter 7. Vertical text
I mayr l sssactlcvnft:hhttlc nx
 lyt inotitovioyri f )ae eladat
ho iitethlo ed  etaew p cr lil?
ascvniw ilrI eamay raPsnh bla
vtri n tsyy ddpyt deye eaweegt:
e etwgpo   he p iiin rtxpi doeP

Hello?

Chapter 8. Stupid songs possibly related to diagonal text but probably not.
--------------------------------
;tempo 8

C D E F G F E D C G E G C   C  ;Highlight these 2 lines and press F4!
C1      F0    G C1      C1  C1 ;
--------------------------------

If you don't mind sacrificing melody for plot, try this one :)
A DEAF BEE HAD A B00 B00  ;Press F4 at this line!
```

*Figure 23. Screenshot: MoonEdit [MoonEdit]*

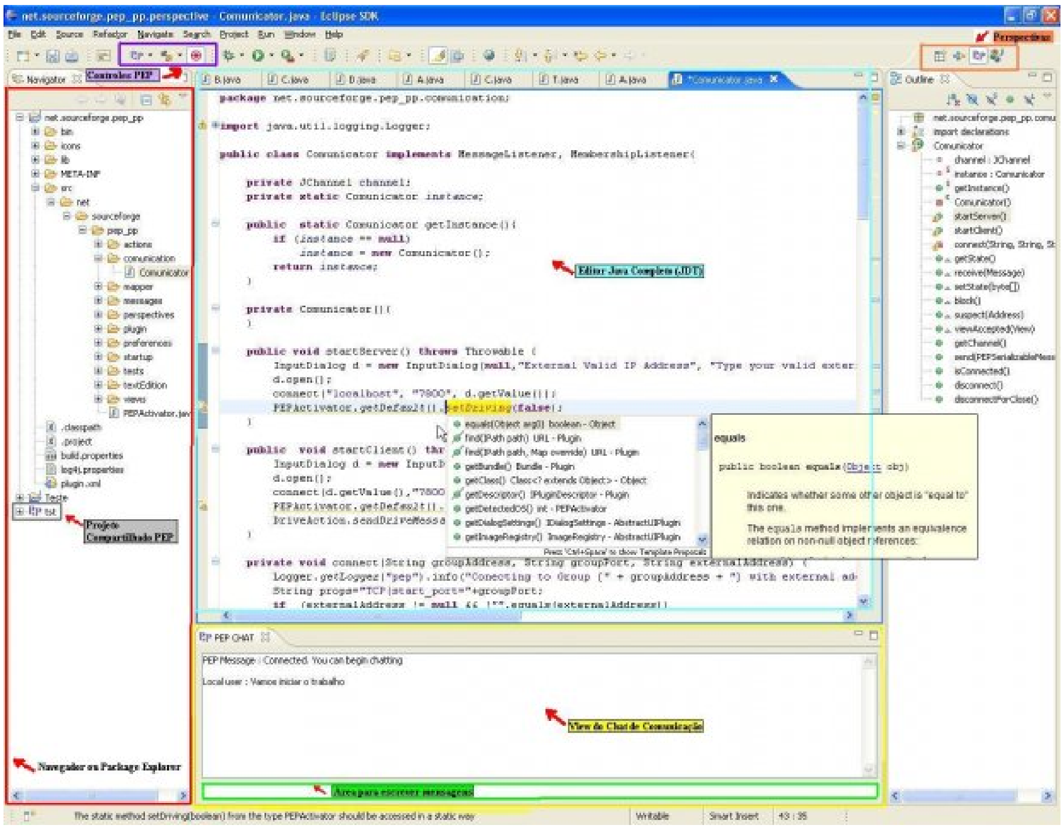
When a number of collaboration partners join in a session each of the collaborating users can edit the shared document. To provide means of workspace awareness and hint to the other users which text was typed by whom multiple text cursors are visible on the screen and each user has its own colour. Any movement of the cursor or change of the artefact is simultaneously visible on the screens of the other users.

MoonEdit comes with a change-log feature that makes it possible to view and undo the history of changes done to an artefact. This is an asset especially in such situations, where an artefact is edited by multiple users and possibly even offline.

The keyboard typing sound simulation of MoonEdit contributes to MoonEdits workspace awareness in communication to all collaborating users that somebody is typing.

### 4.2.12 PEP - Pair Eclipse Programming

PEP [PEP] is an Eclipse plug-in supporting the Distributed Pair Programming practice for Code written in Java. It is developed by Brazilian students of the Federal University of Santa Catarina as Open Source software. Unfortunately there is almost no description of its features available. The only information given on its homepage is that it features code (re)synchronization, has possibilities for chatting included, that it does not need a server to run and that it strictly adheres to the Pair Programming principles.


net.sourceforge.pep_pp.perspective - Comunicator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Controles PEP
Perspectivas
Navigator
Outline
Editor Java Completo (JDT)
Projeto Compartilhado PEP
View de Chat de Comunicação
Navegador ou Package Explorer
Área para escrever mensagens
PEP CHAT

```
package net.sourceforge.pep_pp.comunication;

import java.util.logging.Logger;

public class Comunicator implements MessageListener, MembershipListener{

    private JChannel channel;
    private static Comunicator instance;

    public  static Comunicator getInstance(){
        if (instance == null)
            instance = new Comunicator();
        return instance;
    }

    private Comunicator(){
    }

    public void startServer() throws Throwable {
        InputDialog d = new InputDialog(null,"External Valid IP Address", "Type your valid exter
        d.open();
        connect("localhost", "7800", d.getValue());
        PEPActivator.getDefault().setDriving(false);
    }

    public  void startClient() thr
        InputDialog d = new InputD
        d.open();
        connect(d.getValue(),"7800
        PEPActivator.getDefault().
        DriveAction.sendDriveMessa
    }

    private void connect(String groupAddress, String groupPort, String externalAddress) {
        Logger.getLogger("pep").info("Conecting to Group (" + groupAddress + ") with external ad
        String props="TCP(start_port="+groupPort;
        if  (externalAddress != null && !"".equals(externalAddress))
```

```
equals(Object arg0) boolean - Object
find(IPath path) URL - Plugin
find(IPath path, Map override) URL - Plugin
getBundle() Bundle - Plugin
getClass() Class<? extends Object> - Object
getDescriptor() IPluginDescriptor - Plugin
getDetectedOS() int - PEPActivator
getDialogSettings() IDialogSettings - AbstractUIPlugin
getImageRegistry() ImageRegistry - AbstractUIPlugin
Press 'Ctrl+Space' to show Template Proposals
```

equals

public boolean equals(Object obj)

Indicates whether some other object is "equal to" this one.

The equals method implements an equivalence relation on non-null object references.

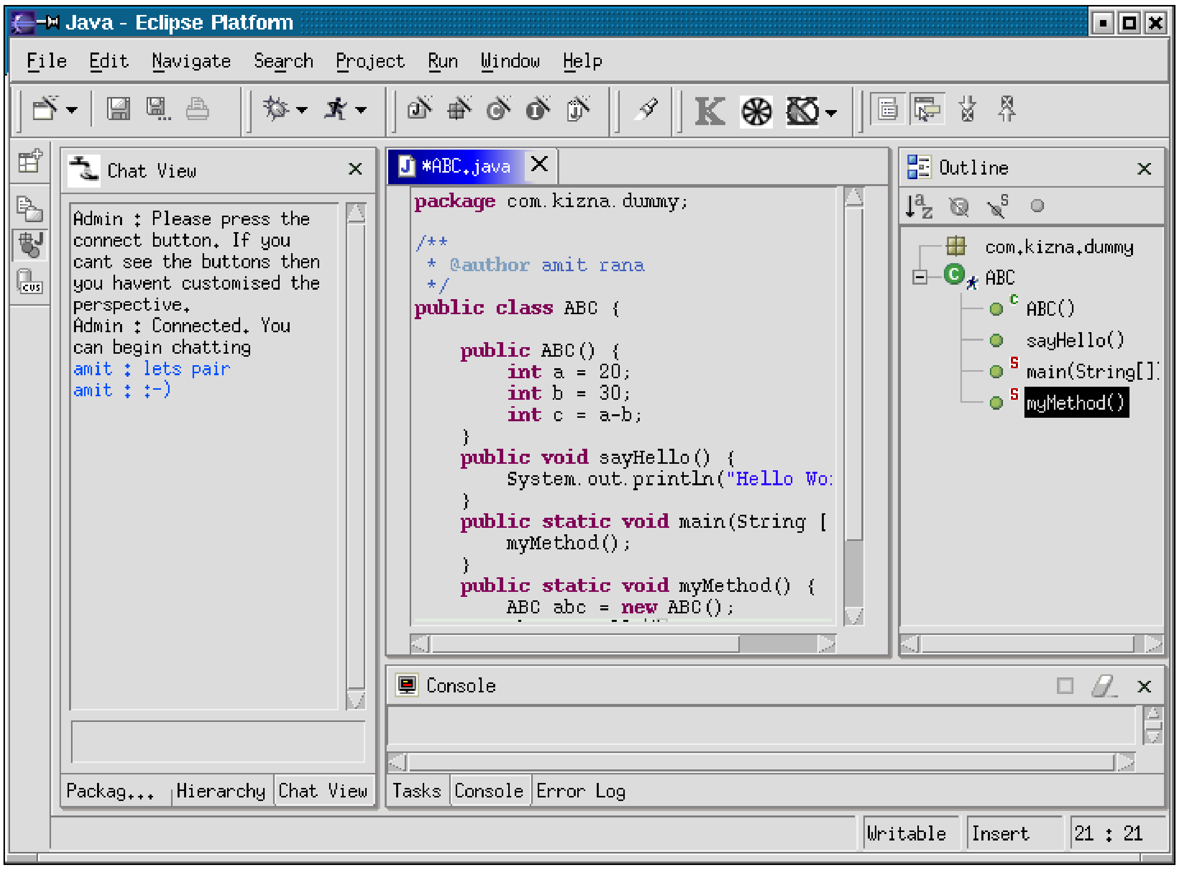
```
PEP Message : Connected. You can begin chatting
Local user : Vamos iniciar o trabalho
```

The static method setDriving(boolean) from the type PEPActivator should be accessed in a static way | Writable | Smart Insert | 43 : 35

Figure 24. Screenshot: PEP [PEP]

### 4.2.13 Sangam - A plug-in for collaboration

Sangam [Sangam] is an Eclipse plug-in specifically designed to support the Distributed Pair Programming practice. It is based on the open source Syncshare server. Sangam’s goal is to provide a robust platform for Distributed Pair Programming that retains usability over low-bandwidth connections.



```
Admin : Please press the
connect button. If you
cant see the buttons then
you havent customised the
perspective.
Admin : Connected. You
can begin chatting
amit : lets pair
amit : :-)
```

```
package com.kizna.dummy;

/**
 * @author amit rana
 */
public class ABC {

    public ABC() {
        int a = 20;
        int b = 30;
        int c = a-b;
    }
    public void sayHello() {
        System.out.println("Hello Wo
    }
    public static void main(String [
        myMethod();
    }
    public static void myMethod() {
        ABC abc = new ABC();
```

*Figure 25. Screenshot: Sangam [Sangam]*

To achieve this, Sangam uses an event-driven design for the communication between the Eclipse IDEs with the Sangam plug-in installed [Ho, Raha, et al., October 2004]. When the Driver changes the artefact under work in Eclipse, the plug-in intercepts the event and notifies the plug-in at the Navigator’s end to automatically perform the same task.

With Sangam, each developer is free to customize his screen as desired, since both sides are running a separate instance of Eclipse. A toolbar that is added to the Eclipse GUI when Sangam is installed is used by the pairing developers to connect to or disconnect to a Distributed Pair Programming session as well as for means of floor control to start and stop driving.

The implemented floor control mechanism is quite formal. A developer becomes the Driver by pressing the Start-Driving button. Then the Driver is in control of mouse and keyboard until he or she stops driving or exchanges roles with the Navigator by hitting the Stop-Driving button. In Sangam the Navigator is also able to move the mouse, but without any affect on the cursor on the Driver’s screen. The current version of the Sangam plug-in does not prevent the Navigator from typing, so the members of the Pair Programming Pair need to work out a social protocol so that one does not use the keyboard unless he is the Driver.

Sangam also provides means of textual chatting with the respective Distributed Pair Programming partner right in the Eclipse application window.

The main drawback to Sangam is that due to its event-driven design it is limited to Java development and can not be easily extended for other programming languages.

### 4.2.14 Saros

Saros [Saros] is an Eclipse plug-in designed for the support of the Distributed Pair Programming practice. It is developed by the Software-Engineering group of the Freie Universität Berlin as open source software. Saros is designed as a server based architecture. The communication between clients is done over Jabber servers.

Saros enables partners joining in a Distributed Pair Programming session to edit code in real-time. The display of each partner's view port and marking text selections of remote users in colours assigned to these users contribute to the creation of a sense of workspace awareness. It also provides the possibility to use instant messaging for means of communication during a Distributed Pair Programming session.

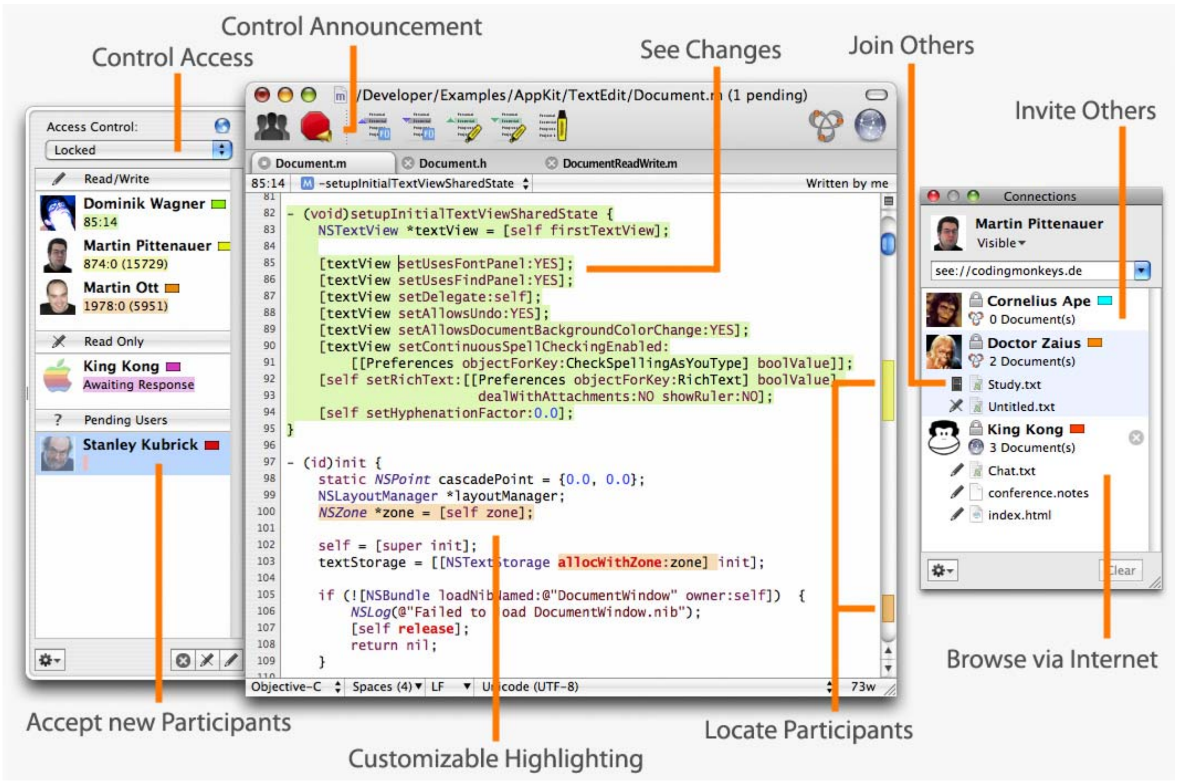
Saros implements a formal floor control mechanism following the role distribution between Driver and Navigator known from traditional Pair Programming. The role of Driver as defined in Saros is that of the user being in control of modification and the session. Only the Driver can edit files and only he can give this role to any other collaborating user. With Saros there can be one or multiple Navigators. The Navigator is allowed to see the view scope, open files and see selections of the Driver. There is a so called 'follow mode' for the Navigator that will keep his Eclipse to follow the scope of the current scope of the Driver.

### 4.2.15 SubEthaEdit

SubEthaEdit [SubEthaEdit] is a powerful text editor developed by Coding Monkeys for the Apple Mac OS X and originally designed for collaborative coding within distributed Teams and thus for the support of the Distributed Pair Programming practice. The sharing of documents for collaborative work with SubEthaEdit can either be done by using Apple's Bonjour technology or simply via the Internet. By using the Bonjour technology it is possible to see all announced documents available on the local network and to find users available for collaboration on the local network. As mentioned it is also possible to connect to other users using SubEthaEdit over the Internet and join existing collaborative sessions from anywhere.

In SubEthaEdit there is no mandatory emphasis of a floor control mechanism. In general anybody that has joined a collaborative session can type anywhere in the document at anytime. Nevertheless it is possible to mark certain documents as private and to have only selected users authorized to join such private documents. Additionally it is possible to grant not only full but also read-only access to the collaborating team members.

There are a lot of features available with SubEthaEdit to support workspace awareness. Amongst the most prominent are that other users' cursors and selections are replicated to all the collaborating users to make them aware of each other user's activities in the document. Further on in the document's scrollbar the view scope of each of the users in the session is displayed.



```
- (void)setupInitialTextViewSharedState {
    NSTextView *textView = [self firstTextView];

    [textView setUsesFontPanel:YES];
    [textView setUsesFindPanel:YES];
    [textView setDelegate:self];
    [textView setAllowsUndo:YES];
    [textView setAllowsDocumentBackgroundColorChange:YES];
    [textView setContinuousSpellCheckingEnabled:
        [[Preferences objectForKey:CheckSpellingAsYouType] boolValue]];
    [self setRichText:[[Preferences objectForKey:RichText] boolValue]
                    dealWithAttachments:NO showRuler:NO];
    [self setHyphenationFactor:0.0];
}

- (id)init {
    static NSPoint cascadePoint = {0.0, 0.0};
    NSLayoutManager *layoutManager;
    NSZone *zone = [self zone];

    self = [super init];
    textStorage = [[NSTextStorage allocWithZone:zone] init];

    if (![NSBundle loadNibNamed:@"DocumentWindow" owner:self])  {
        NSLog(@"Failed to load DocumentWindow.nib");
        [self release];
        return nil;
    }
```

*Figure 26. Screenshot: SubEthaEdit [SubEthaEdit]*

With respect to means of communication SubEthaEdit integrates with iChat and Mail to enable users to communicate via text.

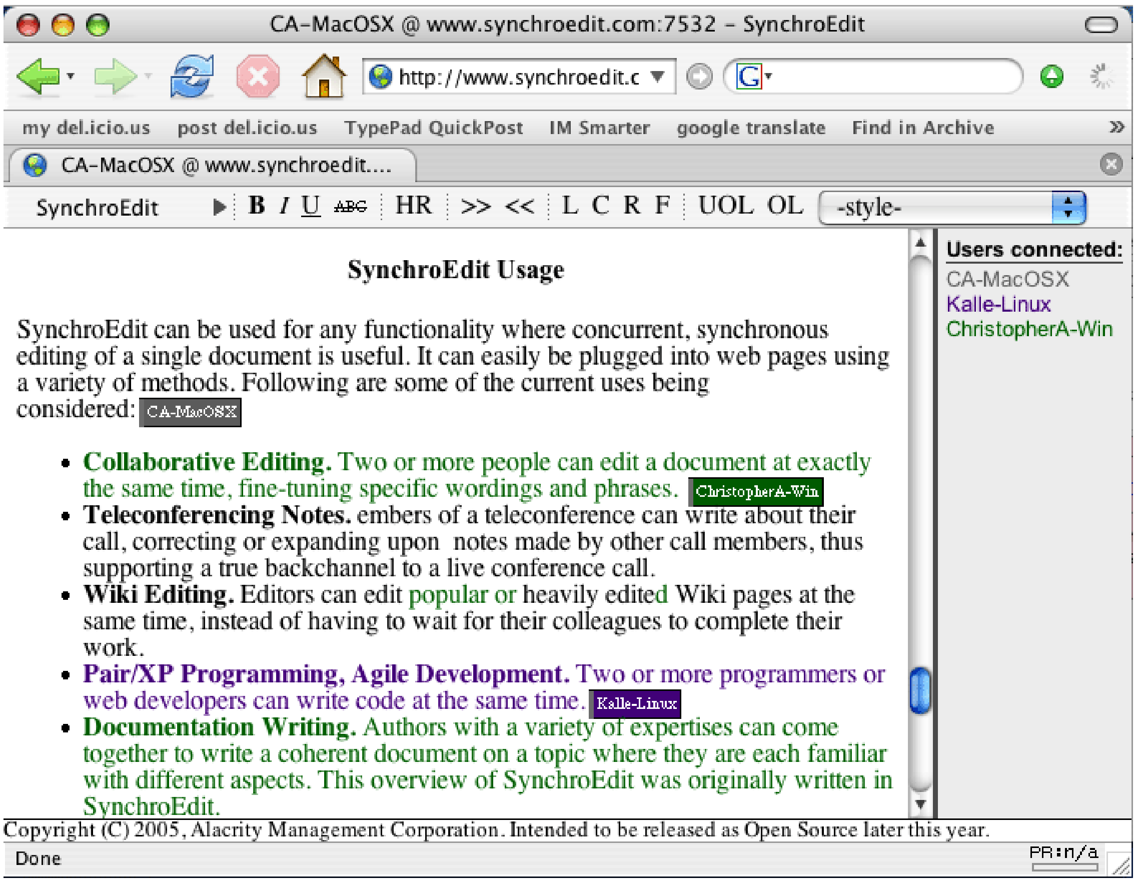
Being not only a collaborative editor but also providing support especially for coding activities, SubEthaEdit provides features that a user would rather be used to find within an IDE like bracket matching and selection, self-contained, extensible syntax modes supporting syntax highlighting and function popup menu as well as customizable syntax colours. This comes along with a fully fledged editor including all the features general expected.

## 4.2.16 SynchroEdit

As stated on SynchroEdits homepage [SynchroEdit], SynchroEdit is a browser-based simultaneous multi-user editor, a form of same-time, different-place groupware. It allows multiple users to edit a single web-based document at the same time, and it continuously synchronizes all changes so that users always have the same version.

Remarkable about SynchroEdit, although this might not be as important when using it for coding activities, is that its editor provides full what-you-see-is-what-you-get (WYSIWYG) support including dynamic display of bolds, italics, underlines, strikethroughs, indents and listing styles as entered by the author. Additionally SynchroEdit also supports a simple, text-only editor for more basic documents.

The sense of workspace awareness in multi-user environments is supported by clearly marking which changes where done by which user using a colour that is assigned to each user. Along this it is marked where each user’s cursor is located by using a coloured flag listing the user's name.



**SynchroEdit Usage**

SynchroEdit can be used for any functionality where concurrent, synchronous editing of a single document is useful. It can easily be plugged into web pages using a variety of methods. Following are some of the current uses being considered: CA-MacOSX

- **Collaborative Editing.** Two or more people can edit a document at exactly the same time, fine-tuning specific wordings and phrases. ChristopherA-Win
- **Teleconferencing Notes.** embers of a teleconference can write about their call, correcting or expanding upon notes made by other call members, thus supporting a true backchannel to a live conference call.
- **Wiki Editing.** Editors can edit popular or heavily edited Wiki pages at the same time, instead of having to wait for their colleagues to complete their work.
- **Pair/XP Programming, Agile Development.** Two or more programmers or web developers can write code at the same time. Kalle-Linux
- **Documentation Writing.** Authors with a variety of expertises can come together to write a coherent document on a topic where they are each familiar with different aspects. This overview of SynchroEdit was originally written in SynchroEdit.

Copyright (C) 2005, Alacrity Management Corporation. Intended to be released as Open Source later this year.

*Figure 27. Screenshot: SynchroEdit [SynchroEdit]*

SynchroEdit advertises itself as being useful for any functionality where concurrent, synchronous editing of a single document is useful [SynchroEdit], thus making it a good candidate for the support of the Distributed Pair Programming Practice.

### 4.2.17 TalkAndWrite plug-in for Skype

TalkAndWrite [TalkAndWrite] is a plug-in for Skype that enables its users to work on any document as though they are sitting side by side. Skype used along with its TalkAndWrite plug-in provides all functionality needed for a complete video conferencing system. As advertised on its homepage [TalkAndWrite] this program allows you to work together on a document or a white sheet of paper, making notes on it, handwrite, scribble, draw, erase, type or highlight text, all of this talking freely through Skype as if you were sitting side by side. This makes it a good candidate to be taken into account as tool support for the Distributed Pair Programming practice, although it is definitely not specifically designed for this purpose.

An asset for the TalkAndWrite plug-in to Skype with respect to Distributed Pair Programming is that it provides independent pointers to both users in a call that are visible to each other. In this way it is easily possible to point out to each other any part of the document like in a traditional Pair Programming session. Additionally text highlighting is supported.

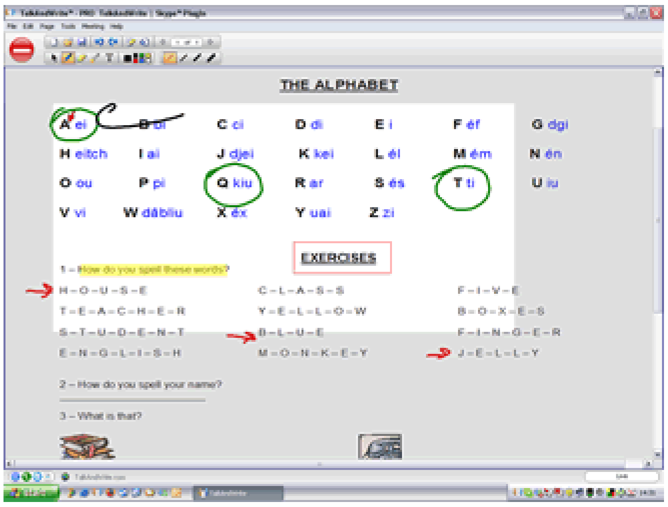


*Figure 28. Screenshot: TalkAndWrite plug-in for Skype [TalkAndWrite plug-in for Skype]*

A further contribution to the sense of workspace awareness is that TalkAndWrite visualizes which areas the partner can see on his screen by greying those areas which are not visible on the partner's screen.

### 4.2.18 Telescope

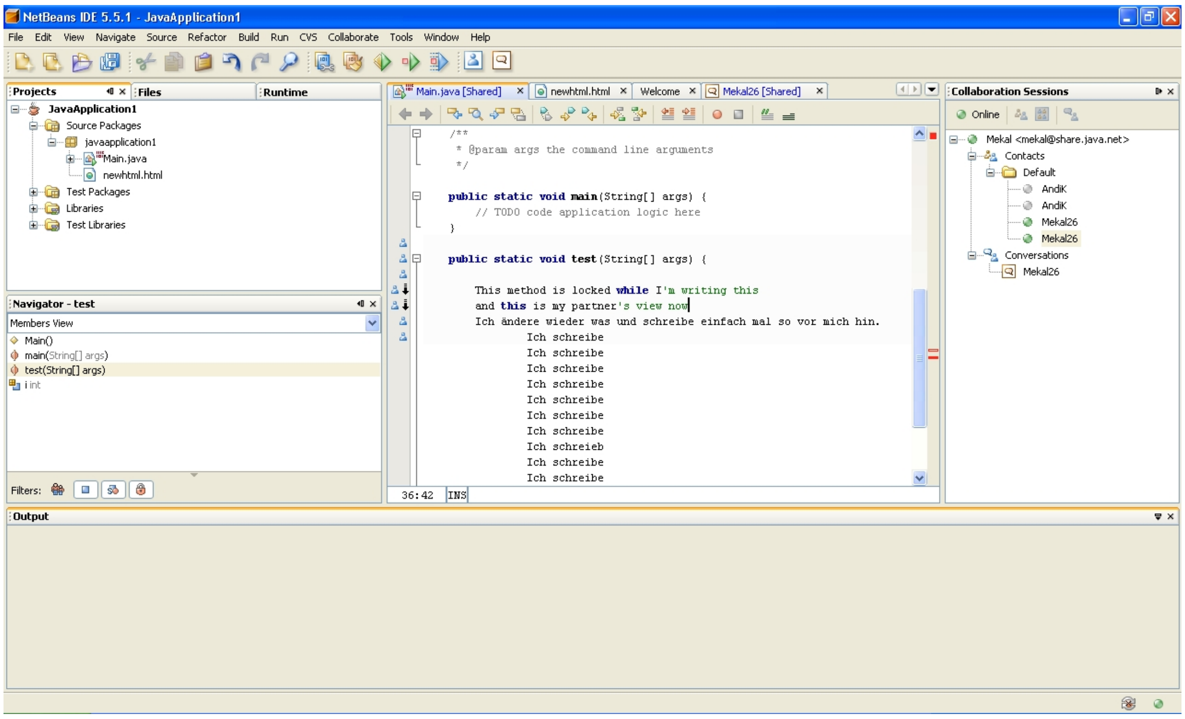
According to its homepage on Sourceforge [Telescope], the Telescope Communication and Collaboration Platform is Eclipse-based audio and video conference software for distributed software development teams. Therefore it might also be useful in support of the Distributed Pair Programming practice, but unfortunately all the documentation is only available in Spanish and could not be evaluated due to lack of knowledge of this language.

### 4.2.19 The NetBeans Collaboration Project

The NetBeans Collaboration Project [collab.NetBeans], also known as collab.NetBeans, is a plug-in to the NetBeans IDE. Its developer collaboration feature allows its users to connect to collaboration servers and by this to collaborate with other developers online wherever they are located.

It is possible to share projects and files in real time, allowing others in the conversation to make changes, which are replicated on the screens of the rest of the group. This feature makes collab.NetBeans a potential candidate for the support of the Distributed Pair Programming practice. Chatting capabilities with all appropriate formatting functionalities in place within collab.NetBeans provide means of textual conversation between the collaborating partners. It is possible to send messages in plain text, XML, HTML, or Java code format.

Some kind of a floor control is also implemented in collab.NetBeans. If one person is typing in a section of the shared document or code this section is locked for all other users joining in the collaborative work. Nevertheless there is no formal mechanism supporting the role distribution between Driver and Navigator characteristic for Pair Programming and although a section worked on by one user is locked to any other users, each of them is free to work on any other section (and lock it for him working there) while the first user is still typing.



```
    /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {
        // TODO code application logic here
    }

    public static void test(String[] args) {

        This method is locked while I'm writing this
        and this is my partner's view now
        Ich ändere wieder was und schreibe einfach mal so vor mich hin.
                Ich schreibe
                Ich schreibe
                Ich schreibe
                Ich schreibe
                Ich schreibe
                Ich schreibe
                Ich schreibe
                Ich schreieb
                Ich schreibe
                Ich schreibe
```

*Figure 29. Screenshot: The NetBeans Collaboration Project*

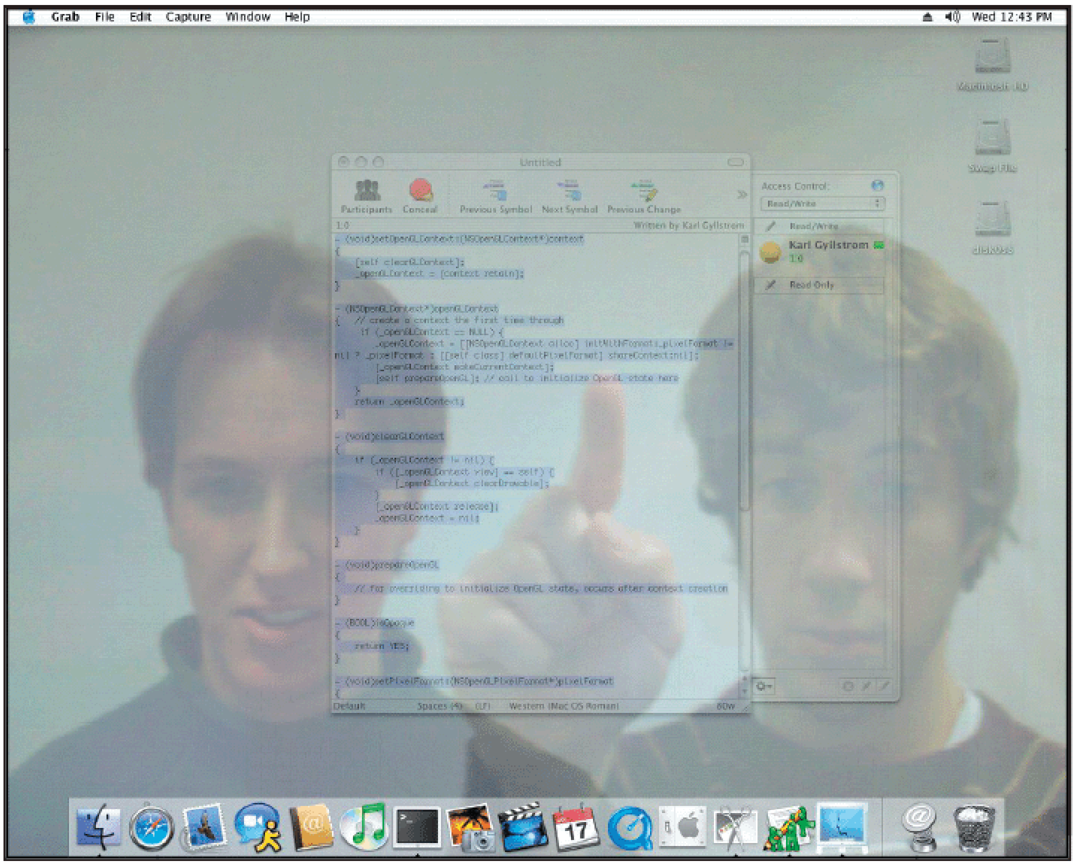
### 4.2.20 Transparent Video Facetop

The Transparent Video Facetop system developed by Stotts et al. [Stotts, Smith and Gyllstrom, 2004] takes quite a different approach to the support of distributed collaborative work including Pair Programming than most of the other tools found throughout the research.

The system needed is a Macintosh platform equipped with a webcam pointing at the user. A screen-sharing application is used to replicate the desktop used for the collaborative work to both users' workstations. The video stream generated by each of the Distributed Pair Programming pairs' webcams is transmitted to the other partner's workstation. Instead of displaying the video stream in a separate window, both of the streams along with the shared desktop are composed in a way that the video image appears as a transparent overly on the screen. In this way the members of the Distributed Pair Programming pair see their desktops overlaid with semi-transparent images of both of them as if they were sitting side by side.

By using the possibilities arising from this setup the system provides a very intuitive form of gesturing. A user wanting to show something to his partner simply needs to point on the screen and his partner can see where he is pointing, which is just the way Pair Programming works in collocated environments.

The most significant drawback of the Transparent Video Facetop system is that it requires a high bandwidth peer-to-peer connection between the collaborating partners to make the transmission of the video-stream possible.



```
- (void)setOpenGLContext:(NSOpenGLContext*)context
{
    [self clearGLContext];
    _openGLContext = [context retain];
}

- (NSOpenGLContext*)openGLContext
{   // create a context the first time through
     if (_openGLContext == NULL) {
        _openGLContext = [[NSOpenGLContext alloc] initWithFormat:_pixelFormat !=
nil ? _pixelFormat : [[self class] defaultPixelFormat] shareContext:nil];
        [_openGLContext makeCurrentContext];
        [self prepareOpenGL]; // call to initialize OpenGL state here
    }
    return _openGLContext;
}

- (void)clearGLContext
{
    if (_openGLContext != nil) {
        if ([_openGLContext view] == self) {
            [_openGLContext clearDrawable];
        }
        [_openGLContext release];
        _openGLContext = nil;
    }
}

- (void)prepareOpenGL
{
    // for overriding to initialize OpenGL state, occurs after context creation
}

- (BOOL)isOpaque
{
    return YES;
}

- (void)setPixelFormat:(NSOpenGLPixelFormat*)pixelFormat
{
```

*Figure 30. Screenshot: Transparent Video Facetop*
*[Stotts, Smith and Gyllstrom, 2004]*

### 4.2.21 TUKAN

The TUKAN environment developed by Schummer and Schummer [Schummer and Schummer, 2001] is actually developed for the support of distributed XP teams by integrating various key issues relevant in XP like awareness and communication into a programming environment. For the support of Distributed Pair Programming TUKAN includes a collaborative multi-user editor for code editing.

The editor included with TUKAN takes care that both partners view the same section of the file being edited. There are separate pointers for both of the partners within a session which are also replicated to the other partner's desktop enabling deictic gesturing in scope of the Pair Programming.

TUKAN does not provide a clear distinction between the roles of Driver and Navigator from the tool's side. Such a clear role distribution needs to be done socially between the pairing partners. As the users of TUKAN have separate insertion points it is possible for them to edit distinct parts of the artefact simultaneously.

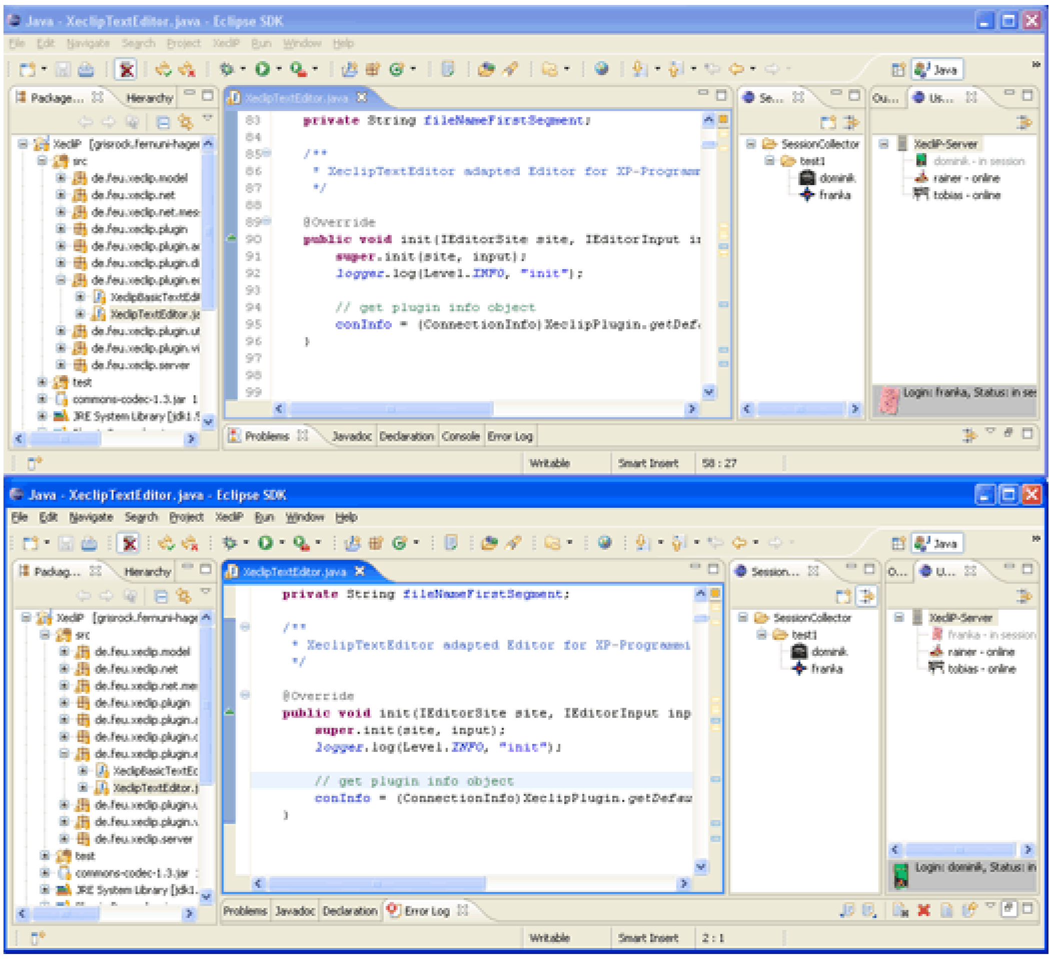
The collaboration within TUKAN is not restricted to the two partners needed for the Distributed Pair Programming practice. It is possible that further partners join a Pair Programming session.

Development of TUKAN was stopped some time ago. Therefore it is not available for current versions of operating systems and mentioned here for matters of completeness. Kind of a successor to TUKAN, although restricted to the

Distributed Pair Programming practice and not meant to support distributed XP in general, is XPairtise which will be discussed below.

### 4.2.22 XecliP

XecliP [XecliP] is a plug-in for the Eclipse IDE that supports Distributed Pair Programming of Java projects via Internet or within a local network. XecliP is based on classic client-server architecture meaning that all communication between clients is handled by a central server, which modifies, dispatches or generates the appropriate messages for the clients.


Java - XeclipTextEditor.java - Eclipse SDK
File Edit Navigate Search Project XecliP Run Window Help
XeclipTextEditor.java
SessionCollector
test1
dominik
franka
XecliP-Server
dominik - in session
rainer - online
tobias - online
Login: franka, Status: in ses
Problems Javadoc Declaration Console Error Log
Writable Smart Insert 58 : 27
Java - XeclipTextEditor.java - Eclipse SDK
franka - in session
Login: dominik, Status: in
Problems Javadoc Declaration Error Log
Writable Smart Insert 2 : 1

```
    private String fileNameFirstSegment;

    /**
     * XeclipTextEditor adapted Editor for XP-Programm
     */

    @Override
    public void init(IEditorSite site, IEditorInput i
        super.init(site, input);
        logger.log(Level.INFO, "init");

        // get plugin info object
        conInfo = (ConnectionInfo)XeclipPlugin.getDef
    }
```

```
    private String fileNameFirstSegment;

    /**
     * XeclipTextEditor adapted Editor for XP-Programmi
     */

    @Override
    public void init(IEditorSite site, IEditorInput inp
        super.init(site, input);
        logger.log(Level.INFO, "init");

        // get plugin info object
        conInfo = (ConnectionInfo)XeclipPlugin.getDefau
    }
```

*Figure 31. Screenshot: XecliP [XecliP]*
*On the top the view of the Navigator and at the bottom the view of the Driver.*

Users of XecliP need to enter a user profile which is stored on the XecliP server. Taking advantage of this need, XecliP provides the possibility to search for a Pair Programming partner within the user database generated in this way according to certain criteria like their proficiency in various programming techniques.

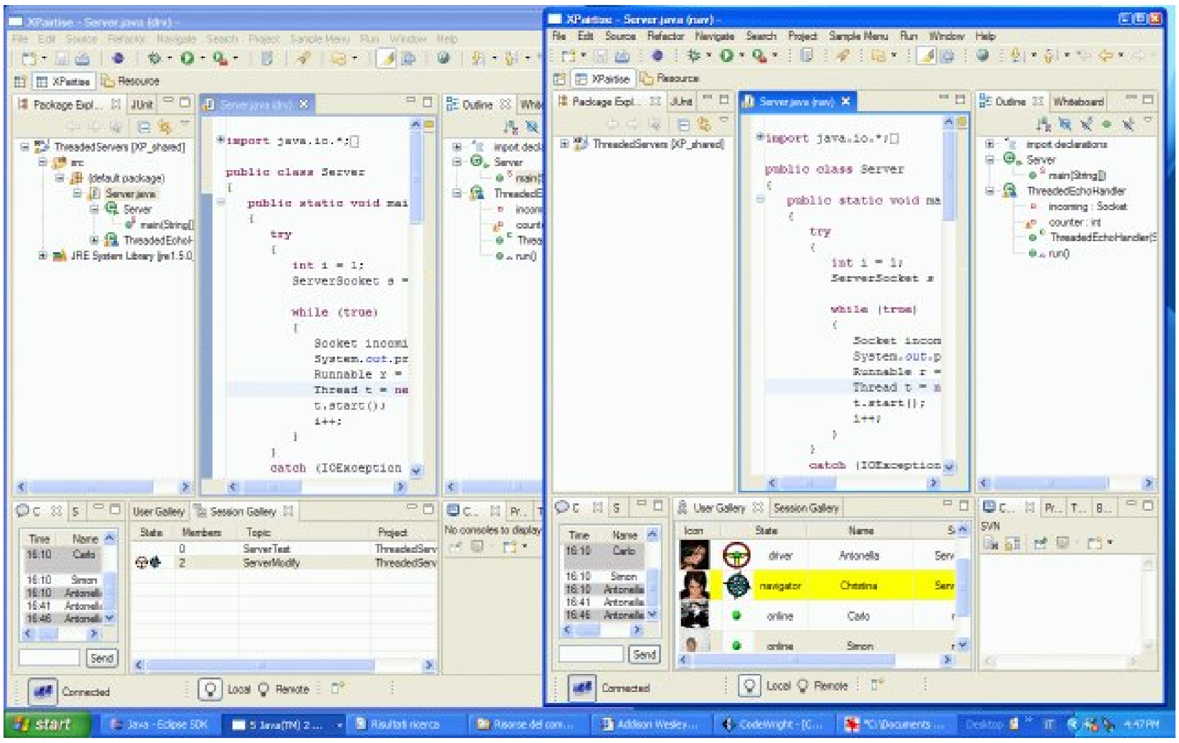
For the XecliP plug-in the XecliP Java editor is the default editor to edit Java documents. As XecliP only works with Java development any use of another editor or use in context of another programming language will not work. The XecliP Java editor features common editor functionality like syntax highlighting, content and code assistance, code formatting, import assistance and integrated debugging features.

XecliP supports a formal concept of floor control. The person who initiates a session is always the Driver in the beginning. A later role change needs to be requested by either Driver or Navigator and accepted by the partner. In XecliP there is a third role implemented called visitor, but the use case of this role is unclear from the available documentation.

As most of the Eclipse plug-ins for Distributed Pair Programming, XecliP also provides a textual chat. A special feature about XecliP is that it is possible to record a session and to playback a recorded session.

### 4.2.23 XPairtise - Pair Programming for Eclipse

XPairtise [XPairtise] is another Eclipse plug-in that provides a platform for Distributed Pair Programming. According to information from Till Schümmer XPairtise can be seen as kind of a successor to the TUKAN system with respect to the support of Distributed Pair Programming. As is the case with most other of the mentioned Eclipse plug-ins the basic architecture is a typical client-server approach. A single server exchanges data with several clients and no direct client-to-client communication is done.



```
import java.io.*;

public class Server
{
    public static void mai
    {
        try
        {
            int i = 1;
            ServerSocket s =

            while (true)
            {
                Socket incomi
                System.out.pr
                Runnable r =
                Thread t = ne
                t.start();
                i++;
            }
        }
        catch (IOException
```

*Figure 32. Screenshot: XPairtise [XPairtise]*

The floor control is implemented in XPairtise as formal floor control. Thus XPairtise features the traditional Pair Programming roles of Driver and Navigator. As a third role there is the role of a spectator who is only allowed to drop in and view the session but is not part of the role change mechanism. To initiate a role change, either Driver or Navigator has to request it and the respective other partner has to accept the request.

XPairtise provides the means of a textual chat as do most of the Eclipse plug-ins for Distributed Pair Programming. Unlike the other Eclipse plug-ins for Distributed Pair Programming XPairtise also provides a whiteboard for graphical communication.

Highlighting as a method of pointing out some sections of the code is supported by XPairtise. When the Driver highlights some text in his editor, the same text is highlighted in the Navigator's copy. If the Navigator highlights text in his copy, the same portion of text changes background colour in the Driver's editor view. In this way the Navigator can give contextual meaning to his chat or voice messages.

In additionally to the already mentioned features XPairtise provides shared program and test execution enabling the Distributed Pair Programming team to perform joint tests easily.

## 4.3 Candidate Tools Summary

The descriptions given above of the tools found during the research in scope of this diploma thesis hint on the broad range of approaches taken to provide tool support for non-collocated collaborative work in general and the practice of Distributed Pair Programming in particular. To get a better overview of these tools they can be categorized in screen-sharing applications and collaboration-aware applications. Within these two main categories a further sub-categorization is used to distinguish between tools that, though belonging to one main category, show major differences.

All of the described tools are seen as candidate tools for the support of Distributed Pair Programming according to the information that was found about each tool. Nevertheless this information does not provide a good hint on whether all of these tools are really usable for the support of the Distributed Pair Programming practice or whether any of the tools are better suited than others with respect to their usability in a Distributed Pair Programming environment. Thus a comparable evaluation of these tools against requirements that are desirable to be met by tools used for the application of the Distributed Pair Programming practice needs to be done. This is discussed in the following chapters.

# 5 A generic Evaluation-Framework for Tools supporting the Distributed Pair Programming Practice

As stated above, previous scientific work already brought up a number of requirements for tools supporting or specifically designed for the practice of Distributed Pair Programming. Within this chapter these requirements are consolidated and extended to form a complemented list of such requirements or properties to be desired from tools that are to be used in the practice of Distributed Pair Programming.

These required properties will then be used to create an evaluation framework similar to the abstract model to select software development tools created by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005]. In this way a generic evaluation framework for tools supporting the practice of Distributed Pair Programming is set up.

## 5.1 Properties of Tools supporting the Distributed Pair Programming Practice

The geographic distance between the collaborating members of a Distributed Pair Programming team leads to various issues with respect to the application of the Pair Programming practice. These issues can be broken down to a number of properties that need to be present to some extent in tools that are supporting or even specifically designed for the practice of Distributed Pair Programming to overcome the drawbacks resulting from the non-collocated environment.

The properties needed to be available with tools supporting or specifically designed for the practice of Distributed Pair Programming can be broken down into a number of categories. The following categorization is found to be reasonable.

- The basic properties category comprises such properties that are crucial to enable the work in a Distributed Pair Programming team. These crucial properties are the support of screen-sharing for screen-sharing applications, the sense of collaboration awareness for collaboration-aware applications and the sense of workspace awareness for both application types.
- Another category includes all properties with respect to the support of gesturing for the Driver as well as the Navigator and the degree to which deictic referencing is supported by the tooling.
- Floor control support is the category containing all properties that deal with the extent to which the role concept of Driver and Navigator and the exchange of these roles between the collaborating partners of a Distributed Pair Programming team is implemented within the tool and in which way this support is implemented.
- As was already discussed the practice of Pair Programming is an extremely communicative activity. This certainly needs to be supported by software tools that support Distributed Pair Programming. The category of communication possibilities summarizes all the properties dealing with the various communication channels that might be needed in scope of appliance of the Distributed Pair Programming practice.

- In the case of Distributed Pair Programming the collaborating partners might use different operating system platforms on their respective machines. Thus platform independence is a further important category for properties of tools supporting or specifically designed for the practice of Distributed Pair Programming.
- As with all software products the category of usability properties is an important one for tools supporting or specifically designed for the practice of Distributed Pair Programming as well.
- A further category is that of tool immanent properties. These properties are not so much important for the practice of Distributed Pair Programming but rather include such properties of the tool that described the tool itself, for example whether it is off-the-shelf, special purpose, a viewer and so on.
- The final category of Documentation summarizes the quality of the installation as well as user manual of the tool.

Not all of the properties are mandatory to be fulfilled by a tool to make it usable in the appliance of Distributed Pair Programming practice. Quite a number of them are rather optional and an additional asset to the usability of the tooling if available.

### 5.1.1 Basic Properties

The category of Basic Properties deals with such properties that are crucial to remove the main obstacles that arise from the non-collocated environment when applying the practice of Distributed Pair Programming. In this way the properties subsumed under this category on one hand have to simulate the collocated situation of the two Pair Programming partners sitting in front of a shared screen; on the other hand they need to fulfil the emulation of a similar sense of workspace awareness as present in the collocated environment.

#### 5.1.1.1 Support of Screen Sharing

The Support of Screen Sharing is the most basic property of the tools subsumed under the category of screen-sharing tools and therefore only valid for such tools. The basic idea of screen-sharing is that one screen is shared by Driver and Navigator as is the case in the collocated case of Pair Programming. Both members of the Pair Programming pair see the same screen and therefore interact on the same desktop. To achieve this either one user’s workstation works as a server replicating its desktop onto the other user’s workstation which therefore acts as a server, or a dedicated server machine is used which replicates its desktop on both users’ workstations to provide the common environment to work in. In both ways the basic idea of screen-sharing is fulfilled and both users have the same view on their common environment.

The Support of Screen Sharing property provides information how good the tool supports the idea behind screen-sharing and how far any drawbacks arising from this concept are overcome by the tool.

Large amounts of data can lead to problems with the screen actuality and therefore the usability of screen-sharing applications. Therefore all screen-sharing applications use compression algorithms to reduce the amount of data needed to be transmitted between server and client.

Another potential problem with screen-sharing applications is that the actual view given on the replicated screen might differ from that on the server that is replicated due to lower screen resolution settings of the clients. This can be overcome by setting the same screen resolution or by adapting the replicated screen within the screen-sharing application to fit the possibly varying screen resolutions of the clients.

#### 5.1.1.2 Support of Collaboration Awareness

Similar to the Support of Screen Sharing property for screen-sharing tools, the Support of Collaboration Awareness is the core property of such tools that fit into the collaboration-aware tools category and therefore is only valid for such tools.

Collaboration-aware tools usually do not support screen-sharing. They rather support different views based on the roles of the individual users or support a common view on the shared artefact without actually sharing the screen. Often collaboration-aware tools come with a richer environment that more effectively supports collaboration than it is the case with screen-sharing tools. For example all IDEs specifically designed or enhanced for the support of the Distributed Pair Programming practice fall into this tool category.

The Support of Collaboration Awareness property covers to which extent the tool supports the sense of collaboration awareness meaning how easy it is with the help of the tool to collaborate in scope of the appliance of the Distributed Pair Programming practice.

#### 5.1.1.3 Support of Workspace Awareness

As already discussed the term of workspace awareness is a term commonly used in the literature discussing distributed software development and is defined by Gutwin and Greenberg [Gutwin and Greenberg, September 1999] as the “up-to-the-moment understanding of another person’s interaction with the shared space”.

In the case of collocated Pair Programming the developers are sitting next to each other in front of a shared desktop. In this way it is quite clear that the partners are aware of each others presence, focus and activities. In cases where multiple persons are interacting with a shared artefact it is essential for effective collaboration that each user knows who else is participating in the collaborative work, where the collaborating partners put their focus within the shared artefact that they are working on and what they are actually doing at all times during a collaborative session. This is especially valid if the users are not restricted by the tooling to view the same sections of the shared artefact all the time.

Although Hanks [Hanks, December 2005] states that issues surrounding workspace awareness are not as important in screen-sharing application due to the fact that every user is presented with the same view, the Support of Workspace Awareness property is nevertheless valid for screen-sharing tools as well as collaboration-aware tools and rates how good the feeling of workspace awareness is transported by the tool and how much contextual information about the shared workspace is provided.

### 5.1.2 Support of Gesturing

As already discussed, gesturing and deictic reference are important elements of collaborative work. It is of utter importance that gestures or deictic references done by one of the partners joining in the Distributed Pair Programming practice are visible to the other as well.

This is not so much a problem for the partner acting as Driver, as he is in control of all the input devices like mouse and keyboard. For him it is easy to hint onto specific sections of the shared artefact by simply using the input devices under his control to do the same.

As the Navigator has no control of the actual devices used for the normal input this is an issue for him. He needs a possibility to hint the Driver onto specific sections of the artefact they collaboratively work on. This can be achieved by the possibility to mark specific sections of the artefact in a way shown to the Driver in real time by using some highlighting mechanism.

An even better way to support this need is the availability of a second pointer that is controlled by the Navigator as this is more similar to the collocated case of Pair Programming where the Navigator would use his finger to point out something on the screen to the Driver. In this second case it is clear that the Driver’s pointer and its movements need to be replicated to the Navigator and vice versa. As it might be the cause for confusion if both pointers are displayed by the same icon their shape should make it easy to distinct between the Driver’s and the Navigator’s cursor using some semantic adaptation. A further drawback to the effective use of replicated pointing devices for Driver and Navigator and their movements might be due to network latency which could result in delays of the transmission of the cursors’ positions and thus impossibility to follow the deictic references given by the partner.

#### 5.1.2.1 Availability of a Second Pointer for the Navigator

The Availability of a Second Pointer for the Navigator property deals with the availability of a second pointer that can be controlled by the Navigator with his mouse and is replicated to the Drivers desktop.

A minimal grade of fulfilment of this property is that at least a single pointer that is available to Driver as well as Navigator is replicated on both partners’ desktops and that it can be used by both of them for issues of deictic referencing. A complete fulfilment of this property is only available if there are separate pointers for Driver and Navigator and both pointers are replicated to each partners’ shared workspace.

#### 5.1.2.2 Availability of Semantic Adaptations of Pointers with Respect to Collaboration

In single user environments it is often the case that the shape of the cursor is changed based on the user’s activity carried out or possible in the location where the pointer is located. In tools supporting collaboration of remote users, semantic adaptations have a somewhat different meaning than this. Such adaptations might for example be that each pointer has its user’s name attached to it or the shape of the Navigators pointer could be changed for means of low level communication like for example a gesturing pointer for the Navigator that only appears when the Navigator activates a dedicated gesturing mode.

A basic grade of availability of this property is that there is at least so much semantic adaptation of the users’ pointers available that it is clear which pointer belongs to which user respectively role.

#### 5.1.2.3 Suppression of Telepointer Jitter

Network congestion, delay and data loss can lead to jitter in updating the remote pointers on each of the users' screens making it difficult to accurately follow gesturing movements. The Suppression of Telepointer Jitter property defines how good a tool is in suppressing this jitter effect, or whether it provides any mechanisms like a tail that follows moved pointers to overcome the negative effects of telepointer jitter.

#### 5.1.2.4 Support of Highlighting with Respect to Collaboration

The Support of Highlighting with Respect to Collaboration property can either be fulfilled by a tool as a basic method of providing a minimal possibility to hint the collaborating partner on specific sections of the shared artefact or along with the availability of a second pointer for the Navigator as a further add-on to increase usability and convenience for gesturing and deictic referencing. The meaning of this property is that the tool provides some mechanism so that highlighting of sections of the artefact worked on and the replication of this highlighting on the partner's screen or shared workspace is supported for one or both users.

### 5.1.3 Floor Control Support

As the role concept of Pair Programming defining the roles of Driver and Navigator along with responsibilities to be taken care of by each of these roles is an important issue of the Pair Programming practice, there is also the need for tools supporting or specifically designed for the Distributed Pair Programming practice to provide support for this role concept as well as for the role changes between the collaborating partners. An important issue for the support of role changes between the collaborating partners is that the process of exchanging roles does not disturb the actual Pair Programming process.

There is a vast range of ways how the support of floor control can be implemented within a tool usable for the practice of Distributed Pair Programming ranging from a very formal approach implementing a request and commit protocol and clearly defined roles of Driver and Navigator via the situation where the tool support in an informal approach a minimum of floor control by locking the sections of the artefact one of the users is working on but does not provide a tool-immanent support for role changes to the situation where no floor control support at all is given by the tool and the whole floor control concept is to be socially negotiated and agreed between the collaborating partners.

#### 5.1.3.1 Support of Floor Control

As discussed floor control allows one user of a shared application to take charge of it so that only the user in charge can provide input to the application, while the other user can view the application and see its output as well as the input of the user in charge. The Support of Floor Control property states whether there is any floor control support implemented within the tool at all and if yes how good the support of floor control provided by the tool is.

A minimum fulfilment of this property would be to have at least the section of the artefact which one of the users makes input in locked to any input by the other user. A full fulfilment of this property would be if there is a formal role distribution between Driver and Navigator along with a mechanism for role exchanges implemented by the tool.

#### 5.1.3.2 Support of Role Changes

In the Pair Programming practice Driver and Navigator exchange roles smoothly. Therefore there is the need to have these role changes supported by a tool used in scope of the Distributed Pair Programming practice. The Support of Role Changes property specifies whether there is a mechanism for role changes between Driver and Navigator implemented within the tool and how good it supports the role changes.

#### 5.1.3.3 Support of Informal Role Changes

The role changes in traditional collocated Pair Programming usually happen quite informally and smoothly, meaning that there is no clear cut when Driver and Navigator exchange the roles. This is often defined as social floor control. This property reflects how far this is supported by the tool.

#### 5.1.3.4 Support of Formal Role Changes

Formal role changes are a form of implementation of the role exchange between Driver and Navigator where this role exchange is done via a request and commit mechanism, or by mediation by the application itself or a master-user. This is easier to implement than a support of informal role changes as according to the action taken in this case it is quite clear who is in the role of Driver and who in the role of Navigator.

For users lacking experience in the practice of Pair Programming it might be easier to perform Pair Programming if the role changes happen following a clearly defined formal protocol. This property reflects how far this is supported by the tool.

#### 5.1.3.5 Easiness of Role Changes

As discussed the exchanging of the roles of Driver and Navigator in the Pair Programming practice should not interrupt the flow of the collaborative work. As Distributed Pair Programming is the transformation of the Pair Programming practice into an environment where the only difference to the traditional approach is that the partners are non-collocated the same is valid for Distributed Pair Programming. The property of Easiness of Role Changes describes how easy it is to exchange the roles with the help of the tool used.

#### 5.1.3.6 Support of the Information "Who is in which role"

An important information in the appliance of the Distributed Pair Programming practice also contributing to the sense of collaboration and workspace awareness is which of the partners holds the role of Driver and which the role of Navigator. In the collocated environment this is quite obvious to the partners sitting next to each other. The same is not so clear in a distributed environment. Thus there is the need for the tool to provide this information to its users.

This information can for example be provided by semantic adaptations of a possibly available second pointer for the Navigator or be showing the role the local user impersonates at any given time in the GUI. This property defines how far the tool environment reflects this information.

#### 5.1.3.7 Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role

For some utilizations of the practice of Distributed Pair Programming it might be important to get the information of "Who is how long in which role" along with statistical information of time spent in each role out of the tooling used. This is especially valid for some scientific approaches. Such information might also be usable in detecting certain grievances in the application of the Distributed Pair Programming practice. For example it could be found out whether both partners impersonated the roles of Driver and Navigator to a similar extent or whether one partner tended to hold the Driver’s role and the other to keep the role of Navigator. As this is contradicting with the Pair Programming approach where both partners are meant to be emancipated this might lead to reducing the benefits of Pair Programming. This property reflects whether such information is available from the tool and how detailed the information available is.

### 5.1.4 Communication Possibilities

Pair Programming is a highly communicative activity. The unrestrained communication between Driver and Navigator during a Pair Programming session is of utmost importance. The same is obviously valid and possibly even more important due to the non-collocated environment in Distributed Pair Programming.

The communication between Driver and Navigator can be supported by the availability of the following ways of communication within a tool used for the practice of Distributed Pair Programming:

- Speech
- Textual or graphical communication (Chat and White-Board)
- Video
- Additionally the support of pair- or partner finding prior to the actual programming session can be supported by the tooling and is also kind of a communication possibility.

The listed communication channels do not rule out one another. They are rather to be used sequentially or even in parallel.

For sake of completeness it has to be stated here that deictic referencing and the usage of pointers and highlighting mechanisms is also a way of communication between non-collocated collaborating partners but due to their importance in the appliance of the Distributed Pair Programming practice they are subsumed under an own property category.

#### 5.1.4.1 Support of a Speech Channel

All cooperative activities rely heavily on the possibility to speak with each other. The same is valid for computer supported cooperative work like Distributed Pair Programming. Therefore the support of communication by speech is an important property and asset to a tool that is to be used in scope of the appliance of the Distributed Pair Programming practice, although if not available it can be quite easily substituted by the use of nowadays available technology like Internet telephony for example via Skype [Skype] or teleconferences.

Previous work showed that in Distributed Pair Programming environments not supporting a speech channel but only a textual chat for the communication between the collaborating partners they tended to reduce their communication with each other and started to act more like solo-programmers after some time [Hanks, December 2005].

The availability of a speech channel for communication is not only important during the actual Pair Programming session but also for any discussions prior to starting the session or for the discussion of any problems. Communication via speech has the advantage that information can be transmitted faster by voice than by typed text.

#### 5.1.4.2 Support of textual Chat

The support of a textual chat is quite important as technical data is often rather suitable to be communicated via text. For example an Internet URL or piece of code is less prone to communication errors when transmitted using text and the transmitted data can be used directly by using copy-and-paste mechanisms. If similar information is transmitted via voice this will frequently lead to mistakes.

From the statements above it is quite clear that the availability of means for textual chat is quite important in the practice of Distributed Pair Programming as well. The easiest thing for the practitioners of Distributed Pair Programming is to have the same implemented right within the tool they use to enable them to perform the Distributed Pair Programming practice, making this an important property for such tools. If not supported by the tool used this channel of communication can be established by the use of separate tools like ICQ [ICQ], Jabber [Jabber], Skype [Skype] or any instant messaging software.

Often textual chats provide a history function as well as the possibility to store its content. This makes it easy to reuse prior information already entered in the chat.

#### 5.1.4.3 Support of a White-Board

The availability of a white-board within a tool used for the appliance of the Distributed Pair Programming practice is a further asset to such a tool as there is some information like drawings, flow-charts or diagrams like database-designs that can not be communicated easily via voice nor via textual chat.

Like stated for textual chats white-board implementations often provide the possibility to store and restore its content enabling the reusability of information transmitted via white-board.

#### 5.1.4.4 Support of a Video Channel

Although found in prior work to be of minor importance for the appliance of the Distributed Pair Programming practice [Hanks, December 2005], there are still some tools usable or specifically designed for the practice of Distributed Pair Programming that provide a video channel for communication or even focus on the use of video to support Distributed Pair Programming.

On the other hand it needs to be mentioned that the support of a video channel can be quite useful for meetings prior or after a Distributed Pair Programming session. If the developers that are to be joining each other in a Pair Programming team do not know each other an introductory meeting via video would be quite an asset as it is always easier to work with someone you have met in person.

If there is no video channel available with the tool but the need of such a means of communication arises it can be quite easily substituted by the use of other tools like for example Skype [Skype].

#### 5.1.4.5 Pair/Partner Finding Support

An additional asset rather than a required property of a tool usable in scope of the Distributed Pair Programming practice is the support of pair- or partner finding by the tooling. Pair- or partner finding means that the tool supports the search for potential partners to collaborate with during a Pair Programming session based on their current availability or even their proficiency in various programming areas or their experience. There are some tools like Moomba for example that support such a feature [Reeves and Zhu, 2004].

Pair- or partner finding support will be of minor importance in professional commercial software development in large companies as in such environments the available partners for Pair Programming will be quite limited due to organizational and project-immanent issues. Nevertheless it can be an important property of a tool used in open-source projects to find a potential collaboration partner.

### 5.1.5 Platform Independence

As the collaboration in Distributed Pair Programming teams might not be restricted to users using the same operating system, platform independence might be an issue for such tools usable or specifically designed for the practice of Distributed Pair Programming.

#### 5.1.5.1 Support of Microsoft Windows

This property states whether Microsoft Windows is supported by the tool.

#### 5.1.5.2 Support of Unix/Linux

This property states whether UNIX or Linux or both are supported by the tool.

#### 5.1.5.3 Support of MacOS

This property states whether MacOS is supported by the tool.

### 5.1.6 Usability

The property category of Usability contains such properties that affect how easily and intuitively the tool can be used. This is actually only partly specific for such tools that are usable or specifically designed for usage in the appliance of the Distributed Pair Programming practice. Nevertheless if is of significant importance for the acceptance of such tools and therefore for the appliance of the Distributed Pair Programming practice itself that the tools for this issue follow the general requirements on usability of software products.

With respect to a tool’s usability the following criteria are important:

- The tool’s effectiveness.
- The tool’s efficiency.
- The user’s satisfaction when using the tool.

A tool is effective if the targeted goal can be reached by the use of the tool. The efficiency of a tool is defined as reaching the targeted goal with as little effort as possible. The user’s satisfaction with a tool describes the subjective feeling of the fun the user has in applying the tool. This also includes that the usage of the tool is intuitive in a way that the user is not hampered by some implementations of functionalities of the tool [ISO9241].

#### 5.1.6.1 Intuitiveness of tool usage

The intuitiveness of the usage of a tool defines how easy and efficient it is by using the tool to fulfil the assignment at hand. An intuitive tool makes it easier to use the tool’s features as it is not necessary to search a long time for specific functionalities. This also increases the user’s satisfaction with the tooling. No user will use a tool that might come with a wide range of features but where it is not intuitive for the user how to use its features.

#### 5.1.6.2 Easiness of Installation and configuration

Nowadays the user of a software application expects it to be easy to install and configure using a wizard or something like that. The same is certainly valid for tools usable or specifically designed for the support of Distributed Pair Programming.

If there are already difficulties or problems during installation of a tool the user will not have an unbiased approach to using the tool as he will expect further troubles with the tool and therefore the user’s satisfaction will be damped. This might also affect the effectiveness and efficiency of a tool.

#### 5.1.6.3 Easiness of Start up and establishing a session

This property deals with the easiness of starting the tool to be used in scope of the Distributed Pair Programming practice and setting up a Pair Programming session with the tooling. Depending on how easy this is this property will effect the user’s satisfaction and the user acceptance of the tool.

#### 5.1.6.4 Stability

This property deals with the stability of the tool during a session. If a Pair Programming session is frequently interrupted due to break ups of the session or the connection to the Pair Programming partner this will not be acceptable for the users. Further on this property signals whether a session broken on one side is kept open on the other partner’s side and can be easily rejoined after the connection has been restored or if a session broken on one side is terminated at once throwing all participants out of the session and possibly even destroying all the work done during the session that has not yet been stored.

#### 5.1.6.5 Screen Actuality - Realtime-Probability

The main requirements of Pair Programming are that its participants have the same view on the workspace and can easily communicate with each other. The replication of screen content might lead to the situation where a large amount of data is to be transmitted via the network. This can lead to problems with the screen actuality and thus can lead to some confusion between the Pair Programming partners. This property deals with how good the realtime probability of the tool used is.

#### 5.1.6.6 Easiness of Collaboration

The property Easiness of Collaboration validates how easy it is for the Distributed Pair Programming partners to follow each other's activities and how easy it is to collaborate by using the tool.

#### 5.1.6.7 Intuitiveness of User Interface

Similar to Intuitiveness of Tool Usage the Intuitiveness of User Interface property discusses how intuitive the user interface of the tool is. A tool that follows common standards of user interface design is more easily to use due the intuitiveness of its user interface.

#### 5.1.6.8 Need for other Tools

As mentioned in the Communication Possibilities property category there are some properties that are required to enable an effective and efficient appliance of the Distributed Pair Programming practice that can be substituted by the use of other tools like the provisioning of various communication channels. As well it is quite obvious that when using a screen-sharing application for support of Distributed Pair Programming an additional editor or IDE will be needed to actually write the code. This property quantifies how many additional tools are needed and how readily available needed tools are.

#### 5.1.6.9 Support of Session Management

Support of session management means the possibility to store whole Pair Programming sessions along with documents that were opened, workspace views, chat transcripts and so on. This is an optional property of a tool usable or specifically designed for Distributed Pair Programming but is certainly quite an asset if available with the tooling as it will speed up the re-establishing of a session significantly.

#### 5.1.6.10 Support of Syntaxhighlighting

The Support of Syntaxhighlighting property is only valid for collaboration-aware tools as screen-sharing tools by definition do not feature an own editor which would be needed to support syntax highlighting. If the collaborative editor or IDE used for Distributed Pair Programming only supports a standard editor without syntax highlighting there is the danger that a number of errors will not be recognized by the Driver and Navigator thus reducing the quality of the artefact produced.

### 5.1.7 Tool Properties

The Tool Properties category contains such properties that are actually not properties required for the successful application of the Distributed Pair Programming practice but such properties that give information about what kind of tool the tool in scope is.

#### 5.1.7.1 Off-the-shelf - Commercial

This property informs whether the tool is a commercial off-the-shelf application.

#### 5.1.7.2 Off-the-shelf - Open Source

This property informs whether the tool is an open-source off-the-shelf application.

#### 5.1.7.3 Special Purpose

This property informs whether the tool is specifically designed for the purpose of Distributed Pair Programming or at least distributed development.

#### 5.1.7.4 Integrated Development Environment

This property informs whether the tool is an IDE or a plug-in or extension of an IDE.

#### 5.1.7.5 Viewer

This property informs whether the tool is a simple remote desktop viewing application.

#### 5.1.7.6 Collaborative Editor

This property informs whether the tool is a collaborative editor.

#### 5.1.7.7 Configuration Management/Versioning Support

This property informs whether the tool supports direct access to some kind of a configuration management system like Rational ClearCase or CVS or features an own versioning mechanism.

### 5.1.8 Documentation

This category deals with the documentation available for the tool and the quality of this documentation.

#### 5.1.8.1 Documentation for Installation and Configuration

This property informs whether there is documentation for installation and configuration of the tool available and about the quality of this documentation.

#### 5.1.8.2 User Manual

This property informs whether there is a user manual for the tool available and about the quality of this documentation.

## 5.2 Setting up the generic Evaluation Framework for Tools supporting the Distributed Pair Programming Practice

In the previous chapter a list of requirements or properties to be fulfilled to some extent by a tool that is usable or specifically designed for the application of the Distributed Pair Programming practice was described. This chapter documents how these desirable properties are used to build up a generic evaluation framework for tools supporting the Distributed Pair Programming practice. As a basis for the setup of the generic evaluation framework for tools supporting the Distributed Pair Programming practice the abstract model to select software development tools as defined by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] which is based on the DESMET project [Kitchenham, Linkman and Law, 1997] is used.

The model described by Zwartjes and van Geffen is based on qualitative analysis of tools and their properties and provides a visualization of an evaluation of several tools. The same is valid for the evaluation framework that is to be set up for the evaluation of the appropriateness with respect to usability, effectiveness and efficiency of tools supporting the Distributed Pair Programming practice. The model described by Zwartjes and van Geffen consists of a tool matrix that is built up by executing the following five steps [Zwartjes and van Geffen, 2005]:

- Step 1: Find or refine desired categories of tool support.
- Step 2: Specify the required tool properties and their importance in the tool matrix.
- Step 3: Populate the rows of the tool matrix with tools and its cells with indications of compliance.
- Step 4: Analyze the tool matrix.
- Step 5: Select a set of tools.

These steps and their application in scope of the setup of the generic evaluation framework for Tools supporting the Distributed Pair Programming practice will be discussed in the following subchapters.

### 5.2.1 Step 1: Find or refine desired categories of tool support

According to Zwartjes and van Geffen [Zwartjes and van Geffen, 2005], the first step in the process of selecting a set of tools is to specify the desired categories of tool support. In contradiction to the presumption taken by Zwartjes and van Geffen in the description of their model the interest of the generic evaluation framework for tools supporting the Distributed Pair Programming practice actually lies within one single tool category that is tools supporting the Distributed Pair Programming practice. Although this single category can be split into screen-sharing tools and collaboration-aware tools, both of these tool categories actually have the same area of application and most of the eligible properties are the same for both categories. Therefore this does not match the idea of the first step as described by Zwartjes and van Geffen.

Instead in the single area of application of Distributed Pair Programming it makes sense to group the eligible properties of tools supporting the Distributed Pair Programming practice into property categories as done in chapter 5.1. Therefore the step of finding or refining desired categories of tool support is applied in scope of the setup of the generic evaluation framework for tools supporting the Distributed Pair Programming practice by defining two tool categories named “Screen-Sharing Tools” and “Collaboration-Aware Tools” along with the following eight property categories that are shared by both of the tooling categories and which are described in detail in chapter 5.1.

- Basic Properties
- Support of Gesturing
- Floor Control Support
- Communication Possibilities
- Platform Independence
- Usability Properties
- Tool Properties

- Documentation

With this the first step of the construction of a tool matrix for the generic evaluation framework for tools supporting the Distributed Pair Programming practice is executed.

### 5.2.2 Step 2: Specify the required tool properties and their importance in the tool matrix

As the identified categories are known now, within this second step on the way to construct a tool matrix for the generic evaluation framework for tools supporting the Distributed Pair Programming practice these categories are to be broken down into the properties desired from the tools. According to Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] these desired properties may be in the form of desired features but should also include quality attributes such as usability, learning curve or platform support. The break down of the property categories into concrete desirable properties of tools supporting the Distributed Pair Programming practice can be found in chapter 5.1.

What is missing in chapter 5.1 is the definition of the criticalities of the properties that defines a measure of each properties importance for the appropriateness with respect to usability, effectiveness and efficiency of tools supporting the Distributed Pair Programming practice. In the model described by Zwartjes and van Geffen, three levels of importance are defined. In scope of the appliance of the model for the setup of the generic evaluation framework for tools supporting the Distributed Pair Programming practice it was found that four levels of importance are needed to provide a comprehensive evaluation and description of the usability of tools for the Distributed Pair Programming practice.

#### 5.2.2.1 Properties of Tools supporting the Distributed Pair Programming Practice of Critical Importance

Properties of critical importance are such properties that are basic to enable the effective performance of the Distributed Pair Programming practice in a similar way to the performance of traditional collocated Pair Programming. In other words properties of critical importance are such properties that are either absolutely needed by the tooling to enable two non-collocated partners to collaborate or are basic collaborative activities in the traditional Pair Programming practice. The following properties are defined as of critical importance:

| **Property Category** | **Property** |
|---|---|
| Basic | Support of Screen Sharing |
| Basic | Support of Collaboration Awareness |
| Basic | Support of Workspace Awareness |
| Support of Gesturing | Availability of a Second Pointer for the Navigator |
| Floor Control Support | Support of Floor Control |
| Floor Control Support | Support of Role Changes |
| Usability | Screen Actuality - Realtime-Probability |
| Usability | Easiness of Collaboration |

*Table 1. Properties of Critical Importance*

If any of the critical properties accept for the basic properties is not fulfilled by a tool this does not mean that the tool can not be used for supporting the Distributed Pair Programming practice, but rather that when this tool is used in the appliance of the Distributed Pair Programming practice the users will face severe drawbacks or shortcomings compared to users collaborating in traditional collocated Pair Programming.

#### 5.2.2.2 Properties of Tools supporting the Distributed Pair Programming Practice of High Importance

Properties of high importance are such properties that are highly important to a tools usability, efficiency and effectiveness with respect to its use in the Distributed Pair Programming practice. The following properties are defined as of high importance:

| **Property Category** | **Property** |
|---|---|
| Support of Gesturing | Support of Highlighting with Respect to Collaboration |
| Communication Possibilities | Support of a Speech Channel |
| Usability | Intuitiveness of tool usage |
| Usability | Stability |
| Usability | Need for other tools |

*Table 2. Properties of High Importance*

If any of the highly important properties is not fulfilled by a tool this does not mean that the tool should not be used for supporting the Distributed Pair Programming practice, but rather that when this tool is used in the appliance of the Distributed Pair Programming practice the users will face some major drawbacks or shortcomings compared to users collaborating in traditional collocated Pair Programming.

#### 5.2.2.3 Properties of Tools supporting the Distributed Pair Programming Practice of Medium Importance

Properties of medium importance are such properties that are significant and desirable assets to a tools usability, efficiency and effectiveness with respect to its use in the Distributed Pair Programming practice. The following properties are defined as of medium importance:

| **Property Category** | **Property** |
|---|---|
| Support of Gesturing | Availability of Semantic Adaptations of Pointers with Respect to Collaboration |
| Floor Control Support | Support of Informal Role Changes (social Floor control) |
| Floor Control Support | Support of Formal Role Changes (application/master-user mediated Floor control) |
| Floor Control Support | Easiness of Role Changes |
| Floor Control Support | Support of the Information "Who is in which role" |
| Communication Possibilities | Support of textual Chat |
| Communication Possibilities | Support of a White-Board |
| Platform Independence | Support of Microsoft Windows |

| Property Category | Property |
|---|---|
| Platform Independence | Support of Unix/Linux |
| Usability | Easiness of Start up and establishing a session |
| Usability | Intuitiveness of User interface |
| Usability | Support of Session Management |

*Table 3. Properties of Medium Importance*

If any of the medium important properties is not fulfilled by a tool this might make the work in a Distributed Pair Programming team more complicated but the users will face only minor drawbacks or shortcomings compared to users collaborating in traditional collocated Pair Programming.

#### 5.2.2.4 Properties of Tools supporting the Distributed Pair Programming Practice of Low Importance

Properties of low importance are such properties that are minor assets to a tools usability, efficiency and effectiveness with respect to its use in the Distributed Pair Programming practice and therefore rather nice-to-have features or properties providing additional information about a tool than really necessary for the support of Distributed Pair Programming. The following properties are defined as of low importance:

| Property Category | Property |
|---|---|
| Support of Gesturing | Suppression of Telepointer Jitter |
| Floor Control Support | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role |
| Communication Possibilities | Support of a Video Channel |
| Communication Possibilities | Pair/Partner Finding Support |
| Platform Independence | Support of MacOS |
| Usability | Easiness of Installation and configuration |
| Usability | Support of Syntaxhighlighting |
| Tool | Off-the-shelf - Commercial |
| Tool | Off-the-shelf - Open Source |
| Tool | Special Purpose |
| Tool | IDE |
| Tool | Viewer |
| Tool | Collaborative Editor |
| Tool | Configuration Management/Versioning Support |
| Documentation | Documentation for Installation und configuration |
| Documentation | User Manual |

*Table 4. Properties of Low Importance*

### 5.2.3 Step 3: Populate the rows of the tool matrix wit tools and its cells with indications of compliance

According to the model described by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] the rows of the tool matrix are populated with tools and their compliance to those properties after all required properties have been specified and categorized into corresponding criticalities. The most important consideration of this step is obviously which tools are to be evaluated by using the tool matrix? The selection of tools for the actual evaluation is done in scope of the appliance of the evaluation framework within a concrete tool evaluation. The selection of those tools evaluated in scope of this diploma thesis is therefore described in chapter 6.1.

The population of the cells of the matrix is done by evaluating the tools whether and to which extent they comply with the identified properties. In scope of the generic evaluation framework for tools supporting the Distributed Pair Programming practice a range from 1 to 5 was used to measure the level of compliance of a tool to a property where 5 is maximum compliance and 1 is minimum compliance. In some cases, especially those, where a property was not covered at all by a tool, 0 was used additionally to signal that a certain property was not fulfilled by the tool at all.

The values set for the compliance to the identified properties of a selected tool can therefore be used to provide quantification on how good the evaluated tool supports the Distributed Pair Programming practice by summing up the set values either per property category or for the whole tool. To include the importance of each property into the quantification of the evaluation results, additionally a weighting depending on the importance is introduced that provides a second value for comparison of different tools evaluated within the matrix which also includes the importance levels of the identified properties. The following weighting is defined for the generic evaluation framework for tools supporting the Distributed Pair Programming practice:

- Critical importance … Weighting: 10
- High importance … Weighting: 5
- Medium importance … Weighting: 2,5
- Low importance … Weighting: 1

Further on some evaluation hints and partly even definitions which values to be used for varying levels of compliance can be given for the generic evaluation framework for tools supporting the Distributed Pair Programming practice. This can be found in the following tables.

| **Property Category** | **Property** | **Evaluation hints** |
|---|---|---|
| Basic | Support of Screen Sharing | Rated 1 .. 5 (where 5 is best)<br>Can be rated based on how good the screen sharing is done. Issues to be included within this rating are i.e. whether only a section of the partners screen or the whole one (with a possible resolution change to fit the other screen) is replicated, whether the shared screen is only a small window, and so on. |
| Basic | Support of Collaboration Awareness | Rated 1 .. 5 (where 5 is best) |

| Property Category | Property | Evaluation hints |
|---|---|---|
| Basic | Support of Workspace Awareness | Rated 1 .. 5 (where 5 is best)<br>Rates the feeling of workspace awareness while evaluating the tool. |

Table 5. Evaluation Hints for the Basic Properties Category

| Property Category | Property | Evaluation hints |
|---|---|---|
| Support of Gesturing | Availability of a Second Pointer for the Navigator | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if it is not possible for the Navigator to get a pointer.<br>Rated 1 if there is only one pointer but this can be shared between Driver and Navigator.<br>Rated 2 to 5 if a 2nd pointer for the Navigator is available. Nuances between 2 and 5 rated with respect to how good and usable the 2nd pointer is implemented. |
| Support of Gesturing | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if there is no 2nd pointer or if there are no semantic adaptations.<br>Rated 1 if semantic adaptation is at least so far available that it is clear which pointer belongs to which user or role.<br>Rated 2 to 5 for further semantic adaptations of the Navigators pointer and their usefulness. |
| Support of Gesturing | Suppression of Telepointer Jitter | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if it is almost impossible to follow your partners mouse movement due to telepointer jitter or if the partners pointer is not visible on the other partners screen. |
| Support of Gesturing | Support of Highlighting with Respect to Collaboration | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if such highlighting is not possible. |

Table 6. Evaluation Hints for the Support of Gesturing Property Category

| Property Category | Property | Evaluation hints |
|---|---|---|
| Floor Control Support | Support of Floor Control | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if no floor control is provided by the tooling. |
| Floor Control Support | Support of Role Changes | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if no support for role changes is given. |
| Floor Control Support | Support of Informal Role Changes (social Floor control) | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if no support for informal role changes is given. |
| Floor Control Support | Support of Formal Role Changes | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if no support for formal role changes is given. |
| Floor Control Support | Easiness of Role Changes | Rated 1 .. 5 (where 5 is best)<br>A better rating signals less impact to the Pair Programming workflow when doing the role changes.<br>Rated 0 if it is not possible to change roles. |

| Property Category | Property | Evaluation hints |
|---|---|---|
| Floor Control Support | Support of the Information "Who is in which role" | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if no such information is provided by the tooling. |
| Floor Control Support | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | Rated 1 .. 5 (where 5 is best)<br>Rated 0 if nothing like this is provided by the tooling.<br>Rated 1 to 3 if either presetting of time slices for reminders or statistical information of the times in each role is available.<br>Rated 2 to 5 if both presetting of time slices for reminders and statistical information of the times in each role is available. |

*Table 7. Evaluation Hints for the Floor Control Support Property Category*

| Property Category | Property | Evaluation hints |
|---|---|---|
| Communication Possibilities | Support of a Speech Channel | Rated 1 .. 5 (where 5 is best) based on how good the speech channel is supported by the tooling.<br>Rated 0 if no speech channel is supported. |
| Communication Possibilities | Support of textual Chat | Rated 1 .. 5 (where 5 is best) based on how good the chat channel is supported by the tooling.<br>Rated 0 if no chat channel is supported. |
| Communication Possibilities | Support of a White-Board | Rated 1 .. 5 (where 5 is best) based on how good the Whiteboard channel is supported by the tooling.<br>Rated 0 if no Whiteboard channel is supported. |
| Communication Possibilities | Support of a Video Channel | Rated 1 .. 5 (where 5 is best) based on how good the Video channel is supported by the tooling.<br>Rated 0 if no Video channel is supported. |
| Communication Possibilities | Pair/Partner Finding Support | Rated 1 .. 5 (where 5 is best) based on how good pair finding is supported by the tooling.<br>Rated 0 if there is no support of pair finding by the tooling. |

*Table 8. Evaluation Hints for the Communication Possibilities Property Category*

| Property Category | Property | Evaluation hints |
|---|---|---|
| Platform Independence | Support of Microsoft Windows | Rated 1 if Windows is supported by the tooling.<br>Rated 0 if Windows is not supported. |
| Platform Independence | Support of Unix/Linux | Rated 2 if Linux and Unix are supported by the tooling.<br>Rated 1 if Linux or Unix is supported by the tooling.<br>Rated 0 if neither Linux nor Unix is supported. |
| Platform Independence | Support of MacOS | Rated 1 if MacOS is supported by the tooling.<br>Rated 0 if MacOS is not supported. |

*Table 9. Evaluation Hints for the Platform Independence Property Category*

| Property Category | Property | Evaluation hints |
|---|---|---|
| Usability | Intuitiveness of tool usage | Rated 1 .. 5 (where 5 is best) based on how intuitive the usage of the tool is while working with it. |
| Usability | Easiness of Installation and configuration | Rated 1 .. 5 (where 5 is best) based on how easy it is to install and configure the tool (and other needed tools). |
| Usability | Easiness of Start up and establishing a session | Rated 1 .. 5 (where 5 is best) based on how easy and intuitive it is to start the tool (along with possibly other needed tools) and establish a Pair Programming session. |
| Usability | Stability | Rated 1 .. 5 (where 5 is best) based on the impression from the evaluation tests about how stable the tool seems.<br>Please note that this is only a first impression as it is not in scope of the evaluation to perform a load and stress or duration test. |
| Usability | Screen Actuality - Realtime-Probability | Rated 1 .. 5 (where 5 is best) |
| Usability | Easiness of Collaboration | Rated 1 .. 5 (where 5 is best) |
| Usability | Intuitiveness of User interface | Rated 1 .. 5 (where 5 is best) |
| Usability | Support of Syntaxhighlighting | Rated 1 .. 5 (where 5 is best) based on how good you feel that Syntax-highlighting is supported by the collaborative editor tooling.<br>Rated 0 if no Syntaxhighlighting is supported. |
| Usability | Need for other tools | Rated 1 .. 5 (where 5 is best - no other tools needed)<br>Rated 4 if only some additional environmental tools (Java VM, Editor for screen-sharing tools) which are commonly available or tools substituting a single communication channel which are commonly available are needed.<br>Rated 3 if more than one communication channel needs to be substituted or if not so commonly available environmental tools are needed.<br>Rated 2 if both of the above is the case.<br>Rated 1 if a whole set of commonly not so available tools needs to be installed where part of them is possible even necessary to have licenses for. |
| Usability | Support of Session Management | Rated 1 .. 5 (where 5 is best) based on how good it is supported.<br>Rated 0 if the tool does not support session management. |

*Table 10. Evaluation Hints for the Usability Property Category*

| Property Category | Property | Evaluation hints |
|---|---|---|
| Tool | Off-the-shelf - Commercial | Rated 1 if the tool is a commercial off-the-shelf tool.<br>Rated 0 if not. |
| Tool | Off-the-shelf - Open Source | Rated 1 if the tool is an open-source off-the-shelf tool.<br>Rated 0 if not. |
| Tool | Special Purpose | Rated 1 if the tool is specially designed for support of distributed development or even Distributed Pair Programming.<br>Rated 0 if not. |
| Tool | IDE | Rated 1 if the tool is an IDE.<br>Rated 0 if not. |
| Tool | Viewer | Rated 1 if the tool is a Viewer.<br>Rated 0 if not. |
| Tool | Collaborative Editor | Rated 1 if the tool is a collaborative editor.<br>Rated 0 if not. |
| Tool | Configuration Management/Versioning Support | Rated 1 if the tool provides configuration management/versioning support by itself.<br>Rated 0 if not. |

*Table 11. Evaluation Hints for the Tool Properties Category*

| Property Category | Property | Evaluation hints |
|---|---|---|
| Documentation | Documentation for Installation und configuration | Rated 1 .. 5 (where 5 is best) based on the quality of the installation and configuration documentation of the tooling.<br>Rated 0 if no such documentation exists. |
| Documentation | User Manual | Rated 1 .. 5 (where 5 is best) based on the quality of the user manual documentation of the tooling.<br>Rated 0 if no such documentation exists. |

*Table 12. Evaluation Hints for the Documentation Property Category*

### 5.2.4 Step 4: Analyze the tool matrix

When reaching this step of the model of Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] all appropriate cells of the tool matrix are filled with the required properties, the tools to be evaluated as well as a the grade of compliance to which the tools evaluated fulfil the properties listed.

In the case of the generic evaluation framework for tools supporting the Distributed Pair Programming practice within this step a comparison of the evaluated tools based on the quantification of the grade of fulfilment of the various properties by the evaluated tools can be done, either based on the unweighted or on the weighted ratings. This comparison can either be done based on each individual property, based on the property categories or to get an overview on the complete evaluation.

By doing the analysis of the evaluation results within the tool matrix it can also be easily found whether there are any properties desired to be present within tools supporting the Distributed Pair Programming practice that are not fulfilled by any of the evaluated tools at all. If this is the case three cases resulting in this situation can be distinguished.

- The property might be fulfilled by one or more of the evaluated tools but this was missed during the evaluation.
- There might be tools available that support the desired property but were not included in the actual evaluation. In this case it would make sense to apply the generic evaluation framework for tools supporting the Distributed Pair Programming practice to a wider range of tools than could be evaluated in scope of this diploma thesis.
- There are no tools available supporting the desired functionality. The development of a tool supporting this functionality along with the other desired functionality might be an issue of future work.

### 5.2.5 Step 5: Select a set of tools

In the abstract model to select software development tools as defined by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] within this step an optimum set of tools is selected for the development process and task at hand by going through the matrix. This is actually not valid for the generic evaluation framework for tools supporting the Distributed Pair Programming practice as the target of tool evaluations done with the set up evaluation framework is to get comparable results for tools for the single activity of the Distributed Pair Programming practice and not to get a complete set of tools needed for a development process or a concrete project. Based on the analysis of the evaluation results done in step four the best tool to be used in the specific appliance of the Distributed Pair Programming practice can be selected.

# 6 Evaluation of Tools supporting the Distributed Pair Programming Practice

This chapter discusses the actual evaluation of a number of tools supporting or even specifically designed for the Distributed Pair Programming practice with the help of the generic evaluation-framework for tools supporting the Distributed Pair Programming practice that is discussed in detail in chapter 5 of this diploma thesis. The target of the evaluation is to get a first overview on how suitable available tools are in supporting the Distributed Pair Programming practice and whether there are any properties found to be desirable to be covered by tools used for the support of the Distributed Pair Programming activity that are not covered by any of the available tools.

Due to the fact that an empirical evaluation of a number of the tools found with potential to be supportive or which are specifically designed for Distributed Pair Programming was not possible the evaluation in scope of this diploma thesis was done in the way of a case study where each of the tools evaluated was evaluated by a single pair of students.

As it was not possible to evaluate all the tools found due to time and effort constraints, a selection of a subset of the found tools supporting or specifically designed for the Distributed Pair Programming practice was needed to be done. The selection of the tools for the actual evaluation is discussed in chapter 6.1.

Chapter 6.2 discusses how the generic evaluation-framework for tools supporting the Distributed Pair Programming practice was applied in scope of the actual evaluation of the tools.

The final subchapters of this chapter describe the evaluation results on a per tool basis as well as provide a comparison of the tools based on the results of the evaluation.

## 6.1 Tool Selection

The actual selection of the tools to be taken into the evaluation of tools supporting the Distributed Pair Programming practice in scope of this diploma thesis was an easy one for screen-sharing applications. As discussed above this tool category can be divided into the sub-categories of tools that only provide basic screen-sharing functionality and such tools that provide a rich environment for non-collocated collaboration besides the actual screen-sharing functionality. As the most prominent candidates for each of these sub-categories is commonly available as well, those two, namely VNC respectively its widely known implementation RealVNC for the sub-category of tools that only provide basic screen-sharing functionality and Microsoft's NetMeeting as an example of a tool that provides a rich environment for non-collocated collaboration besides the actual screen-sharing functionality, were selected to be taken into the evaluation of tools supporting the Distributed Pair Programming practice in scope of this diploma thesis.

For the category of collaboration-aware applications the selection of tools to be taken into the evaluation of tools supporting the Distributed Pair Programming practice in scope of this diploma thesis was more difficult. On the one hand this was the case due to the fact that the number of tools with potential to support the

Distributed Pair Programming practice found within this category was larger and much more diverse than for screen-sharing applications and on the other hand that not all tools found in this category were readily available or needed resources not available to the evaluation team.

For the sub-category of collaborative editors ACE, Gobby, GrewpEdit, MoonEdit as well as the TalkAndWrite plug-in for Skype and COPPER as a representative of collaborative editors specifically designed for the practice of Distributed Pair Programming were selected. SynchroEdit was not selected as according to the information available about it, it is the least feasible of the tools found in this sub-category for collaboratively working on code. SubEthaEdit could not be included in the evaluation as this tool needs a MacOS operating system to run on and is not available for any other operating system. Unfortunately none of the student apprentices participating in the evaluation had access to a Mac OS. The development of the TUKAN system was stopped some time ago according to information received from Till Schümmer, one of its developers, and is not available in a version working on nowadays commonly used operating systems. Thus it could not be included in the evaluation.

Neither of the collaborative IDEs of Borland, CodeWright and JBuilder 2007, is available in form of a trial version. Both of them are only available via expensive licenses. This prevented them from being included in the evaluation of tools supporting the Distributed Pair Programming practice done in scope of this diploma thesis. As the developers of the Moomba system did not respond to any queries sent to them at all, Moomba's MCIDE was not evaluated even though it would have been quite interesting to have it evaluated as it is actually the only fully fledged IDE specifically designed for the use in the Distributed Pair Programming practice. During the research work in scope of this diploma thesis a number of eclipse plug-ins specifically designed for the support of the Distributed Pair Programming practice was found. Two of them, namely PEP - Pair Eclipse Programming and XPairtise, were selected to be included in the evaluation of tools supporting the Distributed Pair Programming practice. This is due to the fact that the features of the various Eclipse plug-ins found are quite similar and therefore it seemed to make no sense to include a larger number of these plug-ins in the evaluation. Instead it seemed more interesting to include the extensions to other IDEs in the evaluation as well. Thus DocSynch in its implementation for jEdit and the NetBeans Collaboration Project also known as collab.NetBeans where included in the evaluation. Although it would have been of some interest the Distributed XP Support Tool developed by Atsuta and Matsuura being the only enhancement found specifically designed for Distributed Pair Programming to Microsoft's IDE Visual Studio.Net could not be included in the evaluation as the developers did not respond to any query sent to them.

Unfortunately it was not possible to include any of the tools or tooling environments that take a completely different approach to the support of the Distributed Pair Programming practice into the evaluation of tools supporting the Distributed Pair Programming practice done in scope of this diploma thesis. Hypervideo requires a number of video projectors which were not available. Besides, neither the Hypervideo nor the Transparent Video Facetop system developers responded to the queries sent to them. Therefore both systems were unavailable for the evaluation. Nor could Telescope be included in the evaluation as it was not possible to get hand on the tooling and almost all documentation is available in Spanish only.

To summarize the selection of tools for the evaluation of tools supporting the Distributed Pair Programming practice in scope of this diploma thesis, the following list of tools was evaluated for their usability in the support of the Distributed Pair Programming practice in scope of this diploma thesis:

- ACE - A Collaborative Editor
- COPPER
- DocSynch
- Gobby
- GrewpEdit
- Microsoft NetMeeting
- MoonEdit
- PEP - Pair Eclipse Programming
- The NetBeans Collaboration Project
- The TalkAndWrite plug-in for Skype
- Virtual Network Computing (VNC)
- XPairtise

## 6.2 Evaluation of the selected Tools

In scope of the evaluation of tools supporting the Distributed Pair Programming practice done in scope of this diploma thesis a selected number of tools found in the research for this diploma thesis that seem to be potentially usable or are even specifically designed for the support of the Distributed Pair Programming practice are to be evaluated with the help of the generic evaluation framework for tools supporting the Distributed Pair Programming practice described in chapter 5.2. The selection of the tools to be included in this evaluation can be found in chapter 6.1.

The actual evaluation of the tools with the generic evaluation framework for tools supporting the Distributed Pair Programming practice was done by pairs of student apprentices. Each of the pairs evaluated a number of tools potentially usable or even specifically designed for the support of the Distributed Pair Programming practice in scope of a case study.

To provide kind of a guideline for the evaluation and to cover all identified and defined eligible properties for tools supporting the Distributed Pair Programming practice which are integrated in the generic evaluation framework a number of high level scenarios were defined in scope of a workshop in which all student apprentices participating in the tool evaluation participated. These scenarios were used as a guideline in scope of the actual evaluation as it was found to make no sense to design a concrete programming example that covers all the defined required properties listed in the generic evaluation framework for tools supporting the Distributed Pair Programming practice and is to be implemented with the help of each of the tools in scope of Distributed Pair Programming sessions over and over again. Thus these scenarios describe which activities are to be done during an evaluation session. The description of these scenarios can be found in the appendix in chapter 10.2.

## 6.3 Evaluation Results

The following subchapters document the results and findings of each tool's evaluation. A comparison of the evaluated tools will be made in chapter 6.4. The filled out evaluation framework for tools supporting the Distributed Pair Programming practice with the quantified results of the evaluation of the selected tools can be found in the appendix in chapter 10.3.

### 6.3.1 ACE - A Collaborative Editor

ACE, as its name suggests, is a collaborative editor providing needed functionality for collaboratively editing or reviewing textual artefacts but not implementing any more complex collaborative features. It is available for Windows, UNIX and Linux. There are no mechanisms specific to Distributed Pair Programming supported by ACE which is no surprise as this collaborative editor was obviously not designed for the support of the Distributed Pair Programming practice. Nevertheless the most basic required properties of a collaboration-aware tool to be usable in Distributed Pair Programming are fulfilled to a sufficient degree to make ACE applicable as a tool for the support of Distributed Pair Programming.

Being a collaborative editor ACE supports collaboration awareness in a general sense. It provides a list of partners the user is collaborating with in the current session. Related to the Distributed Pair Programming activity the collaboration awareness is naturally a bit dimmed as ACE does not feature any Distributed Pair Programming specific mechanisms like the role concept along with the support of role changes, the lack of a second pointer and so on. This lack of Distributed Pair Programming specific mechanisms heavily impacts the support of workspace awareness of the tool. The awareness of what the partner is doing at the very moment is quite dimmed. The main reason for this, apart from the already mentioned lack of Distributed Pair Programming specific mechanisms is the fact that the user interface has some obvious flaws. ACE does not provide any functionality to inform its user where the partner currently has his focus in the document or, in case there are multiple documents open, in which document he has his focus. Also the colour markings in the document led to some confusion with respect to their meaning. Further on, although as long as both partners keep their focus on the same section of the same artefact the position of each other's cursor is shown to the remote user by a little mark the workspace awareness still heavily lacks from the absence of the replication of the remote partner's pointer. Each partner has his own local pointer, with which he can mark text locally and so on, but this is not replicated to the remote partner's application and thus it is not possible to give any deictic references at all.

As discussed, ACE does not support the role concept of Distributed Pair Programming with the roles of Driver and Navigator. Both partners can edit in the artefact they work on whenever and wherever they want. The role distribution between Driver and Navigator as well as any role changes needs to be verbally and socially coordinated and agreed by the two partners.

ACE's client SW is quite easy to install and is started by simply starting the ACE program under Windows. Precondition is that Java Runtime Environment 1.4.2 or higher as well as Bonjour SW is installed and running. The establishment of a session needs to be done manually. This is a bit complicated as it needs multiple steps. First the partner has to be locked in via ACE, then the document that is to be worked on needs to be opened and published via ACE, then the pairing user can request to join the session or the user who published the document can invite his pairing user into the session which then has finally to be accepted by the respective other partner. Nevertheless although a little complicated the session set up works fine.

In contradiction to the straight forward session set up the evaluation showed big problems in cases where a session is broken. Reconnecting to the partner and rejoining the same session only worked once during the evaluation. All the other times the session had to be built up by both partners from scratch again. There are some countermeasures implemented like the possibility to save terminated documents, but that did not work out either. In general ACE makes the impression of being not very stable at all. It also crashed on the first trial to connect to the evaluation partner and created high CPU load problems in cases when the LAN connection was broken.

With respect to collaboration ACE follows the design that a document is published by one developer and the partner can join this document. In this way the document moves to the work area of the partner who joined the session successfully. So there is some sort of notification on both sides that more than one person is currently working on the document. One partner can leave the document, but as already mentioned, when trying to rejoin, there were big problems for both sides returning to the previous state experienced during the evaluation. The screen refreshing rate is good, but when the Driver writes something outside of the Navigator’s current focus the tool gives no hint of the Driver’s activity.

There is no support for speech or textual chat channel within ACE. Nevertheless a very limited text-only whiteboard can be emulated by means of using a separate published document in the collaborative editor for this cause.

Concluding it can be said that ACE being a collaborative editor is definitely not designed for the support of the Distributed Pair Programming practice but nevertheless can still be used for the same although with some major restrictions in the collaboration in the sense of Pair Programming. Of those tools evaluated the evaluation showed it to be the least convenient to be used for the support of the Distributed Pair Programming practice.

### 6.3.2 COPPER

COPPER is a tool specifically designed for Distributed Pair Programming, which became clear immediately after starting the software because there was a traffic light in the top left corner to signalise the current role the user was in. While working together, a radar view window gave clues which part of the screen the partner was looking at right at that moment. By using the Jabber client, instant messaging was more or less the only possibility to communicate with each other. A WEBDAV document server would have been needed to test the full range of features COPPER provides, but was unfortunately too tricky to set up correctly. The installation instructions written in Spanish did not help much.

There are some quite supportive features in COPPER providing a sense of collaboration as well as workspace awareness. Although there is no information upon who is connected when successfully establishing a session at least a notification is given in the Status line that the connection to the entered IP-address is established. COPPER’s usage in the Distributed Pair Programming environment felt quite intuitive though there are some flaws in the usage of the tool. A radar view provides the information to each of the partners where the other partner has his focus in the document which is worked on at any given time. The traffic sign clearly communicates which partner impersonates which role. The most obvious drawbacks with respect to Distributed Pair Programming is the lack of a second pointer for the Navigator that is replicated to the Drivers screen as well as that the Navigator is able to mark some sections in the text while the Driver types text in and the text marked by the Navigator is then overwritten with what the Driver is typing. Still the possibility that the Navigator highlights some code which is displayed to the Driver provides a minimum support of deictic referencing. A minor drawback is that in cases where the connection to the partner was broken for example due to a LAN outage, there is no indication that the other partner is not there anymore.

The traffic light provides a quite good and useful mechanism for floor control although it might be a bit misplaced if both partners heavily focus on the code in the collaborative workspace. It is extremely clear which role is impersonated by whom at any given time. The partner who opens a document starts as Driver. The implementation of the floor control mechanism is a quite formal one although this does not impact the collaborative work much. There is no possibility for informal role changes. Write access to the shared artefact can be either granted by the Driver if he wants to take over the Navigator role or the Navigator can request a role change via the tool itself.

The part of the COPPER software that could be installed and configured which was basically the collaborative editor needed some time to figure out how this is done but then was quite easy. A Jabber account is needed, but can be easily created via the integrated client. The setup of the WebDAV server was not possible due to problems with the Spanish installation instructions. COPPER's client software is quite easy to start by simply starting it in the Java runtime environment. Two windows are opened, one called Nombre which is the Jabber Network connection tool implemented for COPPER and the other called Eternity which is the actual collaboration-aware editor. A minor drawback is that for establishing a session it is necessary to log-in twice, once in Nombre to a Jabber server, and after this is done in Eternity to the Host and Port of the partner which is only necessary for that one partner who wants to connect to the others machine.

It is quite easy to collaborate with the use of COPPER except for the already mentioned problems regarding the Navigator highlighting problem. Regarding the user interface, it is a bit annoying that the built-in Jabber client is obscured after starting COPPER by the Eternity window. The dialog for saving artefacts was not quite user friendly since it was not possible to access or save anything elsewhere than on the C partition and only the original Driver or document owner was able to save the current document.

The Nombre connection tooling to the Jabber server is automatically started when starting the tooling. Instant Messaging in an SMS like way is supported by the Nombre tooling. As a limited text-only whiteboard a separate document in the Eternity collaborative editor can be used. According to the available documentation it seems that kind of a voice chat mechanism is implemented in COPPER, but as it was not possible to install the needed WebDAV-Server with the available documentation this could not be evaluated. Java Runtime Environment 1.3.1 is additionally needed to be installed on the client machines. According to the available documentation it seems that kind of a session management is supported by COPPER via the WebDav-Server document server but due to the mentioned problems of setting up such a server this could not be evaluated.

The documentation for installation and configuration was only available in Spanish and had to be translated as good as possible with only rudimentary knowledge of this language. Due to this the configuration of the WebDAV-Server was not possible. There is no user manual available but only a Spanish presentation with some rudimentary information.

All in all, COPPER is an interesting approach to support the Distributed Pair Programming practice and does a rather good job in supporting most of the properties required from a tool for this issue, but still lacks some important ones like for example a second pointer for the Navigator to enable the full scale of deictic referencing. Nevertheless COPPER is one of the tools evaluated being best suited for the support of Distributed Pair Programming.

### 6.3.3 DocSynch

DocSynch is described as an enhancement to various single user editors and IDEs enhancing them so that it is possible to use them in a non-collocated collaborative way. In its currently available implementation it only supports the enhancement of the Java editor jEdit.

The installation of jEdit as well as the DocSynch extension to this editor is easy and does not make any problems. JEdit is simply started. As the technology used by DocSynch is based on the usage of IRC it is necessary to have IRC installed and activated in jEdit to use DocSynch. It needs to be stated that DocSynch made quite some problems during the actual collaborative evaluation. It was not possible to initiate a collaborative session with the latest jEdit version. Only a trial with an older jEdit version could solve this, though nothing about a certain needed version of jEdit is stated in the documentation available for DocSynch. There were also some severe issues until it was possible to set up the network configuration in a way that an IRC session could be established.

As stated DocSynch uses an IRC based chat to support the non-collocated collaboration. Within this chat it is possible to open channels and to invite collaboration partners into the channel. Within each channel, it is possible to start sessions for the collaborative activities and to share documents within such a session.

Collaboration as well as workspace awareness is quite good while using DocSynch which is surprising as it does not support any means of deictic referencing meaning that there is no second pointer for the Navigator nor any mechanism to replicate highlighted sections of an artefact supported. It provides good information about the users that have joined a session, about who inhabits which role, which documents are shared within the session and which document is worked on.

The floor control mechanism implemented in DocSynch is very formal and moderated by a master user who is the one that started the session within a channel. This master user can assign and revoke write permission respectively the Driver’s role to anyone who has joined the session, but always only to one user. Due to the needed administrative overhead for such a moderated formal floor control mechanism there is a significant impact of role changes on the collaborative work done.

No speech channel nor a whiteboard or video channel is supported by DocSynch. These communication channels need to be substituted by the usage of some other tools. Due to its nature of using IRC for collaboration DocSynch provides a full fledged IRC like chat for textual communication. There is also a limited support for finding of a partner provided by DocSynch.

As discussed the installation of DocSynch and its respective editor is easy provided that the fitting versions are available as this is not documented in the available documentation. That is why the installation is rated quite low though not being complicated if the fitting versions are available. As soon as the installation is finished the setup of a session is quite easy. The tool is remarkably stable as soon as a session is set up. Nevertheless the screen refreshing rate and actuality are almost inappropriate for the use in Distributed Pair Programming were it is important to see what the Driver is typing instantly to be able to collaborate in a reasonable way. This also impacts the easiness to collaborate as it is not so easy to follow the Driver’s thoughts when impersonating the Navigator’s role. There is no session management available with DocSynch. It is only possible to store the collaboratively changed document.

Installation as well as user documentation is available but significantly lacking certain important information.

Summarizing the experiences with DocSynch it needs to be stated that though it is certainly usable in the appliance of the practice of Distributed Pair Programming there are still some major drawbacks that affect the Pair Programming experience in a negative way when using DocSynch for this issue so it is definitely not first class. The same is shown by the quantitative evaluation.

### 6.3.4 Gobby

Gobby is quite a simple collaborative editor which supports non-collocated collaborative editing of multiple documents in one session and has an IRC like multi-user chat. It is available for all commonly used operating system platforms.

The installation process contains two steps, as besides the Gobby software itself additionally the Gtkmm runtime environment is required. It uses GTK+ 2.8 as its windowing toolkit and thus integrates nicely into the desktop environment. However, the installation process is still easy and quick.

The process of setting up the first session was quite intuitive during the evaluation. Gobby is based on a host-client-system whereby one user starts a session. The other partners can connect via the IP-address or the DNS of the server. The default port for the Gobby server is “6522”. As an additional asset password session protection is possible.

With respect to collaboration and workspace awareness a list of the users that have joined a collaborative session is displayed. Any text entered by any of the users is underlayed with the colour assigned to the respective user allowing using the tool efficiently for Distributed Pair Programming although some restrictions will be faced in comparison to Pair Programming in a collocated environment. A major drawback is that Gobby does not support any means to give deictic reference to their collaboration partners by neither of the users collaborating in a session.

There is also no mechanism for floor control supported by Gobby. All users collaborating in a session are able to edit the shared documents when- and wherever they want.

Gobby is quite easy to install and very intuitive to use. Start up and session establishment did not make any problems during testing. While testing the tool several crashes of the tool were observed, so there seems to be significant lack of stability of Gobby. Screen actuality and realtime probability are very good and make it very easy to collaborate by using Gobby.

The only available built-in communication channel is the IRC-like chat for communicating with partners while coding. All other communication channels need to be substituted by the usage of additional tools.

An interesting feature is that though there is no real session management a session can be protected by a password. If person A opened a password protected session, the connection fails for all other participants as soon as person A leaves the session. It is impossible to continue working together on the document but the current version and all prior opened or edited documents can be stored. Dropping out of a session by any other participant than person A has no influence on the session. The same is valid for the case of a network crash. Further on, there is no different effect if the session has no password protection. As a minor drawback it has to be stated that there is no automatic request on document storage or recovery of the last session.

The main strengths of Gobby are that it allows working together online at the same document and its integration of a chat which is very similar to IRC. Gobby can be used to work together in a group, to show partners rapid changes and questions, or for collaborative searching for bugs within a piece of code. Summarizing this, Gooby is a tool usable for the support of the Distributed Pair Programming practice although it lacks some major features one would request from such a tool like a mechanism for deictic referencing as well as the support of the role concept immanent to the Distributed Pair Programming practice.

### 6.3.5 GrewpEdit

GrewpEdit is a collaborative editor designed to support close collaboration among the students of software engineering classes. As there were some problems with logging in to a session encountered with the latest beta version of GrewpEdit, the evaluation was done with a previous version of GrewpEdit.

GrewpEdit supports some features that help in collaborating with remote partners like a rudimentary but nevertheless quite useful chat for textual communication, a whiteboard, though this is difficult to use due to the usage of uncommon symbols, the collaborative editor and a Java Output window showing the output of the Java code. Particularly the highlighting of any text in every window with the colour assigned to the user who entered the text or produced the output supports a sense of collaboration awareness as it allows to distinct which changes and comments are made by which user. Unfortunately all of these features are packed in one single screen which dims the sense of workspace awareness a bit as it is not so clear what the collaborating partner is doing at the very moment, respectively where he is putting his focus in the collaborative tool environment. In general the screen makes the impression of being somewhat cramped.

There is nothing like a second pointer for the Navigator nor any means of floor control implemented in GrewpEdit. The only means of providing deictic references is the highlighting mechanism already described that can be used to hint the partner on certain sections of the artefact the team is working on. Any issues with respect to Pair Programming's role distribution need to be agreed on and managed socially as there is no support for this within the tool.

The tool can be launched either as a Java Web Start application directly from its homepage or by downloading it as a jar file and executing it within a Java Runtime Environment. Therefore the installation of GrewpEdit is quite simple and straight forward. To set up a session the user has to enter a username and a group name. Though this sounds quite simple as well it was found during the evaluation that setting up a session was sometimes not so easy.

After starting GrewpEdit and joining a session the GrewpSession Window appears. It includes the central collaboration point where participants can communicate using a chat; sending messages either to all or to a specific user. Participants can either communicate via the chat, program, edit or compile java files or use the integrated whiteboard. An additional feature is the killring which facilitates the navigation through the program files.

The usability of the tool is acceptable as soon as one gets used to it. Throughout the evaluation, no issues with respect to stability problems were encountered and screen actuality was reasonable. Unfortunately GrewpEdit does not provide any support for session management. Though it is certainly possible to store the artefacts that were worked on collaboratively, GrewpEdit lacks anything like automatic storage requests when shutting down the tool and in the case of a network outage. As an issue of the usability of GrewpEdit it has to be stated that the online-help function does not work at all.

Compared to some of the other tools in evaluation, GrewpEdit gives the impression of being designed for usage in scope of the Distributed Pair Programming practice. It is not only possible to collaboratively edit Java code but also to compile it directly within the collaborative environment sporting an obvious asset for the work in Distributed Pair Programming teams. Nevertheless it was found to be one of those tools evaluated that are not to be recommended to be used in the actual appliance of the Distributed Pair Programming practice due to the fact that though it supports some nice features for this practice it lacks some major ones and those features that are provided are not implemented in a way seamlessly integrating them in the collaborative work.

### 6.3.6 Microsoft NetMeeting

Microsoft NetMeeting is a voice-over-IP and multi-point videoconferencing tool which comes as part of the Microsoft operating systems Windows 2000 and Windows XP and is therefore readily available to many computer users.

Its support of screen-sharing is really good with only the minor lack of a feature to adapt the screen size of the shared screen to the screen sizes of the collaborating users. It also provides a quite good sense of workspace awareness.

There is no support for a second pointer provided in NetMeeting put the control of the single pointer available on the shared screen or application can be assigned to any of the collaborating users providing a minimum support of deictic referencing by the Navigator as well as a quite formal and moderated by a master user which is that user that owns the shared desktop implementation of a floor control mechanism. If anybody else but the owner of the shared workspace is in control of the input devices, thus acting as Driver or doing some deictic referencing, the pointer of the owner of the shared desktop is changed into a stop-sign and he can see the pointer of the user in control as well. Unfortunately any other user can only see his own pointer. So there is some minimal support of semantic adaptations of pointers with respect to collaboration available, although this will not be sufficient to be an asset in Distributed Pair Programming. The same is valid for the support of highlighting with respect to collaboration as this is actually only possible for the one in control of the input devices.

As stated above NetMeeting sports a quite formal and master user moderated floor control mechanism. This makes certainly sense if it is kept in mind that NetMeeting is actually an online- or video-conferencing tool. In such an environment it is quite natural that the moderator of such a conference who in general is responsible for providing the shared environment is in control of who is allowed to edit anything in the shared workspace.

If NetMeeting is not activated in a Microsoft Windows installation this can easily be done. Instructions for this can be found on the Internet on various pages. Consequently, installing and configuring NetMeeting is pretty easy and self-explanatory. Further on, a short wizard will guide one through setting up basic preferences and provides information for according audio settings.

The main features of NetMeeting besides talking to each other are a chat, the sharing function, a whiteboard and the file transfer. Those functions can also be used during a call. Both, the chat and the whiteboard provide basic features, whereas the chat program is very rudimentary. The sharing function has much potential as it is possible to share applications on one’s computer and let other NetMeeting users us a program on one’s computer.

Though NetMeeting might be known by a number of users, its usage lacks being intuitive if a user is not used to it. This is especially valid for the establishment of a session which is not so easy as long as one does not know how this is done. NetMeeting’s documentation does not provide any help here as the described way of setting up a session did not work during the evaluation. NetMeeting seems to be quite good with respect to screen actuality and stability though one crash in a specific situation occurred during testing it. As soon as a session is established it is very easy to collaborate with the partner. The only negative impact on collaboration is the formal way of the floor control mechanism. As is quite clear for a screen-sharing tool, it is necessary to use some additional tool for the actual collaborative work on an artefact. There is no support of session management within NetMeeting.

Basically, Microsoft’s NetMeeting is obviously not specifically designed for the usage to support the Distributed Pair Programming practice. It rather provides a complete Internet conferencing solution. However, the various sharing functionalities allow its users to use other programs such as IDEs in a collaborative way and support the exchange of files between the collaborating users. Consequently, NetMeeting can be used for the support of Distributed Pair Programming and though being a screen-sharing tool it is more supportive in this than many of the collaboration-aware tools evaluated.

### 6.3.7 MoonEdit

MoonEdit is a collaborative editor and therefore with respect to the described tool categorisation counts among the collaboration-aware applications. It allows the collaborative editing of textual artefacts stored on a central server for non-collocated users and is available for Microsoft Windows as well as Linux and FreeBSD operating systems. There is also a version for the MacOS operating system available though this needs some additional tooling to work.

There is a significant lack of MoonEdit in the support of collaboration awareness. This is mainly due to the fact that the only things displayed to its users are a list of connected users and the actual editor window. There is nothing else supporting the sense of collaboration awareness. Surprisingly the feeling of workspace awareness was nevertheless quite good when doing the evaluation of this tool. This is mainly due to the fact that on both users' screens the cursors of both users are displayed which gives a feeling about where the remote partner has his focus and what he does.

MoonEdit, being a collaborative editor and obviously not specifically designed for the practice of Distributed Pair Programming, does not support any mechanisms specific to the Distributed Pair Programming practice like the role concept of Driver and Navigator. Thus the distribution of roles and the management of role changes solely rely on social interaction between the collaborating partners. As there is no support of any means of communication between the collaborating users provided by MoonEdit, additional tools are needed to enable the necessary social interaction.

Though there is no second pointer for the Navigator nor any pointer replicated at all, there is a basic support of deictic referencing as the cursors of each user is replicated to the other user's editor and can be used for providing deictic references. Additionally the text written by a specific user is highlighted in the colour assigned to that user. Unfortunately the contrast between the colours is poor which makes it somewhat difficult to distinguish them in the text.

A wizard guides through MoonEdit's installation which makes it quite straight forward and easy. As soon as the tool is installed and started, to start collaborative work it is necessary to connect to a server. There are some open servers available on the Internet. Alternatively it is possible to set up one's own server. As soon as the connection to the server is established a list of files available on that server as well as a list of users connected to that server is displayed. Certainly it is also possible to create a new file on the server. A collaborative session is started in the way that the users that want to collaborate on a file on the server each open the respective file in their MoonEdit applications.

The usage as well as the user interface of MoonEdit is quite intuitive and easy to use. Due to the fact that the GUI is only displayed in 14 colours it looks a bit outdated as long as one is not used to the tool. Screen actuality and realtime probability were great and the tool made a quite stable impression during testing it. As Driver as well as Navigator, respectively all users that have joined a document, see all interactions of each other user with the shared artefact it is quite easy to collaborate using MoonEdit provided that some other tool is available that supports the needed communication channels.

MoonEdit provides some support of session management as each change of an artefact is instantly stored on the server. Additionally it is possible to store any artefact locally. This is important with respect to the Distributed Pair Programming activity as MoonEdit being a collaborative editor does not support any means of compiling developed code.

As a summary it can be stated that MoonEdit has its strength when collaborating on textual artefacts. Nevertheless it lacks many properties that should be supported by a tool that is used for the support of the Distributed Pair Programming practice. Thus, though MoonEdit can certainly be used to in scope of Distributed Pair Programming activities it is rather at the lower end of the scale of the evaluated tools with respect to how good it supports this practice.

### 6.3.8 PEP - Pair Eclipse Programming

PEP is one of two evaluated plug-ins to the widespread and established IDE Eclipse included in this evaluation, the second one being XPairtise that will be discussed later, that are specifically designed for the support of the Distributed Pair Programming practice. It has to be stated here that the evaluation of this tool could not be fully done due to the fact that there were major flaws with respect to its usability in a collaborative environment observed during the evaluation that made collaboratively working with PEP quite difficult if not impossible at all. This might be due to the fact that PEP is still in a quite early development stage. This becomes quite obvious as soon as one tries to establish a connection for starting a Distributed Pair Programming session which during the evaluation never worked on the first trial but always on the second.

The support of collaboration awareness within PEP could be better but is still acceptable. Surprisingly for a tool specifically designed for the support of the Distributed Pair Programming practice, there are only a few special features added to the Eclipse IDE with this plug-in that actively support Distributed Pair Programming. A quite annoying detail with respect to collaboration is that the user acting as Navigator is able to edit the artefact collaborated on even while the user acting as Driver is editing as well. Further on the Driver does not see or get any information by the tool about these changes until he performs a manual synchronisation via a special button. This is definitely not in support of a fluent work process while collaborating in a Distributed Pair Programming environment.

There is a significant lack of the awareness of what one’s partner is doing at the very moment when working with PEP. This is mostly due to the fact that the program does not work in the way one would expect it to. Changing the initially given roles does not have the desired effect with respect to how the workspace reacts to it. The former-Navigator-now-Driver’s input is not replicated to the other partner’s application. Since this is a requirement that is essential for an effective Distributed Pair Programming process, PEP fails quite heavily here, especially when keeping in mind that it is specifically designed for this issue. The only nice feature observed during the evaluation with respect to workspace awareness is that in situations when more than one file is worked on in parallel and the Driver jumps between these files, the Navigator’s active window always automatically changes as well.

Though being specifically designed for the support of Distributed Pair Programming, PEP does not provide a second pointer for the Navigator nor replicates any pointer at all to the remote screen. Strangely highlighting of code sections is only possible for the user impersonating the Driver's role and due to this is of only limited support and usability from a Pair Programming perspective.

The best part about PEP's enhancements to the Eclipse IDE is its implementation of the support of floor control. A button visualizes the current role the local user impersonates by changing its colour between green for the Driver's and red for the Navigator's role upon each role change. The client user who connected his application to his partner's application upon establishment of a session always takes the Driver's role initially. However, the Navigator can any time just take control of the action without a tool-supported formal acceptance of the same by the Driver which can be a little bit problematic when the collaborating partners rely mostly on the features given by the tool when it comes to exchanging or taking over floor control. A good communication via a voice channel seems to be essential to get along without bigger problems or misunderstandings. Not even the rather helpful status messages in the PEP chat window can change this.

With respect to the usability of the tool in collaborative environments beside a good screen refreshing rate, it is almost impossible to not follow one's partner as whenever the Driver opens a new document, this document immediately pops up on the Navigator's screen, too, ensuring the partners to focus on the same document. Further on the user acting as Navigator has no possibility to switch to another document than the one the Driver is working on. Unfortunately this only works in one direction, meaning the Navigator following the Driver. For the Driver there are very few clues about what the Navigator is doing, except for checking the current synchronisation state of the document.

Being actually not a feature of PEP but of the Eclipse IDE itself, PEP nevertheless can rely on Eclipse's CVS support for configuration management which was not tested during evaluation but can be assumed to be working. In general it has to be noted that there are several special features that are provided by the Eclipse IDE as well as by many other plug-ins to Eclipse.

Wit respect to documentation an installation instruction seems to be in the making as suggested by PEP's homepage but not available yet. There is no user manual available. A short demonstration video can be used to get an idea on how PEP is supposed to work.

All in all, PEP is a work-in-progress project with the target to develop an Eclipse plug-in to support Distributed Pair Programming which could, when fully developed, eventually become a supportive tool for developers who want to follow the Distributed Pair Programming concept while sticking to Eclipse as their developing environment. But it is still quite far from being ready for that at the moment though it already shows some interesting approaches.

### 6.3.9 The NetBeans Collaboration Project (collab.NetBeans)

collab.NetBeans represents the developer collaboration feature of the NetBeans IDE, an IDE designed for the development of Java projects. It was quite obvious right from the beginning of the evaluation that collab.NetBeans is not specifically designed to support the Distributed Pair Programming practice but rather has its focus on supporting collaborative code reviewing. The feeling of the workspace awareness and the communication possibilities provided were surprisingly supportive for Distributed Pair Programming as well, although a speech channel for communication is not featured.

The sense of collaboration awareness conveyed by collab.NetBeans is fine except for the fact that no Distributed Pair Programming mechanisms like the role concept is supported, which is not surprising given the fact that the tool is obviously mainly developed for support of collaborative code reviewing. Screen actuality is a problem as the update at the Navigator's side is done only after some seconds after the Driver stopped typing. This made it difficult to follow what the partner was doing without speaking about it all the time during the evaluation. The feeling of the workspace awareness is hampered by the unclearness what each partner does or where his focus is as well as the lack of floor control support. collab.NetBeans does not provide any means to notify to each other where the respective other partner currently has his focus in the document or, if there are multiple documents open, in which document he has his focus.

As already stated there is no support of a second pointer for the Navigator, nor is any code highlighting replicated to the remote user's application. Thus there is no possibility to give deictic references to one another.

collab.NetBeans does not support the classical Pair Programming role distribution between Driver and Navigator and therefore does not support floor control in the sense of Distributed Pair Programming. Both partners can edit the artefact at work any time wherever they want. Only the actual section of the artefact where one partner edits something gets locked for the other, but nevertheless the other can edit another section or document. As Pair Programming's role distribution concept is not supported this role distribution needs to be verbally coordinated and agreed by the two partners.

Installation of collab.NetBeans is straight forward. It is started by simply starting the NetBeans IDE. Upon the first start of the Netbeans IDE after installation it is necessary to provide information about a collaboration account which can as well easily be created at that time. Login to this account can either be set to be done manually after or automatically during the IDE start-up. The session set up is done manually but is quite easy as both partners are logged in to the same collaborative server. By simply opening a conversation as the collaborative work is called in collab.NetBeans and inviting the partner who needs to accept the invitation the collaboration session is started. After this, one partner needs to publish a document or project to work on. It needs to be mentioned, that multiple projects and documents can be opened in the same collaborative session as well as that more than two users can join a collaborative session.

As discussed, screen actuality is a problem as the update at the Navigators side is done only after some seconds after the Driver stops typing. A network outage during the evaluation made it quite difficult for the accidentally disconnected partner to rejoin the collaboration session. Except for this problem, stability seemed to be no issue. collab.NetBeans does not support the feature to store a previous session although it is certainly possible to store the changed projects or files. Further on NetBeans has an integrated CVS support for configuration management that can as well be used in a collaborative environment which was not tested during evaluation but can be assumed to be working.

Concerning documentation, on the website brief but sufficient installation and configuration instructions are available. There is no user manual, only a short developer collaboration demo showing the main functionalities of collab.NetBeans.

collab.NetBeans does not support a speech channel. As this channel is an ultimate must in Distributed Pair Programming this channel needs to be substituted by another tool. Textual chat and a simple text-only whiteboard emulated by sharing a separate file for this issue are available. A minor flaw is that the collaboration window including the chat and the document currently worked on can not be displayed at the same time. To overcome this as soon as a new chat message is received this is signalled by a flashing signal on the collaboration window tag.

So to recapitulate, it can be stated that collab.NetBeans provides a quite good environment for more general and non-specific collaborative activities like code reviewing, and can also be used to serve as a Distributed Pair Programming tool, even though a number of important mechanisms for Distributed Pair Programming like the role concept and a second pointer for the Navigator are not supported. Nevertheless many other tools that were evaluated are more fitting for the application in a Distributed Pair Programming environment.

### 6.3.10 The TalkAndWrite plug-in for Skype

The TalkAndWrite plug-in for Skype is kind of a collaborative editor though not actually intended to be used as such but rather as a whiteboard. In scope of the evaluation the free version of the TalkAndWrite plug-in for Skype was used which is called “TalkAndWrite Basic”. The exclusive difference to the full version with costs called “TalkAndWrite Pro“ is that the session time of the free version is limited to ten minutes. Therefore it needs to be stated that the full version “TalkAndWrite Pro“ would be needed if the TalkAndWrite plug-in for Skype would be applied in real Distributed Pair Programming activities.

With respect to the support of collaboration as well as workspace awareness the TalkAndWrite plug-in for Skype did an astoundingly good job. Certainly this is aided by the various features with respect to collaborative activities that come with Skype but might as well be partly due to the fact that a second pointer for the Navigator is provided. Even semantic adaptations of the users’ pointers are supported. For example there is an own pointer symbol that can be activated to draw attention. So the TalkAndWrite plug-in for Skype provides the full range of support of deictic referencing to its users which is somewhat surprising as it definitely is not specifically designed for the support of the Distributed Pair Programming practice. This is quite obvious as there is no implementation of a support for floor control at all available with the TalkAndWrite plug-in for Skype.

The installation process of the TalkAndWrite plug-in for Skype is simple and easy. The process of setting up a collaborative session is as well very simple and intuitive as it is very similar to doing an actual Skype call. As a very minor flaw it has to be remarked that there is a lack of an explicit notification if one’s partner going online or offline. A more important flaw is that there seem to be some issues with the stability of the TalkAndWrite plug-in for Skype. While doing the evaluation two crashes of the tool happened. Further on, some troubles regarding the real time probability were observed. Twice it happened that one partner could not use the whiteboard anymore without any specific reason.

This is especially annoying due to the fact that the session management supported by the TalkAndWrite plug-in for Skype suffers from major limitations. It is only possible to edit and save one single opened file. There is no automatic storage process during the whole session nor anything like a storage request at the end of a session. The only automatic storage request appears in the case of a shutdown of the complete tool. In the case that one partner opens a new file, through synchronisation the same file will be opened at the partner's screen. There is neither a notification nor a possibility of disagreement for the other partner. As soon as one partner changes the file or its content, the other partner can not avoid having the same action replicated within his local application. Further on, there is no explicit notification in case that one partner goes offline neither if willing nor if not. The worked on artefact still remains open and can be changed or edited at will by the remaining partner. It needs to be remarked that not all changes done to an artefact can be undone. To sum it up, files of a session have to be stored manually and locally.

Available communication channels include a whiteboard, a chat, Internet telephony and video conferencing support whereby besides the whiteboard all features are provided via Skype and therefore automatically available when using the TalkAndWrite plug-in as well.

The available documentation is somewhat brief but was found to be sufficient. Although the usage is very intuitive, during the evaluation some features were only detected while reading the manual.

There are some special editions of the TalkAndWrite plug-in for Skype available for teachers, doctors, lawyers as well as engineer and architects. These were not included in the evaluation.

In conclusion it can be stated that the evaluation process was quite interesting. It was quite surprising how supportive the TalkAndWrite plug-in for Skype was found to be as a tool for the Distributed Pair Programming practice, though certainly some drawbacks regarding this usage were observed. Its major strength is definitely its combination with the powerful features of Skype itself. So, though not going to be the first choice, the TalkAndWrite plug-in for Skype is one of the tools that showed quite good performance in a Distributed Pair Programming environment.

### 6.3.11 Virtual Network Computing (VNC)

Virtual Network Computing, better known under its abbreviation VNC, is a stereotypical screen-sharing application that comes in a number of implementations and is available for Microsoft Windows as well as UNIX and Linux operating systems. Adaptations of implementations for the MacOS operating systems are currently ongoing. In scope of the evaluation the most prominent implementation of VNC, called RealVNC, was used.

Being the stereotypical screen-sharing application it is, VNC's only tiny lack with respect to its support of screen-sharing is that the screen size of the replicated screen does not adjust to the screen respectively window size of the viewer making it sometimes necessary to scroll on screens with lower resolution. As all the actions of any of the users within a session can be viewed by all others in the session the sense of workspace awareness is fully supported as well.

There is no second pointer for the Navigator in the standard VNC. There exists an extension to VNC by Hanks [Hanks, December 2005] that provides a second pointer specifically for the appliance of VNC in Distributed Pair Programming environments. Unfortunately this extended VNC version could not be included in the evaluation as the operating system required for the VNC server part was not available to the evaluation team.

There is no floor control mechanism implemented in VNC as well. Any user can use all the input devices at any time. Still there is some rudimentary support of a kind of social floor control as it is quite obvious if the remote partner grabs the pointer and thus takes over control respectively the Driver’s role.

As VNC is a basic screen-sharing tool, no communication channels for interaction between collaborating team members are supported. Therefore any communication channels needed are to be substituted by the use of additional tools.

Installation of VNC is straight forward and easy. To start the tool and establish a collaborative session either on one of the users’ computers or on an additional server the VNC server software needs to be started. After this the users acting as clients need to start their VNC viewer software and connect to the VNC server. Successful connection to the server changes the colour of the VNC icon on the VNC server.

Usage of the tool is very intuitive and easy as it simply replicates the servers screen onto the client’s viewer application. Collaboration in scope of Distributed Pair Programming is really easy by using VNC. Realtime probability and screen actuality as well as stability were not an issue during evaluation of VNC.

Obviously when using VNC for the support of Distributed Pair Programming an additional editor or IDE is needed to do the actual collaborative work on an artefact and the need for additional tools to provide the needed communication channels was already discussed. Nevertheless VNC is quite good and useful for the support of the Distributed Pair Programming practice.

### 6.3.12 XPairtise

XPairtise is the second plug-in to the well established Eclipse IDE specifically designed to support collaboration in scope of Distributed Pair Programming included within this evaluation. It is kind of a successor to the TUKAN system and supports a wide range of features desirable by a tool supporting the Distributed Pair Programming practice like floor control, communication via chat and whiteboard and so on. The design of the plug-in reflects a client-server-architecture. It is written in Java and thus available on all platforms for which Eclipse is available.

Support of collaboration awareness as well as workspace awareness is great with XPairtise. Both partners of a Distributed Pair Programming team have the same view on the worked on artefact as well as on the development environment whereas only the Driver is authorized to create and edit files.

Although specifically developed for Distributed Pair Programming XPairtise still lacks the availability of a second pointer that can be used by the Navigator, respectively any pointer replication at all. Thus the only possibility to give deictic references is by highlighting certain sections of the artefact which works quite well but still is not the same as using a pointing device that is visible to one’s partner as well. Driver and Navigator get differing colours assigned for this highlighting.

XPairtise has a formal kind of a floor control mechanism implemented. There is a strict distinction between the roles of Driver and Navigator. Only the user in the role of the Driver is authorized to provide input to the artefact collaboratively worked on. The role impersonated by the local user by a changing icon in the status line at the bottom of the XPairtise window. Each user joining an ongoing session can select his initial role upon joining the session though only one Driver is possible at any given time. Beside Driver and Navigator there is the additional role of spectator available within XPairtise. The role of spectator is reserved for users only watching or moderating the other users' collaborative activities. The request to exchange the roles of Driver and Navigator is done by pressing a button. A flashing icon on the other partners screen signals the request of the role change. This can be accepted by the other partner pressing the same button.

As XPairtise is just a plug-in to the Eclipse IDE, Eclipse is needed to be able to install and work with XPairtise. Installation is then as simple as copying the XPairtise jar-file into the plug-ins directory of the Eclipse installation. The XPairtise plug-in is activated after starting Eclipse by clicking the XPairtise button in the Eclipse GUI. To establish a session it is first necessary to connect to an XPairtise server. An account is needed for this but can be created easily directly from within XPairtise as well. As soon as the connection to the server is established any session of users connected to this server can be joined.

The user interface of the XPairtise plug-in to the Eclipse IDE is simple, intuitive and well structured and it is easy to find one's way in the predefined menu- and view-structure. XPairtise somehow does the miracle of packing a number of things like a chat-window, a whiteboard-window, user and session galleries, the actual editor, a list of projects and files as well as the state of the connection to the server and the current role of the local user into one screen and still staying well arranged. As being actually only a plug-in the GUI is seamlessly integrated in Eclipse's GUI. To make it possible to distinct between Eclipse standard views and XPairtise views the later are prefixed with "XP" within their title.

There is no support of complex session management within XPairtise though its certainly possible to store changed files locally and continue work later on.

Screen actuality and the impression of XPairtise's stability were good during the evaluation.

The only communication channel needed not supported by XPairtise itself is the speech channel which can be easily substituted with nowadays commonly available tools. Nor is a video channel supported but this was not missed during the evaluation. Within XPairtise there are two separate textual chat channels available. One being a global chat channel which can be used to chat with all users connected to the same server and one being used for chatting within the collaborative session. Though the global chat might not be needed for the support of Distributed Pair Programming it is certainly a useful asset if you need information from a larger community. The whiteboard provided by XPairtise is a simple but sufficient one that allows also drawing and storing of any outlines done.

Concluding XPairtise was found to be the most supportive tool for the Distributed Pair Programming practice evaluated in scope of this diploma thesis. It is a full fledged and quite usable plug-in to the established Eclipse IDE and therefore supports a number of operating system platforms as well as programming languages. It definitely is the first choice from the tools evaluated out of the group of collaboration-aware applications.

## 6.4 Comparison of evaluated Tools

In chapter 6.3 the strengths and weaknesses of each of the evaluated tools were discussed individually. In this chapter the focus now lies on comparing the tools evaluated with respect to their usability for the support of the Distributed Pair Programming practice.

As already discussed due to the major differences in the basic approach to support non-collocated collaborative activities it is difficult up to impossible to directly compare screen-sharing applications with collaboration-aware applications. Thus the evaluated screen-sharing applications will be compared among themselves as will be the evaluated collaboration-aware applications. Finally a comparison of the findings of the two categories will be done.

### 6.4.1 Comparison of evaluated Screen-Sharing Applications

Out of the category of screen-sharing applications two tools were evaluated in scope of the evaluation of this diploma thesis. RealVNC was evaluated as a candidate for such tools that provide nothing but the basic screen-sharing functionality and Microsoft’s NetMeeting as a candidate for such tools that provide a rich environment for non-collocated collaboration besides the actual screen-sharing functionality.

| | **Basic Properties** | **Support of Gesturing** | **Floor Control Support** | **Communication Possibilities** | **Platform Independence** | **Usability** | **Documentation** |
|---|---|---|---|---|---|---|---|
| Mircosoft NetMeeting | ☺ | ☹ | 😐 | ☺ | ☹ | ☺ | ☺ |
| Virtual Network Computing (VNC) | ☺ | 😐 | ☹ | ☹ | 😐 | ☺ | 😐 |

*Table 13. Overview of evaluated Screen-Sharing Applications*

The basic criterion for a screen-sharing application is certainly the support of actually the screen-sharing functionality. This is quite good with both of the evaluated tools though neither tool provides a functionality to adapt the size of the replicated screen to the local screen resolution. In cases where the collaborating partners screen resolutions differ this might impact the collaborative activity as well as the workspace awareness as in the case that the partner with the lower screen resolution acts as the client in the build up session he is not able to see the complete screen of the other partner. Both tools are quite good at supporting workspace awareness though VNC being a basic screen-sharing application was found in the evaluation to be a little better there as it does not distract its users by all the other windows that might be open in a NetMeeting session.

Neither NetMeeting nor VNC comes with a second pointer that can be used by the Navigator though in both applications it is possible for the Navigator to use the single cursor available. This is obviously a result of the fact that neither of the two applications is developed specifically for the purpose of supporting the Distributed Pair Programming practice. As already discussed there exists an extension to VNC enhanced for the actual support of Distributed Pair Programming by Hanks [Hanks, December 2005] which could unfortunately not be included in the evaluation. There are some minimal semantic adaptations of the single pointer available with NetMeeting. The support of deictic referencing via highlighting sections of the artefact collaboratively worked on works better in VNC though VNC has some problems with smoothing out the telepointer jitter. In general VNC provides a better support of gesturing especially due to the more practical support of deictic referencing via highlighting. In the case of NetMeeting a formal role change is necessary for this issue.

When it comes to the support of floor control, NetMeeting is in the lead as it implements kind of a heavily formal and moderated floor control mechanism whereas VNC does not provide any mechanism of floor control at all though it makes it quite easy to exchange roles as the Navigator that wants to exchange roles simple takes over control of the input devices by using his local devices. The coordination and role distribution in VNC completely relies on social interaction via any communication channels between the two collaborating partners. On the other hand it must be stated that the extremely formal floor control mechanism of NetMeeting makes role exchanges quite disturbing to the flow of the collaborative activity.

Being a tool with a rich environment for non-collocated collaboration NetMeeting supports a number of communication channels that can easily be used to handle the necessary communication between the collaborating Distributed Pair Programming partners. Speech, textual chat and whiteboard as well as a video channel are supported by NetMeeting which thus provides a quite good environment for communication between the two partners. VNC on the other hand is a tool with only basic screen-sharing functionality and thus does not provide any communication channels at all.

NetMeeting is a tool developed by Microsoft and shipped as part of the Microsoft Windows operating system till its version XP/2003 and can be installed as an add-on to later Windows operating system versions like Vista. Therefore it is obviously only available for Microsoft's Windows operating system whereas VNC is available for a wide range of platforms ranging from Microsoft's Windows to Unix and Linux and there is even am implementation of VNC for the MacOS in development.

With respect to usability both tools, VNC and NetMeeting, fulfil the critical properties of screen actuality and easiness of collaboration equally well. VNC is better with respect to the intuitivity of the tool's usage as well as with the easiness to install and start the tool and establish a connection to a remote partner. Still, as discussed there is the drawback of VNC that some other tools are needed to provide the needed communication channels besides the additional IDE or editor needed with both tools to do the actual development work. The documentation of Microsoft's NetMeeting is significantly better than that of VNC which is obvious as most of the current implementations of VNC are developed in scope of open source projects which often lack high quality documentation whereas as discussed NetMeeting was developed as part of former versions of Microsoft's Windows operating system.

When it comes to the decision which of these two tools is better suited for the support of the Distributed Pair Programming practice this depends on what is rather expected by a screen-sharing tool used for this purpose. If a rich environment supporting a number of communication channels and a, though extremely formal, technical implementation of a floor control mechanism is needed and the development is to be done solely on platforms sporting Microsoft's Windows operating system the decision is to be in favour of Microsoft's NetMeeting. If the collaborating partners rather ask for a simple environment available for multiple platforms and therefore accept a highly social and communication intense way of managing floor control and have additional tools at hand that provide the needed means of communication with each other they will rather take any of the VNC implementations. In conclusion both tools evaluated are good at supporting the application of the Distributed Pair Programming practice, especially when taken into mind that neither of the two tools is specifically designed for such a purpose and therefore both tools naturally lack some features one would find supportive when doing Distributed Pair Programming.

### 6.4.2 Comparison of evaluated Collaboration-Aware Applications

A number of collaboration-aware tools that come from almost all found sub-categories of such tools were included in the evaluation done in scope of this diploma thesis. For the sub-category of collaborative editors ACE, Gobby, GrewpEdit, MoonEdit as well as the TalkAndWrite plug-in for Skype and COPPER as a representative of collaborative editors specifically designed for the practice of Distributed Pair Programming were evaluated. PEP, also called Pair Eclipse Programming, and XPairtise, were selected to be included in the evaluation as representatives of plug-ins to the Eclipse IDE specifically designed for the support of the Distributed Pair Programming practice as well as DocSynch in its implementation for the jEdit IDE as well the collaboration enhancement to the NetBeans IDE, the NetBeans collaboration project also known as collab.NetBeans, which are designed for non-collocated collaboration though not specifically for support of the Distributed Pair Programming practice. Unfortunately none of the tools or tooling environments taking a more exotic approach to the support of the Distributed Pair Programming practice could be evaluated as they were either not available or had requirements on additional resources needed to apply them that could not be fulfilled by the evaluation team.

Most of the tools evaluated provided adequate support of collaboration awareness with the exception of MoonEdit which fails to display important information to provide a sense of collaboration awareness on its GUI. ACE and PEP are a little worse with respect to this property than the rest of the tools mainly due to major lacks in the support of workspace awareness that affect the collaboration awareness as well whereas XPairtise is a little better than the rest due to its extensive and well structured GUI. The support of the up-to-the-moment understanding of another user's interaction with the shared workspace is quite divergent among the evaluated tools. The best performance in this property was done by Gobby, the TalkAndWrite plug-in for Skype and XPairtise whereas ACE, collab.NetBeans and PEP showed significant lacks in this area.

| | Basic Properties | Support of Gesturing | Floor Control Support | Communication Possibilities | Platform Independence | Usability | Documentation |
|---|---|---|---|---|---|---|---|
| ACE | 😐 | ☹ | ☹ | ☹ | 😐 | ☺ | ☺ |
| The NetBeans Collaboration Project (collab.NetBeans) | 😐 | ☹ | ☹ | ☹ | ☺ | 😐 | ☺ |
| COPPER | ☺ | ☹ | ☺ | ☹ | ☺ | ☺ | ☹ |
| DocSynch | ☺ | ☹ | 😐 | ☹ | ☺ | 😐 | 😐 |
| Gobby | ☺ | ☹ | ☹ | ☹ | ☺ | ☺ | 😐 |
| GrewpEdit | ☺ | ☹ | ☹ | ☹ | ☺ | ☺ | ☹ |
| MoonEdit | 😐 | ☹ | ☹ | ☹ | ☺ | ☺ | ☺ |
| PEP - Pair Eclipse Programming | 😐 | ☹ | ☺ | ☹ | ☺ | ☺ | 😐 |
| TalkAndWrite plug-in for Skype | ☺ | ☺ | ☹ | ☺ | ☹ | ☺ | ☺ |
| XPairtise | ☺ | ☹ | ☺ | 😐 | ☺ | ☺ | 😐 |

*Table 14. Overview of evaluated Collaboration-Aware Applications*

Of all the collaboration-aware tools evaluated only the TalkAndWrite plug-in for Skype supports a second cursor that can be used by the Navigator for providing deictic references. Interestingly the TalkAndWrite plug-in for Skype is not specifically designed for the support of the Distributed Pair Programming practice whereas a number of the other tools evaluated and not providing this ability to the user acting as Navigator or not replicating pointers at all are. This is quite remarkable as the availability of a second pointer for the Navigator to provide deictic references in a similar way as in the collocated Pair Programming environment is mentioned to be an important requirement on environments supporting the Distributed Pair Programming practice throughout most of the literature about this topic. Most of the evaluated tools at least give a minimum support of deictic referencing by enabling and replicating the highlighting of certain sections of the artefact worked on collaboratively with the exceptions of DocSynch and collab.NetBeans which do not provide any support for deictic referencing at all.

The role concept of Distributed Pair Programming along with the implementation of a floor control mechanism to support this concept is certainly implemented in all the evaluated tools specifically designed for the support of the Distributed Pair Programming practice which are COPPER, DocSynch, PEP and XPairtise. All four tools implement a formal approach to the support of floor control where one of the users can request to exchange the roles and the other has to accept or decline this request or where the floor control is even master user mediated. In PEP a more social approach is additionally possible where the Navigator can simply take over control which leads to some confusion about who is in which role at any given time and also corrupts the tools floor control mechanism. This might actually be a fault and due to the fact that PEP is still in a quite early development stage. Though the master user mediated approach of DocSynch does some impact to the process of Pair Programming there is no really disturbing impact with any of these four tools. Besides these tools specifically designed for the support of Distributed Pair Programming there is also some minimal floor control support available with collab.NetBeans where the local section of the artefact is locked for any further input while one of the users is typing. Still this is actually no fully fledged floor control implementation as any other user is fully granted input permission in other sections of the artefact. The other evaluated tools lack any support of floor control.

During the evaluation it was found that the tools significantly lack to support communication via speech though the traditional Pair Programming activity is a highly communicative activity and the support of a speech channel is therefore mentioned as an important required property in most of the related literature. Still only COPPER and the TalkAndWrite plug-in for Skype support a speech channel. The speech channel supported by COPPER could not be tested in scope of the evaluation due to the mentioned problems in setting up the WebDAV server, while the speech channel support of the TalkAndWrite plug-in for Skype results from the basic functionality of Skype itself which is Internet telephony. Textual communication via either textual chat or whiteboard is supported by all of the tools except MoonEdit with differing qualities. In general the whiteboard support lacks features that would be available in a collocated environment. The video channel, which is not seen as that important in the related literature, is only supported by the TalkAndWrite plug-in for Skype whereas DocSynch and XPairtise provide some sort of support to find a partner for a Distributed Pair Programming session.

With respect to supported platforms all of the evaluated tools are available for Microsoft's Windows operating systems. Most tools, with the exception of the TalkAndWrite plug-in for Skype are also available for Linux or UNIX operating systems and for the MacOS operating system all evaluated tools but the TalkAndWrite plug-in for Skype and ACE are available.

All of the tools evaluated were found to be quite intuitive to be used. The installation and configuration of the tools is quite easy for most of the evaluated tools though it gets a bit tricky with COPPER due to its Spanish installation documentation and with DocSynch and XPairtise due to lack of corresponding documentation. The session establishment made some problems in GrewpEdit and especially in PEP due to the fact that there always only the second attempt to establish a session worked. As discussed, there was no load and stress and testing done during the evaluation. Therefore the evaluation of the stability is only a hint according to the experiences observed during the evaluation. From this there seem to be obvious stability problems with PEP and the TalkAndWrite plug-in for Skype and even more severe ones with ACE and Gobby. Screen actuality and refresh rate were good for most of the tools except for DocSynch were significant delays were encountered and collab.NetBeans where the update of the Navigator’s screen always only happened after the Driver stopped typing for a while. The collaborative work was felt to be easiest with Gobby, GrewpEdit, MoonEdit and XPairtise followed by COPPER. Some significant lacks with respect to the easiness of the collaboration were observed in the other tools. Still it was possible to collaborate with all the tools in a sufficiently effective way. Syntax highlighting, as was expected is only supported by those tools that enhance some established IDE that already supports this feature itself. Therefore those tools supporting syntax highlighting are collab.NetBeans, PEP and XPairtise. The same is valid for configuration management support which is also available for these three tools via the actual IDEs they are plug-ins for. Certainly it is possible to store changed artefacts with all of the tools evaluated, but only COPPER and MoonEdit provide some means of session management. In the case of COPPER this could not be tested during the evaluation due to the mentioned problems with installing the WebDAV server. In the case of MoonEdit any changes done to any artefact are instantly stored on the central server.

The quality of the available documentation tends to be higher for commercial products and lower for open source products though there are exceptions to both cases.

Summarizing the quantified and weighted evaluation shows that the Eclipse plug-in XPairtise is the best choice for the support of the Distributed Pair Programming practice out of the tools evaluated, followed by COPPER which is as well specifically designed for this issue as a collaborative editor. Surprisingly the next best choice is the TalkAndWrite plug-in for Skype which was able to score with its support of a second pointer and the wide range of communication channels supported via Skype itself. PEP would be a good candidate as well as soon as a more mature version of this plug-in to the Eclipse IDE will be available. All of the other tools are certainly useable for the support of the Distributed Pair Programming practice but still lack some important properties that are available with the four already mentioned and thus are not recommended to be used in the appliance of the Distributed Pair Programming practice.

### 6.4.3 Comparison of the two Tool Categories

As already discussed, a comparison between the different tool categories of screen-sharing applications and collaboration-aware applications is difficult due to the completely divergent approach on how collaborative work of non-collocated users is supported. Both tool categories have some advantages and disadvantages.

The support of workspace awareness tends to be a little better in screen-sharing applications as in those applications it is clear and visual to at least the Navigator what the Driver is doing at any given time. As most of the evaluated collaboration-aware tools lack a second pointer and even lack a replication of any pointer at all, here the screen-sharing applications have some advancement as at least the single pointer is visible and depending on the floor control mechanism also usable by both of the partners.

As discussed there are such tools implementing a floor control mechanism and such that rather rely on a user mediated social floor control in both tool categories, so there is actually none of the two categories preferred to the other. The same is valid for the support of communication channels and the usability of the evaluated tools though it needs to be kept in mind that with screen-sharing applications it is obviously necessary to have an additional editor or IDE to do the actual development work.

Therefore the decision whether a screen-sharing application or a collaboration-aware application is selected to support the appliance of the Distributed Pair Programming practice in a concrete development project is more or less a matter of taste of the collaborating partners. Out of the evaluation no concrete hint with respect to this decision can be given.

# 7 Discussion

Based on the results of the work done in scope of this diploma thesis, this chapter will give a retrospection of the research questions defined in chapter 3.1. It will discuss to which extent the work done in scope of this diploma thesis succeeded in providing answers to these questions. The answers found to the research questions will be discussed once more.

## 7.1 Available Tools for the Support of Distributed Pair Programming

The research done in scope of this diploma thesis provided a number of currently available tools that are good candidates for tool support for the application of the Distributed Pair Programming practice. It was found that beside the already mentioned categorization into screen-sharing and collaboration-aware applications a further sub-categorization makes sense to be able to group the tools found into such that share major similarities. The following figure which was already shown in chapter 4 once more shows the categorization of the tools found.

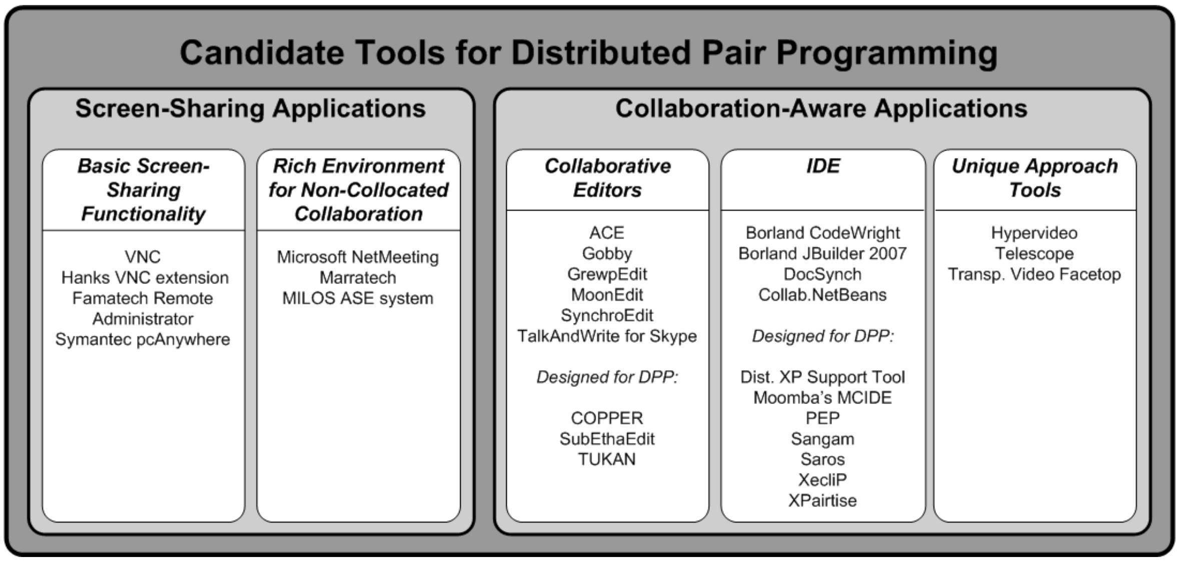


*Figure 33. Available Tools for the Support of Distributed Pair Programming*

Details about the available tools for the support of the Distributed Pair Programming practice and their categorization into the various sub-categories are discussed in chapter 4.

To recapitulate there are a number of research as well as commercial and open source software tools available that have potential to be usable for the support of Distributed Pair Programming. While the research and open source tools are rather specifically designed for the support of Distributed Pair Programming in particular, the commercial tools available rather tend to provide means for non-collocated computer supported collaborative authoring of artefacts and are thus potentially usable for the support of Distributed Pair Programming as well though lacking support of some issues specific to the Pair Programming practice like providing means of a floor control mechanism or providing a second pointer visible to both of the Pair Programming pair for the support of deictic referencing by the Navigator. Interestingly with Hypervideo, Transparent Video Facetop and

Telescope there is also a small number of tools available that takes a rather bohemian approach to the support of Distributed Pair Programming.

Although it would be presumptuous to claim that the candidate tools for the support of the Distributed Pair Programming practice mentioned in this diploma thesis are all the software tools currently available, the presented list is nevertheless the most extensive one that is currently available and thus provides a good starting point for the issue of tool support for Distributed Pair Programming.

## 7.2 Eligible Properties of Tools supporting the Distributed Pair Programming Practice

Though previous work already mentions a number of desirable properties of tools supporting the Distributed Pair Programming practice, a consolidated and comprehensive list of such properties was missing up to now. Chapter 5.1 describes the results of the work done in scope of this diploma thesis with respect to establish such a list. The requirements found were categorized in scope of setting up the generic evaluation framework for tools supporting the Distributed Pair Programming practice.

Basic properties are such properties crucial to enable the work in a Distributed Pair Programming team. The support of screen-sharing for screen-sharing applications and the sense of collaboration awareness for collaboration-aware applications are immanent to these two major categories of tools. The support of workspace awareness which is the basis for efficient collaboration and communication in non-collocated environments also belongs among these properties.

Another category cumulates all properties with respect to the support of gesturing for the Driver as well as the Navigator and the degree to which deictic referencing is supported by the tooling. As the support of deictic referencing is mentioned as being of utter importance for the support of non-collocated collaboration in Pair Programming teams this is an important category of properties. The same is valid for the support of floor control mechanisms as this is needed to implement the role concept of Driver and Navigator immanent to Pair Programming and the exchange of these roles between the collaborating partners of a Distributed Pair Programming team.

As was already discussed the practice of Pair Programming is an extremely communicative activity. This certainly needs to be supported by software tools that support Distributed Pair Programming. Various communication channels are needed in scope of appliance of the Distributed Pair Programming practice. Among those is certainly first and foremost the support of means that enable the collaborating partners to speak with each other as if they were sitting side by side.

As Distributed Pair Programming is a way of distributed development it might as well be that the collaborating partners use different operating system platforms on their respective machines. Thus platform independence is an important issue for tools supporting the application of the Distributed Pair Programming practice.

As with all software products requirements on usability are fundamental for tools supporting the practice of Distributed Pair Programming. Especially as lacking usability of a tool used for the support of Distributed Pair Programming would impact the easiness of collaboration of the pair programming team and might influence the support of workspace awareness as well usability requirements are crucial for application support of Distributed Pair Programming. The same is valid for the quality of the installation and user manual of an applications.

Besides the above mentioned property categories there is the category of tool immanent properties which subsumes not so much properties eligible to be fulfilled by a software tool that is to be used in scope of the Distributed Pair Programming practice but rather such properties of the tool that provide information about the tool itself like whether it is off-the-shelf, a special purpose tool, a viewer and so on.

Not all of the properties are mandatory to be fulfilled by a tool to make it usable in the appliance of Distributed Pair Programming practice. Quite a number of them are rather optional or an additional asset to the usability of the tool in scope of the support of the Distributed Pair Programming practice if available.

## 7.3 Comparative Evaluation of Tools supporting the Distributed Pair Programming Practice

A reasonable approach to provide means for a comparable evaluation of a number of tools for a certain purpose is to set up an evaluation framework that provides means to evaluate tools against certain requirements or properties in a comparative way. The same was done in scope of this diploma thesis to provide a comparable way of evaluating candidate tools for the support in the appliance of the Distributed Pair Programming practice. The set up of the generic evaluation framework for tools supporting the Distributed Pair Programming practice is described in chapter 5.2 of this diploma thesis.

As a basis for the setup of the evaluation framework the abstract model to select software development tools as defined by Zwartjes and van Geffen [Zwartjes and van Geffen, 2005] was used. The model described by Zwartjes and van Geffen consists of a tool matrix that is built up by executing five steps. Though partly some adaptations were needed, the same steps were executed in scope of the work on this diploma thesis to populate the framework based on the found eligible requirements. Then the set up generic evaluation framework for tools supporting the Distributed Pair Programming practice was used to evaluate a subset of the tools found in scope of the research done for this diploma thesis.

## 7.4 Availability of Support of eligible Properties of Tools supporting Distributed Pair Programming

Besides the actual results of the evaluation of a subset of the tools found during the research activities in scope of this diploma thesis to have potential to support the appliance of the Distributed Pair Programming practice which were discussed in chapter 6.3, the result of the evaluation of these tools with the generic evaluation framework for tools supporting the Distributed Pair Programming practice also provides some input to a general discussion on which properties desired to be fulfilled by a tool that is used to support the Distributed Pair Programming activity are supported by today’s tools and which not.

### 7.4.1 Basic Properties

One might think that the support of screen-sharing for screen-sharing applications as well as the creation of a sense of collaboration awareness for collaboration-aware applications is the most basic issue of such applications. This was found to be the case for the evaluated screen-sharing applications which all supported screen-sharing in a reasonable way. The same is not true for collaboration-aware applications. Though most succeeded in providing a sense of collaboration awareness a select few of those tools only showed mediocre support of this property.

As discussed workspace awareness is the basis for efficient collaboration and communication in non-collocated environments because the groupware users know what each other can see. Most of the evaluated tools provided satisfying support of workspace awareness though some few where mediocre in their support of workspace awareness at best.

Hanks' [Hanks, December 2005] observation that the provisioning of information supporting workspace awareness is of particular importance when using tools that make it possible for the collaborating team members to view different sections of the artefact they are working on respectively the shared workspace was found to be completely valid during the evaluation. All the tools that made viewing different sections of the shared artefact or workspace possible and not providing any information about the remote partners current focus suffered from dimmed workspace awareness. A very nice feature that was found during the evaluation for the support of workspace awareness is the radar view provided in COPPER.

The evaluation showed that workspace awareness is not so much an issue in screen-sharing applications as it is in collaboration-aware applications as in the screen-sharing case anyway the same screen is visible to the collaborating Pair Programming partners and thus it is easier for them to stay focused and aware of what the partner is doing at any given time. This observation was expected as the same was observed by Hanks [Hanks, December 2005] in his studies.

All this said it is nevertheless obvious that the sense of workspace awareness experienced in a Distributed Pair Programming environment is not the same as the one in the collocated case.

### 7.4.2 Support of Gesturing

When looking over the results of the evaluation done in scope of this diploma thesis, the most obvious finding is that there is a significant lack in the support of gesturing and giving deictic references. This is especially significant due to the fact that the support of gesturing and deictic references is mentioned as one of the key success factors to enable a performance in Distributed Pair Programming which is comparable to the performance in collocated Pair Programming as well as in many other non-collocated collaborative activities in the corresponding literature [Tang, February 1991], [Cox and Greenberg, 2000], [Hanks, December 2005]. In the tool category of collaboration-aware tools almost none of the evaluated tools with the exception of the TalkAndWrite plug-in for Skype provide a second pointer that can be used or is specifically designed for the usage by the Navigator nor any replication of pointers between the partners at all, though some at least replicate the positions of the cursors within the text of the artefact. Being aware of the findings in prior research with respect to Pair Programming in collocated and non-collocated environments and the importance of deictic references discussed there this is especially surprising as even those tools evaluated that are specifically designed for the support of Distributed Pair Programming lack a second pointer for the Navigator as well as any pointer replication between the two users at all. The same lack is observed in the evaluated screen-sharing applications as well though it is not that kind of critical there as at least the one single available pointer is visible to and can be used by both partners of the Distributed Pair Programming pair.

The research done in scope of this diploma thesis showed that there are some few tools that provide a second pointer for the Navigator or a similar possibility for both partners of the Distributed Pair Programming pair to give deictic references. Among those tools found is Hanks' enhancement of the screen-sharing application VNC [Hanks, December 2005] that adds a second telepointer for the Navigator's use in the form of a red hand with a pointing index finger along with the semantic adaptation that this pointer is only visible if the Navigator activates it by entering a gesturing mode. Another tool found is the Transparent Video Facetop system developed by Stotts et al. [Stotts, Smith and Gyllstrom, 2004] which uses webcams to record the images and gestures of the collaborating members of the Distributed Pair Programming pair and overlays this on each of the partners' screens with a screen-sharing application used to provide the common workspace. As comes to mind both of this tools are developed in scope of research activities and are not readily available on the market to be used in industrial applications of the Distributed Pair Programming practice. No collaboration-aware tools were found in the evaluation to provide a similar support of deictic referencing except, as already mentioned, the TalkAndWrite plug-in for Skype. Though it has to be mentioned that most of the evaluated collaboration-aware tools at least provide a rudimentary and somewhat unintuitive approach to support a minimum possibility of giving some deictic references by replicating sections of the artefact worked on highlighted by one user to the other user in a distinct colour assigned to the user who did the highlighting.

The lack of the support of gesturing in many of the available tools that became obvious during of the evaluation work done in scope of this diploma thesis clearly shows that there is one of the key success factors for Distributed Pair Programming not fulfilled by many of the readily available tools to support this practice. Thus collaborating Pair Programming team members that work in a non-collocated environment suffer some restrictions in the way they can collaborate compared to collocated Pair Programming team members.

### 7.4.3 Floor Control Support

In contradiction to the findings done with respect to the support of gesturing concerning the floor control support the evaluation shows that those tools that are specifically designed for the support of the Distributed Pair Programming practice support the Distributed Pair Programming role concept as well as a mechanism for the exchange of the roles between Driver and Navigator. None of the collaboration-aware tools not specifically designed for the support of the Distributed Pair Programming practice provide floor control functionality although most of them are obviously designed for non-collocated collaboration. This is a bit surprising as it comes to mind that the support of floor control would make sense as well in other non-collocated collaborative activities like the review of code or documents, the collaborative editing of documents and so on. Nevertheless it is not in scope of this diploma thesis to discuss the shortcomings of any tools in other environments than Distributed Pair Programming. Obviously a floor control mechanism is not supported in basic screen-sharing tools like VNC. Microsoft's NetMeeting, being actually designed for the support of distributed meetings, on the other hand supports such a mechanism as it would not make sense that all participants of such a meeting are able to edit the shared screen all the time.

In all of the evaluated tools supporting floor control the implementation of the role exchange mechanism is quite formal. It is assumed that this is due to the fact that such a formal implementation of a floor control concept is easier to implement into a tooling. Though Hanks [Hanks, December 2005] stated that an application-mediated and therefore formal floor control mechanism that requires the users to take physical action such as pressing a button or selecting a menu item would potentially impact the flow of the Pair Programming process which was as well support by the studies of Greenberg [Greenberg, 1990] such an impact was not observed during the evaluation activities with the tools that implement such a mechanism.

In general the floor control mechanism implementations in the tools specifically designed for the support of the Distributed Pair Programming practice were found to be quite suitable and did not significantly hamper the process of Pair Programming during the evaluation.

### 7.4.4 Communication Channels

Williams and Kessler [Williams and Kessler, 2002] stated that collocated Pair Programming is a highly conversation-intensive task. The importance of the availability of a speech communication channel in Distributed Pair Programming teams was also observed by Schummer and Schummer [Schummer and Schummer, 2001] as well as Canfora et al. [Canfora, Cimitile and Visaggio, 2003] which found that means of textual communication like chat and whiteboard, though being highly supportive in a non-collocated collaborative environment, are no sufficient alternative to a voice channel. On the other hand the support of a video channel was not found to increase the effectiveness in Distributed Pair Programming in previous research [Hanks, December 2005], [Baheti, Gehringer and Stotts, 2002], [Olson and Olson, 2000], [Schummer and Schummer, 2001].

All of these observations are supported by the experiences made during the tool evaluations done in scope of this diploma thesis. The more surprising it was to find that on the one hand almost none of the evaluated tools support communication via voice between the collaborating team members and on the other hand most of the evaluated tools provide possibilities for textual communication between the collaborating partners instead. Of course it is quite simple to substitute the voice channel by using either traditional teleconferences or Internet telephony as supported by Skype. Still it is a bit strange that so many of the tools evaluated along with almost all of them that are specifically designed for the support of the Distributed Pair Programming practice except for COPPER provide no support for voice communication between the collaborating users.

Concluding it needs to be stated that here is some potential for improvement as it would make the set up of a Distributed Pair Programming session easier and faster if the needed communication channels were provided by the used tool itself.

### 7.4.5 Platform Independence

As discussed Distributed Pair Programming is a way of distributed development. This means that the members of the collaborating pair might have different operating system platforms on their respective machines. Thus it would be a significant asset if tools used in scope of Distributed Pair Programming would be available on multiple operating system platforms.

When looking on the results of the evaluation done in scope of this diploma thesis though the evaluated screen-sharing applications might lack the support of certain operating system platforms most of the evaluated collaboration-aware applications are available for most common operating system platforms. Due to the fact that the latest screen-sharing applications tend to be web based applications that can be accessed or viewed by using multiple browsers like being the case for Marratech, the issue of Platform Independence is also taken care of for screen-sharing applications. Thus the issue of Platform Independence is sufficiently fulfilled with the currently available tools.

### 7.4.6 Usability and Documentation

As discussed requirements on the usability and documentation of software tools are not specific to the application of tools in scope of Distributed Pair Programming but are rather fundamental with all software products. Nevertheless a reasonable usability of the used tool is still crucial for the effectiveness of Distributed Pair Programming as a tool with shortcomings with respect to its usability would impact the collaborative activity when using such a tool.

In general it can be said that the evaluated tools showed sufficient usability to enable a collaborating pair to perform effective Pair Programming in a distributed environment, though some of the tools showed some shortcomings in selected properties in scope of the usability category. Though obviously only the dedicated IDEs supported syntax highlighting this can not be expected from collaborative editors. Interestingly though Pair Programming in the distributed as well as collocated way is actually kind of a session based activity, meaning that the pair joins for a Pair Programming session and might continue their collaborative work in a later session even the tools specifically designed for the support of Distributed Pair Programming lacked the support of a considerable session management. Obviously all evaluated tools provide means to store and reopen certain artefacts but the storage and restoring of complete sessions along with all open artefacts, chat windows, and so on is only supported by very few of the tools according to their documentation and unfortunately could not be evaluated in scope of the evaluation.

So there is quite some potential to improve the usability of tools for the support of Distributed Pair Programming as a full-fledged session management implementation would make the reestablishment of an interrupted Distributed Pair Programming session significantly easier.

# 8 Conclusion and Future Work

In scope of this diploma thesis an extended research for software tools with potential to support the Distributed Pair Programming practice was done. Based on results from previous research a consolidated list of required and desired properties of tools to enable effective and efficient work in scope of a Distributed Pair Programming activity was set up and extended with own findings. This list was then used to set up a generic evaluation framework for tools supporting the Distributed Pair Programming practice based on the abstract model to select software development tools of Zwartjes and van Geffen [Zwartjes and van Geffen, 2005]. This generic evaluation framework for tools supporting the Distributed Pair Programming practice was then used for the evaluation of a subset of the tools found in the research for software tools with potential to support the Distributed Pair Programming practice. The results of this evaluation as well as some general findings with respect to existing tools supporting the Distributed Pair Programming practice were discussed.

The discussion as well as the research shows that the tools with potential to support the Distributed Pair Programming practice existing today still show significant shortcomings with respect to some required and desired properties of such tools. The most severe shortcoming found was the lack of support of gesturing and deictic referencing within most of the existing tools that can be used to support the Distributed Pair Programming practice. Not as severe as the lack of support of gesturing and deictic referencing as it can be easily substituted with other tools but still an issue with some impact to the Distributed Pair Programming activity is the lack of tools with potential to support the Distributed Pair Programming practice to provide means for the collaborating partners to talk to each other like in the collocated case. Today’s available technology is fully fledged to support such means of communication. Certainly there might be potential for improvement in the support of workspace awareness within tools with potential to support the Distributed Pair Programming practice as there is still quite a way to go to provide the same workspace awareness in the tool supported distributed case as in the collocated one.

The outcome of the discussion shows that with respect to tools supporting the Distributed Pair Programming practice there is still much to do in future work. A number of the tools in existence today are well suited to support the Distributed Pair Programming practice but there is much potential to reduce the gap between the collocated and the non-collocated Pair Programming experience.

Another interesting topic for future work would be to apply the generic evaluation framework for tools supporting the Distributed Pair Programming practice in an evaluation of the tools that could not be included in the evaluation of this diploma thesis. Especially for those tools which were found to be quite suitable for the support of the Distributed Pair Programming practice it would be quite interesting as well to use the generic evaluation framework for tools supporting the Distributed Pair Programming practice in a broad empirical evaluation of these tools and compare the outcome of the same to the case study evaluation done in scope of this diploma thesis to verify the findings.

# 9 References

[Abrahamsson, Salo, et al., 2002] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta. *Agile software development methods. Review and analysis*. Espoo 2002. VTT Publications 478.

[Abrahamsson, Warsta, et al., May 2003] P. Abrahamsson, J. Warsta, M. T. Siponen and J. Ronkainen. *New Directions on Agile Methods: A Comparative Analysis*. Copyright 2003 IEEE. Published in *Proceedings of the International Conference on Software Engineering*, May 2003, Portland, Oregon, USA.

[ACE] ACE - A Collaborative Editor. http://ace.iserver.ch/, current on 28.09.2007.

[ACE Usermanual] Mark Bigler, Simon Raess, Lukas Zbinden. http://ace.iserver.ch/documentation/usermanual.pdf, current on 28.09.2007.

[Atsuta and Matsuura, 2004] S. Atsuta and S. Matsuura. *eXtreme programming support tool in distributed environment*. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts* (COMPSAC'04), pages 32–33, 2004.

[Auvinen, Back, et al., October 2005] J. Auvinen, R. Back, J. Heidenberg, P. Hirkman and L. Milovanov. *Improving the Engineering Area at Ericsson with Agile Practices - A Case Study*. Turku Centre for Computer Science, TUCS Technical Report No. 716, October 2005.

[Baheti, Gehringer and Stotts, 2002] P. Baheti, E. Gehringer and D. Stotts. *Exploring the efficacy of distributed pair programming*. In *Extreme Programming and Agile Methods – XP/Agile Universe 2002*, number 2418 in LNCS, pages 208-220, Springer, 2002.

[Baheti, Williams, et al., 2002] P. Baheti, L. A. Williams, E. Gehringer and D. Stotts. *Exploring Pair Programming in Distributed Object-Oriented Team Projects*. Department of Computer Science, North Carolina State University, 2002.

[Baheti, Williams, et al., March 2002] P. Baheti, L. A. Williams, E. Gehringer, D. Stotts and J. Smith. *Distributed Pair Programming: Empirical Studies and Supporting Environments*. University of North Carolina at Chapel Hill, Department of Computer Science, Technical Report TR02-010, March 2002.

[Beck, 2000] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[Canfora, Cimitile and Visaggio, 2003] G. Canfora, A. Cimitile and C. A. Visaggio. *Lessons learned about distributed pair programming: what are the knowledge needs to address?* In *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE03)*, pages 314–319, 2003.

[Ciolkowski and Schlemmer] M. Ciolkowski and M. Schlemmer. *Experiences with a Case Study on Pair Programming*. University of Kaiserslautern, year indeterminate.

[Clark, 1996] H. H. Clark. *Using language*. New York: Cambridge University Press, 1996.

[Clark and Brennan, 1991] H. H. Clark and S. E. Brennan. *Grounding in communication*. In L. Resnick, J. M. Levine, and S. D. Teasley (editors) *Perspectives on Socially Shared Cognition*, Washington, DC: APA, pages 127-149, 1991.

[Cockburn, 2000] A. Cockburn. *Writing Effective Use Cases, The Crystal Collection for Software Professionals*, Addison-Wesley Professional, 2000.

[Cockburn and Williams, 2001] A. Cockburn and L. A. Williams. *The costs and benefits of pair programming*. In *Extreme Programming Examined*, G. Succi, M. Marchesi (editors), pages 223-248, Boston, MA: Addison Wesley, 2001.

[CodeWright Usermanual] Borland CodeWright 7.5 User Manual. http://www.kessler.de/prd/starbase/usrguide.pdf, current on 28.09.2007.

[Cohen, Lindvall and Costa, January 2003] D. Cohen, M. Lindvall and P. Costa. *Agile Software Development*. Fraunhofer Center for Experimental Software Engineering Maryland and the University of Maryland, January 2003.

[collab.NetBeans] The NetBeans Collaboration Project. http://collab.netbeans.org/, current on 30.07.2007.

[Constantine, 1995] L. Constantine. *Constantine and Peopleware*. Englewood Cliffs, NJ: Yourdon Press, 1995.

[Coplien, 1995] J. Coplien. *A Development Process Generative Pattern Language*. In *Pattern Languages of Program Design*, pages 183-237, Ed. Reading, MA: Addison-Wesley, 1995.

[Cox and Greenberg, 2000] D. Cox and S. Greenberg. *Supporting collaborative interpretation in distributed groupware*. In *CSCW ’00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 289–298, 2000.

[DocSynch] DocSynch on Sourceforge.net. http://docsynch.sourceforge.net/, current on 30.09.2007.

[Dyck, Gutwin, et al., 2004] J. Dyck, C. Gutwin, S. Subramanian and C. Fedak. *Highperformance telepointers*. In *CSCW ’04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 172–181, 2004.

[George and Mansour, 2002] B. George, Y. M. Mansour. *A Multidisciplinary Virtual Team*. Accepted at Systemics, Cybernetics and Informatics (SCI), 2002.

[Gerken, 2003/04] J. F. Gerken. *Pair Programming – Eine kurze Abhandlung*. University of Bremen, Studiengang Informatik, Winter term 2003/04.

[Gobby] Gobby Homepage. http://gobby.0x539.de/trac/, current on 30.09.2007.

[Granville and Hickey, 2005] K. Granville and T. J. Hickey. *The Design, Implementation, and Application of the GrewpEdit Tool*. Department of Computer Science, Brandeis University, 2005.

[Greenberg, 1990] S. Greenberg. *Sharing views and interactions with single-user applications*. In *Proceedings of the conference on Office information systems*, pages 227–237, 1990.

[Greenberg, Gutwin and Roseman, 1996] S. Greenberg, C. Gutwin and M. Roseman. *Semantic telepointers for groupware*. In *Proceedings of the 6th Australian Conference on Computer-Human Interaction*, pages 54–61, 1996.

[GrewpEdit] GrewpEdit. http://groupscheme.sourceforge.net/grewpedit/index.html, current on 30.09.2007.

[Gutwin and Greenberg, September 1999] C. Gutwin and S. Greenberg. *The effects of workspace awareness support on the usability of real-time distributed groupware*. In *ACM Transactions on Computer-Human Interaction*, 6(3), pages 243–281, September 1999.

[Gutwin and Penner, 2002] C. Gutwin and R. Penner. *Improving interpretation of remote gestures with telepointer traces*. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 49–57, 2002.

[Hanks, December 2005] B. F. Hanks. *Empirical Studies of Distributed Pair Programming*. Dissertation in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science, University of California Santa Cruz, December 2005.

[Haynes and Friedenberg, May 2006] S. R. Haynes and M. Friedenberg. *Best Practices in Agile Development*. Technical Report No. 0018, College of Information Sciences and Technology, The Pennsylvania State University, May 2006.

[Ho, Raha, et al., October 2004] C.-W. Ho, S. Raha, E. Gehringer and L. A. Williams. *Sangam – A Distributed Pair Programming Plug-in for Eclipse*. Department of Computer Science, North Carolina State University, and Industrial Logic, Berkley. In *OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop*, October 24-28, 2004.

[Humphrey, 1997] W. S. Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley, 1997.

[Hulkko and Abrahamsson, May 2005] H. Hulkko and P. Abrahamsson. *A Mutltiple Case Study on the impact of Pair Programming on Product Quality*. Elektrobit Ltd., VTT Technical Research Center of Finland, ICSE'05, May 2005.

[ICQ] ICQ Homepage. http://icq.com/, current on 29.10.2007.

[ISO9241] ISO 9241: Ergonomic requirements for office work with visual display terminals.

[InfoWorld, March 2007] InfoWorld, March 26, 2007, issue 13, http://www.codegear.com/article/34209/images/34209/1-20483432-eprint.pdf, current 30.09.2007.

[Jabber] Jabber Homepage. http://www.jabber.org/, current on 29.10.2007.

[JBuilder] Borland CodeGear JBuilder 2007. http://www.codegear.com/en/products/jbuilder, current on 30.09.2007.

[JBuilder Reviewer's Guide] Borland CodeGear JBuilder 2007 Reviewer's Guide. http://www.codegear.com/article/34448/images/34448/JBuilder2007R2_ReviewersGuide.pdf, current on 30.09.2007.

[Kitchenham, Linkman and Law, 1997] B. Kitchenham, S. Linkman and D. Law. *DESMET: A methodology for evaluating software engineering methods and tools*. In *Computing and Control Engineering Journal*, pages 120-126, 1997.

[Kruchten, 2000] P. Kruchten. *The Rational Unified Process - An Introduction*. Second Edition, Addison-Wesley, 2000

[Lui and Chan, 2003] K. M. Lui and K. C. C. Chan. *When Does a Pair Outperform Two Individuals?* XP2003, Italy, 2003.

[Mark, Grudin and Poltrock, 1999] G. Mark, J. Grudin and S. Poltrock. *Meeting at the desktop: An empirical study of virtual collocated teams*. In *Proceedings of the 6th European Conference on Computer Supported Cooperative Work*, pages 159-178, 1999.

[Marratech] Marratech Homepage. http://www.marratech.com/, current on 22.10.2007.

[Maurer, 2002] F. Maurer. *Supporting Distributed Extreme Programming*. In D. Wells and L. Williams (Editors) *XP/Agile Universe 2002.* LNCS 2418, pages 13–22, 2002.

[MoonEdit] MoonEdit Homepage. http://moonedit.com/indexen.htm, current on 22.10.2007.

[Müller and Tichy, 2001] M. Müller and W. Tichy. *Case Study: Extreme Programming in a University Environment*. Computer Science Department, University of Karlsruhe, 2001.

[Müller, 2003] M. M. Müller. *Are Reviews an Alternative to Pair Programming?* 7th International Conference on Empirical Assessment in Software Engineering, UK, 2003.

[Natsu, Favela, et al., 2003] H. Natsu, J. Favela, A. Morán, D. Decouchant and A. Martinez-Enriquez. *Distributed Pair Programming on the Web*. In *Proceedings of the 4th Mexican International Conference on Computer Science*, pages 81-88, 2003.

[Naur and Randell, October 1968] P. Naur and B. Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee*. Brussels, Scientific Affairs Division, NATO, 7.-11. October, 1968.

[Navoraphan, Gehringer, et al., October 2006] K. Navoraphan, E. Gehringer, J. Culp, K. Gyllstrom and D. Stotts. *Next-Generation DPP with Sangam and Facetop*. In *OOPSLA Eclipse Technology Exchange*, October 22 -23, 2006, Department of Computer Science, North Carolina State University, and Department of Computer Science, University of North Carolina.

[Nawrocki and Wojciechowski] J. Nawrocki and A. Wojciechowski. *Experimental Evaluation of Pair Programming*, year indeterminate.

[NetMeeting] Microsoft NetMeeting http://www.microsoft.com/downloads/details.aspx?FamilyID=26c9da7c-f778-4422-a6f4-efb8abba021e&displaylang=en, current on 07.09.2007.

[NetMeeting-Screenshot] http://www.www-kurs.de/confernc.htm, current on 09.09.2007.

[Nosek, March 1998] J. T. Nosek. *The case for collaborative programming*. In *Communications of the ACM 41:3*, pages 105-108, March 1998.

[Olson and Olson, 2000] G. M. Olson and J. S. Olson. *Distance Matters*. In *Human-Computer Interaction, volume 15*, pages 139-179, 2000.

[pcAnywhere] Symantec pcAnywhere. http://www.symantec.com/norton/products/overview.jsp?pcid=pf&pvid=pca121, current on 09.09.2007.

[PEP] PEP - Pair Eclipse Programming Homepage. http://sourceforge.net/projects/pep-pp/, current on 09.09.2007.

[Randell and Buxton, October 1969] B. Randell and J. N. Buxton, editors. *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee*. Brussels, Scientific Affairs Division, NATO, 27.-31. October, 1969.

[RealVNC] RealVNC, Ltd. RealVNC. http://www.realvnc.com/, current on 07.09.2007.

[Reeves and Zhu, 2004] M. Reeves and J. Zhu. *Moomba – a collaborative environment for supporting distributed extreme programming in global software development*. In *Proceedings of XP 2004*, number 3092 in LNCS, pages 38-50, Springer, 2004.

[RemoteAdministrator] Famatech Remote Administrator. http://www.famatech.com/products/radmin/index.php, current on 09.09.2007.

[RemoteAdministrator-Wikipedia] Wikipedia entry about Remote Administrator. http://en.wikipedia.org/wiki/Remote_Administrator, current on 09.09.2007.

[Richardson, Stafford-Fraser, et al., February 1998] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper. *Virtual network computing*. In *IEEE Internet Computing*, 2(1), pages 33–38, January-February 1998.

[Sangam] Sangam Homepage. http://sourceforge.net/projects/sangam/, current on 22.10.2007.

[Saros] Saros Homepage. http://dpp.sourceforge.net/, current on 24.10.2007.

[Schummer and Schummer, 2001] T. and J. Schummer. *Support for distributed teams in extreme programming*. In G. Succi and M. Marchesi, editors, *Extreme Programming Examined*, pages 355-378, Addison-Wesley, 2001.

[Sourceforge] SourceForge.net. http://sourceforge.net/, current on 28.09.2007

[Skype] Skype Homepage. http://www.skype.com/, current on 29.10.2007.

[Stotts, Smith and Gyllstrom, 2004] D. Stotts, J. Smith and K. Gyllstrom. *Support for distributed Pair Programming in the transparent Video Facetop*. In *Proceedings of the 4th conference on Extreme Programming and Agile Methods – XP/Agile Universe*, number 3134 in LNCS, pages 92-104, Springer, 2004.

[Stotts, Smith and Williams, March 2002] D. Stotts, J. Smith and L. A. Williams. *Hypervideo support for distributed extreme programming*. Technical Data Report TR02-009, Department of Computer Science, University of North Carolina at Chapel Hill, March 2002.

[SubEthaEdit] Coding Monkeys' SubEtha Edit. http://www.codingmonkeys.de/subethaedit/, current on 09.09.2007

[SynchroEdit] SynchroEdit Homepage. http://www.synchroedit.com/, current on 24.10.2007.

[TalkAndWrite] TalkAndWrite pug-in for Skype Homepage. http://www.talkandwrite.com/english/, current on 24.10.2007.

[Tang, February 1991] J. C. Tang. *Findings from observational studies of collaborative work*. In *International Journal of Man Machine Studies, 34(2)*, pages 143–160, February 1991.

[Telescope] Telescope Sourceforge Homepage. http://sourceforge.net/projects/telescope/, current on 24.10.2007.

[TightVNC] TightVNC. http://www.tightvnc.com/, current on 09.09.2007.

[Tockey, November/December 1997] S. Tockey. *A Missing Link in Software Engineering*. In *IEEE Software*, pages 31-36, November/December 1997.

[UltraVNC] UltraVNC. http://www.uvnc.com/, current on 09.09.2007.

[Versteegen, 2000] G. Versteegen (Hrsg.). *Das V-Modell in der Praxis*. dpunkt.verlag, 2000.

[VNC-Wikipedia] Wikipedia entry about VNC. http://en.wikipedia.org/wiki/VNC, http://de.wikipedia.org/wiki/VNC, current on 09.09.2007.

[Weinberg, 1998] G. M. Weinberg. *The Psychology of Computer Programming*. Silver Anniversary Edition, New York: Dorset House Publishing, 1998.

[Williams] L. A. Williams. *Integrating Pair Programming into a Software Development Process*. Department of Computer Science, North Carolina State University, year indeterminate.

[Williams, 2000] L. A. Williams. *The Collaborative Software Process*. PhD Dissertation, Department of Computer Science, University of Utah. Salt Lake City, 2000.

[Williams and Erdogmus] L. A. Williams and H. Erdogmus. *On the Economic Feasibility of Pair Programming*. Year indeterminate.

[Williams and Kessler, May 2000] L. A. Williams and R. Kessler. *All I Really Need to Know about Pair Programming I Learned in Kindergarten*. In *Communication of the ACM May 2000/Vol 43 No. 5*, pages 108-114, May 2000.

[Williams and Kessler, 2002] L. A. Williams and R. Kessler. *Pair Programming Illuminated*. Addison-Wesley, 2002.

[Williams, Kessler, et al., 2000] L. A. Williams, R. Kessler, W. Cunningham and R. Jeffries. *Strengthening the case for pair-programming*. In *IEEE Software*, 17:4, pages 19-25, July/August 2000.

[Williams, Layman, et al.] L. A. Williams, L. Layman, W. Krebs and A. Antón. *Exploring the Use of a "Safe Subset" of Extreme Programming: An Industrial Case Study*. North Carolina State University, Department of Computer Science, IBM Corporation, year indeterminate.

[x11vnc] x11vnc. http://www.karlrunge.com/x11vnc/, current on 09.09.2007.

[XecliP] XecliP Sourceforge Homepage. http://sourceforge.net/projects/xeclip/, current on 24.10.2007.

[XPairtise] XPairtise Sourceforge Homepage. http://sourceforge.net/projects/xpairtise/, current on 24.10.2007.

[Zwartjes and van Geffen, 2005] G. Zwartjes and J. van Geffen. *An Agile Approach Supported by a Tool Environment for the Development of Software Components*. Master Thesis, Technical University Eindhoven, 2005.

# 10 Appendix

## 10.1 Abbreviations

| | |
|---|---|
| *CPU* | *Central Processing Unit* |
| *CSCW* | *Computer Supported Cooperative Work* |
| *DPP* | *Distributed Pair Programming* |
| *IDE* | *Integrated Development Environment* |
| *IP* | *Internet Protocol* |
| *IT* | *Information Technology* |
| *LAN* | *Local Area Network* |
| *LOC* | *Lines of Code* |
| *PP* | *Pair Programming* |
| *UML* | *Unified Modelling Language* |
| *VoIP* | *Voice over IP* |
| *WYSIWYG* | *What You See Is What You Get* |
| *XP* | *Extreme Programming* |

*Table 15. Abbreviations*

## 10.2 Scenario Descriptions

### 10.2.1 Scenario #1: Installation and configuration of the tool to be used

This scenario covers activities to be performed during installation and configuration of the tool which is going to be tested with the following scenarios. Additionally this scenario covers general properties of the tool like the quality of the available documentation as well as which tool immanent properties are covered by the tool and on which platforms it can be used.

This scenario is performed offline by each of the evaluation team partners.

#### 10.2.1.1 Pre-Conditions

- The installation package(s) of the tool is (are) available.
- Any already known additional software needed for the set up of the tooling is available.
- If available, the installation and configuration manual of the tooling is at hand.

#### 10.2.1.2 Scenario

- Evaluate the quality of the available installation and configuration documentation of the tool.

- Evaluate (from the available documentation / description of or knowledge about the tool) to which category the tool to be evaluated belongs and which tool immanent properties are covered by the tooling.
- Evaluate (from the available documentation / description of or knowledge about the tool) which platforms are supported by the tool.
- Install and configure the tool (as well as any additional software needed if known so far) on the machines of the pair doing the evaluation.

#### 10.2.1.3 Post conditions

- Tooling is installed and configured.
- The evaluating pair is ready to set up a Distributed Pair Programming session with the tooling.

### 10.2.2 Scenario #2: Setting up a Distributed Pair Programming session

This scenario covers activities to be performed while setting up a session between two partners with a tool usable or specifically designed for the support of Distributed Pair Programming.

#### 10.2.2.1 Pre-Conditions

- Scenario #1 (Installation and configuration of the tool to be evaluated) has been executed successfully.
- If available, the user manual of the tool is at hand.

#### 10.2.2.2 Scenario

- Both partners of the evaluation pair start the tool (as well as any additional software needed) on their respective machines.
- If the connection between the machines of the evaluation pair is not established automatically (i.e. due to corresponding settings done during configuration of the tool) by the tool, the partners perform the needed steps to establish the connection needed for the Pair Programming session with the tool. Depending on the tool this is either done by connecting to a server to which both partners need to connect or in a way that one partner of the pair connects to the others machine.
- Both partners check if the tool provides support with respect to the awareness of each other partner’s presence. Is there any notification providing information that the connection to the other partner or the server has been established successfully? In case of a tool using a common server to connect to, is there any notification providing information that the other partner has joined the session?
- If the tool features the possibility to store a previous session and such a session was already stored during a prior Pair Programming session, try to restore the stored session. If no previous session has been stored but this feature is supported by the tooling, this scenario has to be performed multiple times to check this.
- Both partners evaluate how the initial distribution of the Pair Programming roles (Driver, Navigator) is done. Is the initial role distribution preset by the tool or is it negotiated and manually set up by the partners? If a previous session is

restored, is the initial role distribution restored to the role distribution as it was when the previous session was stored, is it preset by the tooling or is it negotiated and manually set up by the partners?

- The partners start any other tools they already found the need to use to enable them to perform Pair Programming activities, respectively continue with the evaluation (i.e. tools to provide communication channels not supported by the evaluated tool itself).
- The partners negotiate what they plan to do in the now set up session.

#### 10.2.2.3 Post conditions

- The Pair Programming session is set up and active.
- The initial role distribution is done and clear to both partners.
- Both partners are aware of what they plan to do in their set up session.

### 10.2.3 Scenario #3: First steps and getting to know the tool

Within this scenario the Pair Programming partners perform their first steps with the tool under evaluation. The target of this scenario is to get to know the main features of the tool under evaluation and get a first impression on how good Distributed Pair Programming is supported by the tool.

#### 10.2.3.1 Pre-Conditions

- Scenario #2 (Setting up a session for Distributed Pair Programming) has been executed successfully.
- All additional tools which are already found to be needed to enable the partners to perform Pair Programming activities, respectively do the evaluation are started.
- Both partners are aware of what they actually plan to do while going through this scenario (especially with respect to the simple example which will be done in scope of this scenario).
- If available, the user manual of the tool is at hand.

#### 10.2.3.2 Scenario

- Both partners check the tool’s user manual documentation (if available) for the features supported by the tool and evaluate the quality of the available user manual documentation of the tool. This step can also be done offline separately by the partners prior to establishing the session.
- The Pair Programming pair checks which communication channels are provided by the tool. For each communication channel provided by the tool the pair evaluates whether the channel works as expected.
    - o Does the tool provide means to communication with each other via speech?
    - o Does the tool provide means to communication with each other by means of typed text? Is there a chat channel? Is there a whiteboard? Are there both? Do they work as expected?
    - o Does the tool provide means to communication with each other by means streamed video?

- Is there a need to use additional tools to provide communication channels not supported by the tool itself? If yes, such tools are now started and the corresponding connection is established by both partners.

- The Pair Programming pair does first trials of the usage and features of the tools user interface.
    - How easy and intuitive is the tool's user interface to use on the local screen?
    - How easy and intuitive is the tool's user interface to use with respect to the collaborative environment? Is there a support of the tool's user interface to provide information about the collaboration with each other's partner? How good is this supported?
- The Pair Programming pair performs a simple first example to get to know how to work with the tool. This simple example covers the following activities / tasks:
    - The Pair Programming pair writes some code or text. The Navigator actively participates in scope of his role. In scope of this it is checked how fluently the screen is updated at the Navigator's side (how up-to-date the screen is).
    - If a second pointer (for the Navigator) is provided by the tooling, the Pair Programming pair tries out its usage and semantics.
    - The Pair Programming pair performs simple role changes between Driver and Navigator while writing.
    - The Pair Programming pair evaluates the "feeling" of the workspace awareness.

#### 10.2.3.3 Post conditions

- The Pair Programming partners are now aware how to use the tool under evaluation.
- The tool has undergone first evaluation against those properties defined as critical for a tool supporting Distributed Pair Programming and supports these features.
- The Pair Programming partners have a first impression on how good the tool supports many of the other properties.

### 10.2.4 Scenario #4: Role Changes

This scenario covers all evaluations with respect to role changes between Driver and Navigator in the process of Distributed Pair Programming. Although role changes are certainly also part of scenarios #3 and #6, as this is a major as well as complex activity in the process of Distributed Pair Programming it is put into an own scenario so that the different use cases and ways of role changes are clear to be covered in scope of the evaluation.

#### 10.2.4.1 Pre-Conditions

- A Pair Programming session is ongoing.
- The role change between Driver and Navigator is triggered in some way. Possible triggers for such a role change are:

  - An intuitive change is done as the Navigator intuitively takes over the role of the Driver. For such a role change the tool needs to support informal role changes.
  - In scope of the Pair Programming activities the Navigator requests to take over the role of the Driver (or vice versa) and both agree to exchange their roles. Although such a role change is also possible and might as well happen if the tool in evaluation supports informal role changes, this is the only way to exchange the roles between Driver and Navigator if only formal role changes are supported by the tool.
  - The tool itself gives a reminder (i.e. based on a preset time-slice between role changes) that the Pair Programming partners should exchange their roles (or the set time has expired). Possibly some tools even force the Pair Programming partners to exchange their roles after a set time in which no role change occurred expired. A reminder could be possible in such tools supporting informal role changes as well as such supporting formal role changes whereas a forced role change is definitely rather a formal role change.

#### 10.2.4.2 Scenario

In this scenario actually one of the following sub-scenarios is performed. To evaluate all of the following sub-scenarios, this scenario needs to be performed multiple times.

- The Pair Programming partners exchange their roles in scope of an intentional and agreed role change. This takes place if either one of the pair requests a role change or the tool reminds the partners to change their roles (but does not force them to) and both agree on exchanging their roles.
  - Does the tool support a reminding functionality based on setting a time-slice after which the partners should be reminded to exchange roles?
  - How is the request / reminder for the role change communicated from one partner to the other (respectively from the tool to the partners)?
  - Does the tool require a formal role change or is an informal role change after the social agreement of the role change by the partners sufficient?
  - How easy is it to exchange the roles? Does the role change impact the flow of the Pair Programming activity?
  - Does the tool in some way provide information to the Pair Programming partners about who is currently in which role (Driver, Navigator)?
- The Pair Programming partners exchange their roles unintentional and intuitively due to the workflow of the Pair Programming activity or the Navigator just takes over the Drivers role by force.
  - Is there a formal activity (i.e. by the Navigator) necessary for such a role change?
  - How easy is it to exchange the roles? Does the role change impact the flow of the Pair Programming activity?
  - How is the role exchange reflected by the tool? How do the partners know in which role they are in after such a role change?
- The Pair Programming partners are forced to exchange their roles by the tool.
  - How is this formally handled in the tool?

    - Does the role change impact the flow of the Pair Programming activity?
- Does the tool provide statistics about which of the partners impersonated which role for which amount of time?

#### 10.2.4.3 Post conditions

- The roles of the Pair Programming partners are exchanged.
- The Pair Programming partners are (at least intuitively) aware of their changed roles.

### 10.2.5 Scenario #5: Session Management

This scenario covers all evaluations with respect to session management, therefore part of it can only be tested if the tool in evaluation supports means to store and restore sessions. Although session management is certainly also part of scenarios #3 and #6, as this is a complex scenario by itself in the process of Distributed Pair Programming it is put into an own scenario so that the different use cases of session management are clear to be covered in scope of the evaluation.

#### 10.2.5.1 Pre-Conditions

- A Pair Programming session is ongoing.

#### 10.2.5.2 Scenario

In this scenario actually one of the following sub-scenarios is performed. To evaluate all of the following sub-scenarios, this scenario needs to be performed multiple times.

- Both Pair Programming partners intentionally quit their ongoing Pair Programming session at the same time.
    - Is it possible to store the session?
    - Can the session be restored when the partners join for the next Pair Programming session?
    - If some kind of configuration management (respectively versioning) is supported by the tool and testable (not needing additional tools which need licenses) in the evaluation, is their a possibility to restore a session with an older saved version?
- One of the Pair Programming partners intentionally quits the ongoing Pair Programming session without agreeing this with the other partner.
    - How is such a one-sided break up of the session communicated to the other partner by the tool?
    - Is the session stored and if yes, in which state (at the moment of the break up of the session, if the quitting partner stores it intentionally, if the other partner stores it manually, if the other partner quits the session as he lost his Pair Programming partner)?
    - Can the session be restored when the partners join for the next Pair Programming session?
    - If some kind of configuration management (respectively versioning) is supported by the tool and testable (not needing additional tools which need

licenses) in the evaluation, is their a possibility to restore a session with an older saved version (as it might happen that the Driver continues to work as he is not aware that the Navigator has left and then stores the session)?

- The Pair Programming session is broken due to LAN outage or crash of the tool.
    - o If the tool of one or both partners is still up and running, how is such a break up of the session communicated to the partners by the tool?
    - o Is the session stored and if yes, in which state (at the moment of the break up of the session, if one of the partners still active quits the session as he lost his Pair Programming partner)?
    - o Can the session be restored when the partners join for the next Pair Programming session?
    - o If some kind of configuration management (respectively versioning) is supported by the tool and testable (not needing additional tools which need licenses) in the evaluation, is their a possibility to restore a session with an older saved version (as if might happen that the Driver continues to work as he is not aware that the Navigator has left and then stores the session)?

#### 10.2.5.3 Post conditions

- Not appropriate.

### 10.2.6 Scenario #6: Using the tooling in performing Pair Programming

This scenario covers all activities and properties evaluated while actually working with the tool under evaluation in scope of a Pair Programming process. This scenario is actually a collection of sub-scenarios where also the scenarios #4 and #5 are mentioned once again for matters of completeness.

#### 10.2.6.1 Pre-Conditions

- Scenario #2 (Setting up a session for Distributed Pair Programming) has been executed successfully.
- Scenario #3 (First steps and getting to know the tool) has been executed to get to know the tool under evaluation.
- All additional tools which are needed to enable the partners to perform Pair Programming activities respectively to do the evaluation are started.
- Both partners are aware of what they actually plan to do while going through this scenario (especially with respect to the simple example which will be done in scope of this scenario).
- If available, the user manual of the tool is at hand.

#### 10.2.6.2 Scenario

In this scenario actually one or multiple of the following sub-scenarios is performed. To evaluate all of the following sub-scenarios, this scenario might be needed to be performed multiple times.

- The Driver writes code intentionally including errors into the code.
- The Navigator examines the Drivers work. The Navigator finds a fault or design flaw.

  - The Navigator wants to inform the Driver of this sighting. How does the communication of this to the Driver take place?
  - If necessary, a role change between Driver and Navigator is done (see scenario #4).
- The communication channels provided by the tooling in evaluation to the two Pair Programming partners are thoroughly tested and evaluated in real work situations. This could for example be that the Driver requests some information from the Navigator which is looked up by the Navigator and then provided to the Driver or that the Navigator finds a fault or possible design flaw and communicates this to the Driver.
- The Pair Programming pair executes multiple role changes and evaluates all possibilities provided by the tool under evaluation to exchange roles. See scenario #4.
- The Pair Programming pair evaluates the session management support of the tool under evaluation. See scenario #5.
- Any special features or the tooling are tested and documented by the Pair Programming partners.
  - A special feature found as a possibly usable property of a tool supporting Distributed Pair Programming is a pair-finding support given by the tool.

#### 10.2.6.3 Post conditions

- The evaluation of the tool currently in evaluation is finished.
- All of properties are quantified with respect to their grade of fulfilment by the tooling in evaluation.
- Verbal description for all properties (including describing the set grade of fulfilment) of the tool in evaluation and all findings not mapping to properties of the evaluation framework is available.

## 10.3 Evaluation Results

### 10.3.1 Screen-Sharing Applications

| Property-ID | Tool Category | Property Category | Property | importance | NetMeeting | NetMeeting (weighted) | VNC | VNC (weighted) |
|---|---|---|---|---|---|---|---|---|
| 1 | ***Screen Sharing Tools*** | *Basic* | Support of Screen Sharing | C | 4 | 40 | 4 | 40 |
| 2 | ***Screen Sharing Tools*** | *Basic* | Support of Workspace Awareness | C | 4 | 40 | 5 | 50 |
| 3 | ***Screen Sharing Tools*** | *Support of Gesturing* | Availability of a Second Pointer for the Navigator | C | 1 | 10 | 1 | 10 |
| 4 | ***Screen Sharing Tools*** | *Support of Gesturing* | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | M | 1 | 2,5 | 0 | 0 |
| 5 | ***Screen Sharing Tools*** | *Support of Gesturing* | Suppression of Telepointer Jitter | L | 5 | 5 | 3 | 3 |
| 6 | ***Screen Sharing Tools*** | *Support of Gesturing* | Support of Highlighting with Respect to Collaboration | H | 2 | 10 | 5 | 25 |
| 7 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of Floor Control | C | 3 | 30 | 0 | 0 |
| 8 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of Role Changes | C | 2 | 20 | 0 | 0 |
| 9 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of Informal Role Changes (social Floor control) | M | 0 | 0 | 2 | 5 |
| 10 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of Formal Role Changes (application/master-user mediated Floor control) | M | 2 | 5 | 0 | 0 |
| 11 | ***Screen Sharing Tools*** | *Floor Control Support* | Easiness of Role Changes | M | 2 | 5 | 5 | 12,5 |
| 12 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of the Information "Who is in which role" | M | 2 | 5 | 0 | 0 |
| 13 | ***Screen Sharing Tools*** | *Floor Control Support* | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | L | 0 | 0 | 0 | 0 |
| 14 | ***Screen Sharing Tools*** | *Communication Possibilities* | Support of a Speech Channel | H | 5 | 25 | 0 | 0 |
| 15 | ***Screen Sharing Tools*** | *Communication Possibilities* | Support of textual Chat | M | 4 | 10 | 0 | 0 |
| 16 | ***Screen Sharing Tools*** | *Communication Possibilities* | Support of a White-Board | M | 4 | 10 | 0 | 0 |
| 17 | ***Screen Sharing Tools*** | *Communication Possibilities* | Support of a Video Channel | L | 5 | 5 | 0 | 0 |
| 18 | ***Screen Sharing Tools*** | *Communication Possibilities* | Pair/Partner Finding Support | L | 0 | 0 | 0 | 0 |
| 19 | ***Screen Sharing Tools*** | *Platform Independence* | Support of Microsoft Windows | M | 1 | 2,5 | 1 | 2,5 |
| 20 | ***Screen Sharing Tools*** | *Platform Independence* | Support of Unix/Linux | M | 0 | 0 | 1 | 2,5 |

| Property-ID | Tool Category | Property Category | Property | importance | NetMeeting | NetMeeting (weighted) | VNC | VNC (weighted) |
|---|---|---|---|---|---|---|---|---|
| 21 | ***Screen Sharing Tools*** | *Platform Independence* | Support of MacOS | L | 0 | 0 | 0 | 0 |
| 22 | ***Screen Sharing Tools*** | *Usability* | Intuitiveness of tool usage | H | 3 | 15 | 5 | 25 |
| 23 | ***Screen Sharing Tools*** | *Usability* | Easiness of Installation and configuration | L | 4 | 4 | 5 | 5 |
| 24 | ***Screen Sharing Tools*** | *Usability* | Easiness of Start up and establishing a session | M | 3 | 7,5 | 5 | 12,5 |
| 25 | ***Screen Sharing Tools*** | *Usability* | Stability | H | 4 | 20 | 5 | 25 |
| 26 | ***Screen Sharing Tools*** | *Usability* | Screen Actuality - Realtime-Probability | C | 4 | 40 | 4 | 40 |
| 27 | ***Screen Sharing Tools*** | *Usability* | Easiness of Collaboration | C | 4 | 40 | 4 | 40 |
| 28 | ***Screen Sharing Tools*** | *Usability* | Intuitiveness of User interface | M | 3 | 7,5 | 5 | 12,5 |
| 29 | ***Screen Sharing Tools*** | *Usability* | Need for other tools | H | 4 | 20 | 2 | 10 |
| 30 | ***Screen Sharing Tools*** | *Usability* | Support of Session Management | M | 0 | 0 | 0 | 0 |
| 31 | ***Screen Sharing Tools*** | *Tool* | Off-the-shelf - Commercial | L | 1 | 1 | 1 | 1 |
| 32 | ***Screen Sharing Tools*** | *Tool* | Off-the-shelf - Open Source | L | 0 | 0 | 1 | 1 |
| 33 | ***Screen Sharing Tools*** | *Tool* | Special Purpose | L | 0 | 0 | 0 | 0 |
| 34 | ***Screen Sharing Tools*** | *Tool* | IDE | L | 0 | 0 | 0 | 0 |
| 35 | ***Screen Sharing Tools*** | *Tool* | Viewer | L | 1 | 1 | 1 | 1 |
| 36 | ***Screen Sharing Tools*** | *Tool* | Configuration Management/Versioning Support | L | 0 | 0 | 0 | 0 |
| 37 | ***Screen Sharing Tools*** | *Documentation* | Documentation for Installation und configuration | L | 5 | 5 | 2 | 2 |
| 38 | ***Screen Sharing Tools*** | *Documentation* | User Manual | L | 4 | 4 | 2 | 2 |
|  |  |  |  |  | 87 | 390 | 69 | 327,5 |

*Table 16. Evaluation Results: Screen-Sharing Applications*

## 10.3.2 Collaboration-Aware Applications

| Property-ID | Tool Category | Property Category | Property | importance | ACE | ACE (weighted) | collab.netbeans | collab.netbeans (weighted) | COPPER | COPPER (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***Collaboration Aware Tools*** | *Basic* | Support of Collaboration Awareness | C | 3 | 30 | 4 | 40 | 4 | 40 |
| 2 | ***Collaboration Aware Tools*** | *Basic* | Support of Workspace Awareness | C | 2 | 20 | 2 | 20 | 4 | 40 |
| 3 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of a Second Pointer for the Navigator | C | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | M | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Suppression of Telepointer Jitter | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Support of Highlighting with Respect to Collaboration | H | 4 | 20 | 0 | 0 | 2 | 10 |
| 7 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Floor Control | C | 0 | 0 | 1 | 10 | 5 | 50 |
| 8 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Role Changes | C | 0 | 0 | 0 | 0 | 5 | 50 |
| 9 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Informal Role Changes (social Floor control) | M | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Formal Role Changes (application/master-user mediated Floor control) | M | 0 | 0 | 0 | 0 | 5 | 12,5 |
| 11 | ***Collaboration Aware Tools*** | *Floor Control Support* | Easiness of Role Changes | M | 0 | 0 | 0 | 0 | 5 | 12,5 |
| 12 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is in which role" | M | 0 | 0 | 1 | 2,5 | 5 | 12,5 |
| 13 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Speech Channel | H | 0 | 0 | 0 | 0 | 1 | 5 |
| 15 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of textual Chat | M | 0 | 0 | 5 | 12,5 | 3 | 7,5 |
| 16 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a White-Board | M | 2 | 5 | 2 | 5 | 2 | 5 |
| 17 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Video Channel | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Pair/Partner Finding Support | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Microsoft Windows | M | 1 | 2,5 | 1 | 2,5 | 1 | 2,5 |
| 20 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Unix/Linux | M | 2 | 5 | 2 | 5 | 1 | 2,5 |
| 21 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of MacOS | L | 0 | 0 | 1 | 1 | 1 | 1 |

| Property-ID | Tool Category | Property Category | Property | importance | ACE | ACE (weighted) | collab.netbeans | collab.netbeans (weighted) | COPPER | COPPER (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 22 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of tool usage | H | 4 | 20 | 5 | 25 | 5 | 25 |
| 23 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Installation and configuration | L | 5 | 5 | 5 | 5 | 3 | 3 |
| 24 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Start up and establishing a session | M | 4 | 10 | 5 | 12,5 | 4 | 10 |
| 25 | ***Collaboration Aware Tools*** | *Usability* | Stability | H | 2 | 10 | 4 | 20 | 4 | 20 |
| 26 | ***Collaboration Aware Tools*** | *Usability* | Screen Actuality - Realtime-Probability | C | 5 | 50 | 1 | 10 | 5 | 50 |
| 27 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Collaboration | C | 3 | 30 | 3 | 30 | 4 | 40 |
| 28 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of User interface | M | 4 | 10 | 4 | 10 | 4 | 10 |
| 29 | ***Collaboration Aware Tools*** | *Usability* | Support of Syntaxhighlighting | L | 0 | 0 | 5 | 5 | 0 | 0 |
| 30 | ***Collaboration Aware Tools*** | *Usability* | Need for other tools | H | 4 | 20 | 4 | 20 | 4 | 20 |
| 31 | ***Collaboration Aware Tools*** | *Usability* | Support of Session Management | M | 1 | 2,5 | 1 | 2,5 | 3 | 7,5 |
| 32 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Commercial | L | 0 | 0 | 1 | 1 | 0 | 0 |
| 33 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Open Source | L | 1 | 1 | 0 | 0 | 0 | 0 |
| 34 | ***Collaboration Aware Tools*** | *Tool* | Special Purpose | L | 1 | 1 | 1 | 1 | 1 | 1 |
| 35 | ***Collaboration Aware Tools*** | *Tool* | IDE | L | 0 | 0 | 1 | 1 | 0 | 0 |
| 36 | ***Collaboration Aware Tools*** | *Tool* | Collaborative Editor | L | 1 | 1 | 0 | 0 | 1 | 1 |
| 37 | ***Collaboration Aware Tools*** | *Tool* | Configuration Management/Versioning Support | L | 0 | 0 | 5 | 5 | 0 | 0 |
| 38 | ***Collaboration Aware Tools*** | *Documentation* | Documentation for Installation und configuration | L | 5 | 5 | 5 | 5 | 2 | 2 |
| 39 | ***Collaboration Aware Tools*** | *Documentation* | User Manual | L | 4 | 4 | 4 | 4 | 1 | 1 |
| | | | | | 58 | 252 | 73 | 255,5 | 85 | 441,5 |

*Table 17. Evaluation Results: Collaboration-Aware Applications (1)*

| Property-ID | Tool Category | Property Category | Property | importance | DocSynch | DocSynch (weighted) | Gobby | Gobby (weighted) | GrewpEdit | GrewpEdit (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***Collaboration Aware Tools*** | *Basic* | Support of Collaboration Awareness | C | 4 | 40 | 4 | 40 | 4 | 40 |
| 2 | ***Collaboration Aware Tools*** | *Basic* | Support of Workspace Awareness | C | 4 | 40 | 5 | 50 | 3 | 30 |
| 3 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of a Second Pointer for the Navigator | C | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | M | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Suppression of Telepointer Jitter | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Support of Highlighting with Respect to Collaboration | H | 0 | 0 | 1 | 5 | 3 | 15 |
| 7 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Floor Control | C | 4 | 40 | 0 | 0 | 0 | 0 |
| 8 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Role Changes | C | 3 | 30 | 0 | 0 | 0 | 0 |
| 9 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Informal Role Changes (social Floor control) | M | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Formal Role Changes (application/master-user mediated Floor control) | M | 40 | 10 | 0 | 0 | 0 | 0 |
| 11 | ***Collaboration Aware Tools*** | *Floor Control Support* | Easiness of Role Changes | M | 3 | 7,5 | 0 | 0 | 0 | 0 |
| 12 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is in which role" | M | 4 | 10 | 0 | 0 | 0 | 0 |
| 13 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Speech Channel | H | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of textual Chat | M | 5 | 12,5 | 5 | 12,5 | 3 | 7,5 |
| 16 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a White-Board | M | 0 | 0 | 0 | 0 | 3 | 7,5 |
| 17 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Video Channel | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Pair/Partner Finding Support | L | 2 | 2 | 0 | 0 | 0 | 0 |
| 19 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Microsoft Windows | M | 1 | 2,5 | 1 | 2,5 | 1 | 2,5 |
| 20 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Unix/Linux | M | 2 | 5 | 2 | 5 | 2 | 5 |
| 21 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of MacOS | L | 1 | 1 | 1 | 1 | 1 | 1 |
| 22 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of tool usage | H | 4 | 20 | 5 | 25 | 4 | 20 |

| Property-ID | Tool Category | Property Category | Property | importance | DocSynch | DocSynch (weighted) | Gobby | Gobby (weighted) | GrewpEdit | GrewpEdit (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 23 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Installation and configuration | L | 2 | 2 | 5 | 5 | 4 | 4 |
| 24 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Start up and establishing a session | M | 4 | 10 | 4 | 10 | 3 | 7,5 |
| 25 | ***Collaboration Aware Tools*** | *Usability* | Stability | H | 5 | 25 | 2 | 10 | 5 | 25 |
| 26 | ***Collaboration Aware Tools*** | *Usability* | Screen Actuality - Realtime-Probability | C | 2 | 20 | 5 | 50 | 4 | 40 |
| 27 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Collaboration | C | 3 | 30 | 5 | 50 | 5 | 50 |
| 28 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of User interface | M | 2 | 5 | 4 | 10 | 4 | 10 |
| 29 | ***Collaboration Aware Tools*** | *Usability* | Support of Syntaxhighlighting | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | ***Collaboration Aware Tools*** | *Usability* | Need for other tools | H | 4 | 20 | 4 | 20 | 3 | 15 |
| 31 | ***Collaboration Aware Tools*** | *Usability* | Support of Session Management | M | 1 | 2,5 | 1 | 2,5 | 1 | 2,5 |
| 32 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Commercial | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Open Source | L | 1 | 1 | 1 | 1 | 1 | 1 |
| 34 | ***Collaboration Aware Tools*** | *Tool* | Special Purpose | L | 1 | 1 | 1 | 1 | 1 | 1 |
| 35 | ***Collaboration Aware Tools*** | *Tool* | IDE | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 36 | ***Collaboration Aware Tools*** | *Tool* | Collaborative Editor | L | 1 | 1 | 1 | 1 | 1 | 1 |
| 37 | ***Collaboration Aware Tools*** | *Tool* | Configuration Management/Versioning Support | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 38 | ***Collaboration Aware Tools*** | *Documentation* | Documentation for Installation und configuration | L | 3 | 3 | 5 | 5 | 1 | 1 |
| 39 | ***Collaboration Aware Tools*** | *Documentation* | User Manual | L | 3 | 3 | 0 | 0 | 0 | 0 |
|  |  |  |  |  | 73 | 344 | 62 | 306,5 | 57 | 286,5 |

*Table 18. Evaluation Results: Collaboration-Aware Applications (2)*

| Property-ID | Tool Category | Property Category | Property | importance | MoonEdit | MoonEdit (weighted) | PEP - Pair Eclipse Programming | PEP - Pair Eclipse Programming (weighted) | TalkAndWrite plug-in for Skype | TalkAndWrite plug-in for Skype (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***Collaboration Aware Tools*** | *Basic* | Support of Collaboration Awareness | C | 2 | 20 | 3 | 30 | 4 | 40 |
| 2 | ***Collaboration Aware Tools*** | *Basic* | Support of Workspace Awareness | C | 4 | 40 | 2 | 20 | 5 | 50 |
| 3 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of a Second Pointer for the Navigator | C | 0 | 0 | 0 | 0 | 5 | 50 |
| 4 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | M | 0 | 0 | 0 | 0 | 4 | 10 |
| 5 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Suppression of Telepointer Jitter | L | 0 | 0 | 0 | 0 | 4 | 4 |
| 6 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Support of Highlighting with Respect to Collaboration | H | 3 | 15 | 1 | 5 | 3 | 15 |
| 7 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Floor Control | C | 0 | 0 | 3 | 30 | 0 | 0 |
| 8 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Role Changes | C | 0 | 0 | 4 | 40 | 0 | 0 |
| 9 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Informal Role Changes (social Floor control) | M | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Formal Role Changes (application/master-user mediated Floor control) | M | 0 | 0 | 4 | 10 | 0 | 0 |
| 11 | ***Collaboration Aware Tools*** | *Floor Control Support* | Easiness of Role Changes | M | 0 | 0 | 5 | 12,5 | 0 | 0 |
| 12 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is in which role" | M | 0 | 0 | 4 | 10 | 0 | 0 |
| 13 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Speech Channel | H | 0 | 0 | 0 | 0 | 5 | 25 |
| 15 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of textual Chat | M | 0 | 0 | 5 | 12,5 | 5 | 12,5 |
| 16 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a White-Board | M | 0 | 0 | 2 | 5 | 3 | 7,5 |
| 17 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Video Channel | L | 0 | 0 | 0 | 0 | 5 | 5 |
| 18 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Pair/Partner Finding Support | L | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Microsoft Windows | M | 1 | 2,5 | 1 | 2,5 | 1 | 2,5 |
| 20 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Unix/Linux | M | 1 | 2,5 | 2 | 5 | 0 | 0 |
| 21 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of MacOS | L | 1 | 1 | 1 | 1 | 0 | 0 |
| 22 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of tool usage | H | 4 | 20 | 4 | 20 | 5 | 25 |

| Property-ID | Tool Category | Property Category | Property | importance | MoonEdit | MoonEdit (weighted) | PEP - Pair Eclipse Programming | PEP - Pair Eclipse Programming (weighted) | TalkAndWrite plug-in for Skype | TalkAndWrite plug-in for Skype (weighted) |
|---|---|---|---|---|---|---|---|---|---|---|
| 23 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Installation and configuration | L | 4 | 4 | 4 | 4 | 5 | 5 |
| 24 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Start up and establishing a session | M | 5 | 12,5 | 2 | 5 | 5 | 12,5 |
| 25 | ***Collaboration Aware Tools*** | *Usability* | Stability | H | 5 | 25 | 3 | 15 | 3 | 15 |
| 26 | ***Collaboration Aware Tools*** | *Usability* | Screen Actuality - Realtime-Probability | C | 5 | 50 | 5 | 50 | 4 | 40 |
| 27 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Collaboration | C | 5 | 50 | 3 | 30 | 3 | 30 |
| 28 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of User interface | M | 3 | 7,5 | 4 | 10 | 5 | 12,5 |
| 29 | ***Collaboration Aware Tools*** | *Usability* | Support of Syntaxhighlighting | L | 0 | 0 | 5 | 5 | 0 | 0 |
| 30 | ***Collaboration Aware Tools*** | *Usability* | Need for other tools | H | 3 | 15 | 4 | 20 | 4 | 20 |
| 31 | ***Collaboration Aware Tools*** | *Usability* | Support of Session Management | M | 3 | 7,5 | 1 | 2,5 | 1 | 2,5 |
| 32 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Commercial | L | 0 | 0 | 0 | 0 | 1 | 1 |
| 33 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Open Source | L | 1 | 1 | 1 | 1 | 0 | 0 |
| 34 | ***Collaboration Aware Tools*** | *Tool* | Special Purpose | L | 0 | 0 | 1 | 1 | 0 | 0 |
| 35 | ***Collaboration Aware Tools*** | *Tool* | IDE | L | 0 | 0 | 1 | 1 | 0 | 0 |
| 36 | ***Collaboration Aware Tools*** | *Tool* | Collaborative Editor | L | 1 | 1 | 0 | 0 | 1 | 1 |
| 37 | ***Collaboration Aware Tools*** | *Tool* | Configuration Management/Versioning Support | L | 0 | 0 | 5 | 5 | 0 | 0 |
| 38 | ***Collaboration Aware Tools*** | *Documentation* | Documentation for Installation und configuration | L | 3 | 3 | 0 | 0 | 5 | 5 |
| 39 | ***Collaboration Aware Tools*** | *Documentation* | User Manual | L | 4 | 4 | 4 | 4 | 5 | 5 |
|  |  |  |  |  | 58 | 281,5 | 84 | 357 | 91 | 396 |

*Table 19. Evaluation Results: Collaboration-Aware Applications (3)*

| Property-ID | Tool Category | Property Category | Property | importance | XPairtise | XPairtise (weighted) |
|---|---|---|---|---|---|---|
| 1 | ***Collaboration Aware Tools*** | *Basic* | Support of Collaboration Awareness | C | 5 | 50 |
| 2 | ***Collaboration Aware Tools*** | *Basic* | Support of Workspace Awareness | C | 5 | 50 |
| 3 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of a Second Pointer for the Navigator | C | 0 | 0 |
| 4 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Availability of Semantic Adaptations of Pointers with Respect to Collaboration | M | 0 | 0 |
| 5 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Suppression of Telepointer Jitter | L | 0 | 0 |
| 6 | ***Collaboration Aware Tools*** | *Support of Gesturing* | Support of Highlighting with Respect to Collaboration | H | 4 | 20 |
| 7 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Floor Control | C | 5 | 50 |
| 8 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Role Changes | C | 4 | 40 |
| 9 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Informal Role Changes (social Floor control) | M | 0 | 0 |
| 10 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of Formal Role Changes (application/master-user mediated Floor control) | M | 5 | 12,5 |
| 11 | ***Collaboration Aware Tools*** | *Floor Control Support* | Easiness of Role Changes | M | 5 | 12,5 |
| 12 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is in which role" | M | 5 | 12,5 |
| 13 | ***Collaboration Aware Tools*** | *Floor Control Support* | Support of the Information "Who is how long in which role" along with Statistical Information of Time Spent in each Role | L | 0 | 0 |
| 14 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Speech Channel | H | 0 | 0 |
| 15 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of textual Chat | M | 5 | 12,5 |
| 16 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a White-Board | M | 3 | 7,5 |
| 17 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Support of a Video Channel | L | 0 | 0 |
| 18 | ***Collaboration Aware Tools*** | *Communication Possibilities* | Pair/Partner Finding Support | L | 3 | 3 |
| 19 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Microsoft Windows | M | 1 | 2,5 |
| 20 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of Unix/Linux | M | 2 | 5 |
| 21 | ***Collaboration Aware Tools*** | *Platform Independence* | Support of MacOS | L | 1 | 1 |
| 22 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of tool usage | H | 4 | 20 |
| 23 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Installation and configuration | L | 3 | 3 |

| Property-ID | Tool Category | Property Category | Property | importance | XPairtise | XPairtise (weighted) |
|---|---|---|---|---|---|---|
| 24 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Start up and establishing a session | M | 4 | 10 |
| 25 | ***Collaboration Aware Tools*** | *Usability* | Stability | H | 5 | 25 |
| 26 | ***Collaboration Aware Tools*** | *Usability* | Screen Actuality - Realtime-Probability | C | 4 | 40 |
| 27 | ***Collaboration Aware Tools*** | *Usability* | Easiness of Collaboration | C | 5 | 50 |
| 28 | ***Collaboration Aware Tools*** | *Usability* | Intuitiveness of User interface | M | 4 | 10 |
| 29 | ***Collaboration Aware Tools*** | *Usability* | Support of Syntaxhighlighting | L | 5 | 5 |
| 30 | ***Collaboration Aware Tools*** | *Usability* | Need for other tools | H | 4 | 20 |
| 31 | ***Collaboration Aware Tools*** | *Usability* | Support of Session Management | M | 1 | 2,5 |
| 32 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Commercial | L | 0 | 0 |
| 33 | ***Collaboration Aware Tools*** | *Tool* | Off-the-shelf - Open Source | L | 1 | 1 |
| 34 | ***Collaboration Aware Tools*** | *Tool* | Special Purpose | L | 1 | 1 |
| 35 | ***Collaboration Aware Tools*** | *Tool* | IDE | L | 1 | 1 |
| 36 | ***Collaboration Aware Tools*** | *Tool* | Collaborative Editor | L | 1 | 1 |
| 37 | ***Collaboration Aware Tools*** | *Tool* | Configuration Management/Versioning Support | L | 5 | 5 |
| 38 | ***Collaboration Aware Tools*** | *Documentation* | Documentation for Installation und configuration | L | 3 | 3 |
| 39 | ***Collaboration Aware Tools*** | *Documentation* | User Manual | L | 1 | 1 |
|  |  |  |  |  | 105 | 477,5 |

*Table 20. Evaluation Results: Collaboration-Aware Applications (4)*