JDP: A TOOL TO SUPPORT PAIR PROGRAMMING IN DISTRIBUTED ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

HASAN TURAN KARAPINAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JULY 2005

Approval of the Graduate School of Informatics

_____

Assoc. Prof. Nazife BAYKAL

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Onur DEMİRÖRS

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Semih BİLGEN

Supervisor

Examining Committee Members

Assoc. Prof. Onur DEMİRÖRS (METU, IS) _____

Prof. Dr. Semih BİLGEN (METU, EE) _____

Dr. Ali ARIFOĞLU (METU, IS) _____

Dr. Cüneyt F. BAZLAMAÇCI (METU, EE) _____

Dr. Altan KOÇYİĞİT (METU, IS) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Hasan Turan KARAPINAR

_____

ABSTRACT

JDP: A TOOL TO SUPPORT PAIR PROGRAMMING IN DISTRIBUTED ENVIRONMENTS

Karapınar, Hasan Turan
M.S., Department of Information Systems
Supervisor: Prof. Dr. Semih BİLGEN

July 2005, 57 pages

This thesis focuses on the development of a distributed pair programming tool that enables two programmers to generate code together in JBuilder editor over the web. First, software development processes are generally reviewed and Extreme Programming, Distributed Extreme Programming, and Distributed Pair Programming issues are examined. The tools that enable Distributed Pair Programming are compared. This thesis also specifies the functional requirements of the newly presented tool and includes information about its design and implementation processes. Finally, an evaluation is given by indicating the positive and negative sides of the tool.
Keywords: JDP, Distributed Pair Programming, JBuilder plug-in, Distributed Extreme Programming.

ÖZ

JDP: DAĞITIK ORTAMDA EŞLİ PROGRAMLAMAYI DESTEKLEYEN BİR ARAÇ

Karapınar, Hasan Turan
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Semih Bilgen

Temmuz 2005, 57 sayfa

Bu tez, dünyanın farklı yerlerinde bulunan iki programcının JBuilder editörünü kullanarak eşli programlama yapabilmesine imkan sağlayan bir aracın geliştirilmesini hedeflemiştir. Başlangıçta, yazılım geliştirme süreçleri genel olarak incelenmiş, Uç Programlama (XP), Dağıtık Uç Programlama (DXP) ve Dağıtık Eşli Programlama (DPP) konuları ele alınmıştır. Dağıtık Eşli Programlama yapmaya olanak veren diğer araçlar/ programlar değerlendirilmiş ve karşılaştırılmıştır. Geliştirilen yazılımın gereksinimleri vurgulanmış, tasarım ve geliştirilme süreçleriyle ilgili bilgiler verilmiştir. Son olarak, sunulan aracın olumlu ve olumsuz yanları belirtilerek genel değerlendirmesi yapılmıştır.

Anahtar Kelimeler: JDP, Dağıtık Eşli Programlama, JBuilder program eklemesi, Dağıtık Uç Programlama.

To my wife Yasemin and daughter Çağla,

ACKNOWLEDGEMENTS

I first thank my advisor Prof. Dr. Semih BİLGEN for providing guidance, encouragement and patience, which promoted this study. I am grateful to Prof. Dr. Semih BİLGEN for his inspiration and desire.

I am forever indebted to my wife, Yasemin, for her support and endless patience. I express exceptional thanks to Mustafa KUBİLAY for his technical assistance and ideas.

Finally, words cannot truly express my deepest gratitude to my mother and father who always gave me their love and emotional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

API:	Application Programming Interface
CASE:	Computer Aided Software Engineering
CMM:	Capability Maturity Model
DPP:	Distributed Pair Programming
DXP:	Distributed Extreme Programming
IDE:	Integrated Development Environment
JDP:	JBuilder DPP Plug-in
MILOS:	Minimally Invasive Long-term Organizational Support
MS:	Microsoft
RPP:	Remote Pair Programming
SCM:	Software Configuration Management
TCP/IP:	Transmission Control Protocol / Internet Protocol
VNC:	Virtual Network Computing
XP:	Extreme Programming

CHAPTER 1

Introduction

Agile software development methods are becoming more popular day by day because of their different approach to software engineering compared to classical methods. Extreme Programming (XP) is the leading one among agile methodologies.

XP has twelve rules/ key practices one of which is pair-programming. It advocates the idea that in order to increase the quality of the software, two programmers should generate code side by side at one computer. Some advantages of pair programming are listed as economics, design quality, and satisfaction according to an experiment at the University of Utah [Cockburn and Williams, 2000]. The results show that the percentage of increase in time (15%) of the pairs is the same as the percentage of the decrease in the number of defects compared to individual programmers. Since the cost of fixing defects is high, pair programming is economic. Second advantage, better design, is indicated by fewer lines of code. The high satisfaction of the programmers is also understood from the interviews.

"Increasingly, programmers are working in geographically distributed teams. The trends toward teleworking, distance education, and globally distributed organizations are making these distributed teams an absolute necessity. These trends are beneficial in many ways, particularly for those in geographically disadvantaged areas [Baheti, 2002]." The demand of virtual teams, whose members work from different places, about using pair programming due to its effectiveness led to born "Distributed Pair Programming-DPP". It is defined in [Baheti, 2002] as "collaborate on the same design or code, but from different locations".

1.1 Problem Statement

This study aims to develop a practical tool to support distributed pair programming. While this may constitute the core of a general Distributed Extreme Programming (DXP) tool, our aim was confined to pair programming only. Although there are other professional tools to enable Distributed Pair Programming (DPP), JBuilder Distributed Pair Programming Plug-in (JDP) has some advantages due to being a plug-in to JBuilder.

The dilemma in distributed pair programming is that the commercial DPP tools do not have software development capabilities as much as the commercial software development environments and the software development tools do not have extended distributed pair programming facilities.

According to the pair programming concept, the two programmers, driver and navigator should be responsible for different issues, i.e. the driver for writing code and the navigator for observing the larger picture and evaluating the driver's code [Baheti, 2002]. So, the navigator should see the written code of the driver, but should not intervene, and should be able to guide the driver. Another issue is

the ability of working locally and trying two different ideas at their own computers without loosing time.

The research problem tackled in this study is the development of a practical tool to support this fundamental need. As this is intended simply as an “illustration of concept” study, rather than a field evaluation of the developed tool by usage in professional environments, a straightforward comparison of the goals and the implemented facilities has been attempted in lieu of a formal validation exercise.

1.2 Thesis Structure

This study is composed of five chapters:
Chapter 2 presents research about software development processes, extreme programming, distributed extreme programming, pair-programming, distributed pair programming and tool support for DXP/ DPP.

In Chapter 3, the purpose, scope and functional requirements of JDP are presented; the design is documented, implementation and restrictions are described.

Chapter 4 consists of an evaluation of JDP. The positive and negative sides of it are examined and a comparison between JDP and other evaluated tools is made. As the fundamental aim of the study is simply to present a working, albeit restricted, tool to support the needs of DPP, the evaluation consists of a comparison of the needs stated in Chapter 3 with the achievements realized by the implemented tool.

In Chapter 5, as a conclusion, a general discussion about JDP is presented and some clues about improving JDP as a future work are given.

CHAPTER 2

Related Research

This chapter reviews the main issues about software development processes, extreme programming, distributed extreme programming, pair-programming, distributed pair programming and tool support for DXP/ DPP. At the end of the chapter, a comparison table of the examined tools is given.

2.1 Software Development Processes

In the early years of computing, software was quite small, generally developed by one person to use himself. At those times, there were no methodologies except code-and-fix. The crisis in software industry led developers to handle software by more sophisticated methodologies.

Sorensen observes that the Waterfall Model was the first documented life cycle model. This model emphasizes completing a phase of the development before proceeding to the next phase [Sorensen, 1995]. It is formal and a type of top-down development. The steps of waterfall model can be listed as:

1. feasibility study
2. requirements analysis and specification
3. design and specification
4. coding and module testing
5. integration and system testing
6. delivery and maintenance [Scacchi, 2001]

The waterfall model can be effective only in the situations that the requirements are clear and well understood.

The need of using the software as soon as possible and saving time led to another methodology: incremental development. In this model, “the overall requirements of the final system or product are known at the start of the development, however a limited set of requirements is allocated to each increment and with each successive (internal) release more requirements are addressed until the final (external) release satisfies all requirements” [Wallin,2002]. We can simply explain this method as follows: If the whole software is a circle, for example, in the first increment, the first quarter is released, and so on. The quarter is working software, but only a small part of the whole. After four increments, the circle is established, that is, the software is completed.

Above two methods can only be used if the requirements are known from the beginning. For the projects in which the requirements are not known definitely or change rapidly, a new model is more suitable: evolutionary model. In this model, “initially the requirements are partly defined and then refined and extended with each successive (internal) release as the requirements picture mature and until a satisfactory solution is reached” [Wallin,2002]. Usually, prototypes are used in order to discard misunderstandings. This model works like an artist, making a statue from rock and showing the result to the customer every day and getting feedback until it finishes.

Another model is the spiral model proposed by Boehm, in 1988. “The spiral model uses concepts of prototyping and evolutionary system implementation to primarily identify and evaluate risk and cost” [Davidson, 2002]. “This model divides the development activities into four quadrants through which the effort proceeds. Each time a quadrant is visited, the scope is increased based on go/no-go decisions made in the previous efforts. Thus, an expanding spiral effect that finally leads to a deliverable system is created” [Davidson, 2002].

Agile software development methods emerged as a reaction to traditional ways of developing software. In [Beck, et. al., 2001], it is admitted that there is a “need for an alternative to documentation driven, heavyweight software development processes”. Jim Highsmith defines being agile as being able to “Deliver quickly. Change quickly. Change often” [Highsmith, et. al., 2000]. Some of the agile programming methods are: Extreme Programming, Scrum, Crystal Methods, Feature Driven Development, Lean Development, and Dynamic Systems Development Methodology [Cohen, 2003]. One of the most popular agile methods, Extreme Programming, is explained in detail in Section 2.2.

The demand for measuring the level of assurance of software quality provided by developer organizations led to the Capability Maturity Model, becoming a de-facto standard in this area. This model is “a five-level model that describes good engineering and management practices and prescribes improvement priorities for software organizations” [Paulk, 2001]. The model defines some key process areas and goals for organizations in order to classify them. For being in Level-5, that is called “Optimized”, an organization should succeed in 18 key process areas and 52 goals. According to [Cohen, 2003], most of the software organizations in the world are in CMM Level-1, which is called Chaotic and only a few are Optimized (CMM level 5). CMM is focused on the processes and procedures instead of the software produced. The importance given to procedures and

documentation of CMM was criticized in 1990's as being too formal and for that reason being big and slow by some software engineers who later declared a new approach: "agile" [Boehm, 2002], [Cohen, 2003], [Paulk, 2001], [Paulk,2002], [Glazer, 2001], [Highsmith, 2002b].

2.2 Extreme Programming

Extreme Programming is the most popular Agile Method which emerged in recent years. "Introduced by Beck, Jeffries, et. al., in 1998 [The C3 Team, 1998] and further popularized by Beck's 'Extreme Programming Explained: Embrace Change' in 1999 and numerous articles since, XP owes much of its popularity to developers disenchanted with traditional methods [Highsmith, 2002] looking for something new, something extreme" [Cohen, 2003].

Extreme Programming is most suitable for small (generally 2-10) and co-located teams. [www.extremeprogramming.org] The more members of the team after ten, the more need for communication and the coordination, which will have a negative effect on the project.

XP is usually considered most suitable for projects whose requirements are not clearly defined or changing continuously.

XP methodology emphasizes team work and customer involvement throughout software development. Communication and feedback are main points in Extreme Programming. Communication is advocated not just between the customer and the analyst but also between the customer and the developers and within the development team. Feedback is achieved through early and continual testing. "Extreme Programming is asserted that its benefits include faster time to market, higher quality software, better customer satisfaction, and highly motivated development teams" [Smith, 2001].

Extreme Programming has 12 rules/ key practices [Cohen, 2003]:

The Planning Game: At the start of each iteration, customers, managers, and developers meet to flesh out, estimate, and prioritize requirements for the next release. The requirements are called “user stories” and are captured on “story cards” in a language understandable by all parties.

Small Releases: An initial version of the system is put into production after the first few iterations. Subsequently, working versions are put into production anywhere from every few days to every few weeks.

Metaphor: Customers, managers, and developers construct a metaphor, or set of metaphors after which to model the system.

Simple Design: Developers are urged to keep design as simple as possible, “say everything once and only once” [Beck, 1999].

Tests: Developers work test-first; that is, they write acceptance tests for their code before they write the code itself. Customers write functional tests for each iteration and at the end of each iteration, all tests should run.

Refactoring: As developers work, the design should be evolved to keep it as simple as possible.

Pair Programming: Two developers sitting at the same machine write all code.

Continuous Integration: Developers integrate new code into the system as often as possible. All functional tests must still pass after integration or the new code is discarded.

Collective ownership: The code is owned by all developers, and they may make changes anywhere in the code at anytime they feel necessary.

On-site customer: A customer works with the development team at all times to answer questions, perform acceptance tests, and ensure that development is progressing as expected.

40-hour Weeks: Requirements should be selected for each iteration such that developers do not need to put in overtime.

Open Workspace: Developers work in a common workspace set up with individual workstations around the periphery and common development machines in the center.

Together with the twelve key practices, XP methodology is based on four principles: Communication, simplicity, feedback and courage. In XP methodology, the rules and practices must support in each other.

A general view of XP project is in the Figure 1.

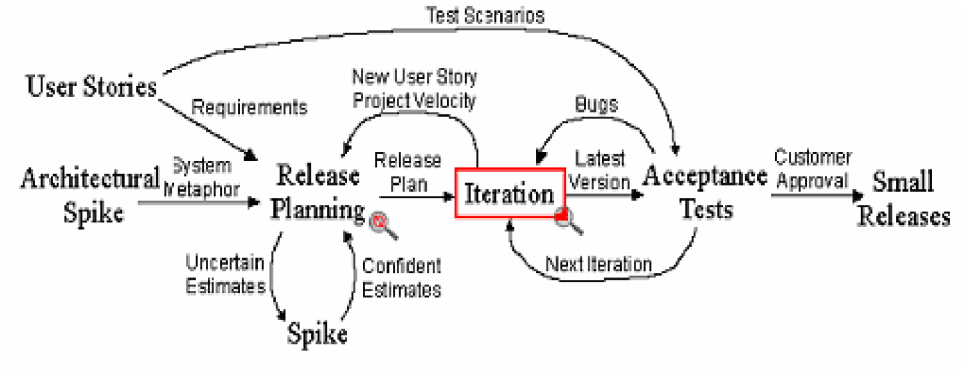

Figure 1 XP Project Model [www.extremeprogramming.org]

User stories are written by the customers. They are scenarios declared in natural rather than technical language. Developers estimate how long the stories might take to implement. Each story will get a 1-3 week estimate under normal circumstances. Then a release plan is created. Acceptance tests are created from user stories. These tests ensure the functionality of the software developed. At this phase, customers should participate in the process actively by verifying the correctness of the tests. Small releases become outputs of iterations after customer approval.

Despite the fact that XP has some principles and key practices, it is not a static but adaptive methodology. Moreover, in [Hayes, 2001], it is claimed that no XP projects are expected to look exactly the same.

2.3 Distributed Extreme Programming

XP advocates a strong level of communication among team members and pair programming, both of which requires physically being near and in the same location, ideally in the same room. However, due to individual constraints, cost, mobility, convenient customer involvement, or other reasons, it may not be feasible to have team members physically located close to each other. Kircher, in [Kircher, et.al, 2001], proposed a customized version of XP called Distributed Extreme Programming (DXP). He defines DXP as "Extreme Programming with certain relaxations on the requirements of close physical proximity of the team members" [Kircher, et.al, 2001]. DXP accepts all the XP principles but propose to use XP in distributed and mobile team environments.

In order not to loose effectiveness in DXP compared to XP, such tools and communication devices should be established that not only the team members and pairs, but also the customer representative should not hesitate to communicate.

DXP has some assumptions beyond the assumptions of XP. These are shortly: connectivity, e-mail, configuration management, application sharing, video conferencing, and familiarity [Kircher, et.al, 2001].

Kircher classifies the key practices of XP (defined in 2.2) whether they require co-located team or not. Four key practices, planning game, pair programming, continuous integration and on-site customer, are affected from the location. So,

these four practices should be handled and customized. Kircher handles them as follows[Kircher, et.al, 2001]:

Planning Game - For the planning game with the customer being remote, video conferencing and application sharing software support is needed. Ideally more than two participants should be supported.

Pair Programming - For pair programming between team members in different locations, Remote Pair Programming (RPP) should be used. This requires video conferencing and application sharing support, to share the Integrated Development Environment (IDE).

Continuous Integration - Because a remote team member cannot move to a separate integration machine, a workaround needs to be defined. If one team member is working at the central team site, he/she can invite the other remote team member to do common integration at that machine.

On-site Customer - Video conferencing should be used to involve remote customers.

The DXP faces some challenges because of not being co-located. These are:

a. Communication: The pair shall not be able to read the body language like gestures, the face, and the voice.
b. Coordination: Synchronizing availability, adjusting time differences, and coordinating distribution may cause some problems.
c. Infrastructure: Available software and hardware, bandwidth of connecting network is a must for DXP.
d. Availability: Distributed members may be available at different times.
e. Management: Trust of the senior manager must be in high level [Kircher, 2001].

One of the main features of DXP is pair programming from different locations. The “pair programming” key practice of XP is now called “Remote Pair Programming-RPP” or “Distributed Pair Programming-DPP” in DXP. Pair-

programming and distributed pair programming (DPP) is focused in the next two parts.

2.4 Importance of Pair-Programming

"Pair programming is a style of programming in which two programmers work side by side at one computer, continuously collaborating on the same design, algorithm, code or test" [Baheti, 2002]. "One of them, called the driver, is in control of the keyboard and mouse. The other, called the navigator, observes what the driver is doing and offers advice. It is the driver's job to write the code. The navigator has a chance to observe the larger picture, evaluating the driver's code for correctness of design and implementation" [Gehringer, 2003]. Briefly, the driver is responsible for writing more; the navigator is responsible for thinking more. They unite their powers by brainstorming at any time. The pairs exchange their jobs at certain time intervals. In an effective pair-programming relationship, pairs should be active, that is they should communicate from 45 seconds to 1 minute [Baheti, 2002].

Researchers have indicated that pair programming is better than individual programming in a collocated environment [Williams, 2000], [Nosek, 1998]. "It has been observed that the code resulting from pair programming is more defect free, does not take significantly more time to develop than if developed by one member, yields fewer lines of code, and is more satisfying to programmers" [Cockburn and Williams, 2000]. Advantages and disadvantages, main concepts, best practices, and practical advice to successful pair programming are discussed in a paper based on an experiment at the University of Utah where one third of the class developed the projects individually and the rest developed in pairs. The results were analyzed from the point of views of economics, satisfaction, and design quality [Cockburn and Williams, 2000]:

Economics: The results showed that the pairs only spent 15% more time to program than the individuals and the code produced by pairs had 15% fewer defects. Thus, pair programming can be justified purely on economic grounds since the cost of fixing defects is high.

Satisfaction: Results from interviews with individuals who tried pair programming were analyzed. Although some were skeptical and did not feel comfortable at first, most programmers enjoyed the experience.

Design quality: In the Utah study, the pairs not only completed their projects with better quality but also implemented the same functionality in fewer lines of code. This is an indication of better design.

Other benefits of pair programming are continuous reviews, problem solving, learning, and staff and project management [Cockburn and Williams, 2000]:

Continuous reviews: Pair programming serves as a continual design and code review that helps the removal of defects.

Problem solving: The teams found that, by developing in pairs, they had the ability to solve problems faster.

Learning: The teams emphasized how much they learned from each other by doing pair programming. Pair programmers often mention that they also learned to discuss and work together, improving team communications and effectiveness. "Extreme Programming demonstrates that programmers can not only work well in pairs but also learn much from discussing code and design with another person" [Smith, 2001].

Staff and project management: From the staff and project management point of view, since people are familiar with each piece of code, staff-loss risks are reduced.

In an organization, the issue of how much training is required is also important while deciding which methodology to use. “Pair programming helps minimize what is needed in terms of training, because people mentor each other. This kind of mentoring (by some referred to as tacit knowledge transfer) is argued to be more important than explicit training” [Cohen, 2003].

Many researchers [Williams et al, 2000], [Cockburn and Williams, 2000], [Kalermo, 2002], [Cohen, 2003] emphasize one point: It is hard to convince some programmers to do pair programming at the beginning. After convincing, it is observed that programmers were not comfortable at first times. But later, they were used to the new methodology and even liked it, started to work naturally and efficiently with his/her partner.

The pair programming concept is becoming more popular as the positive sides emerge and the effectiveness is proved.

2.5 Distributed Pair Programming

“Increasingly, programmers are working in geographically distributed teams. Escalating trends in teleworking, distance education, and globally distributed organizations are making these distributed teams an absolute necessity” [Baheti, 2002]. What if programmers working in geographically different places want to have the benefits of pair- programming? Surely they can use pair-programming, but with the help of some technical facilities. At least, they should be able to see the same screen and share comment in a conversation environment.

By using either the tools presented in Section 2.6 or some other tools, it can be said that pair programming is possible, but is it as effective as co-located pair programming? In one research, [Canfora, 2003], it was concluded that only screen sharing applications without audio or video support does not work,

because after some time distributed pairs tended to stop cooperating and began to work as two single programmers. In [Baheti, 2002], the results of another research about distributed pair programming are presented. According to it, “Collocated teams did not produce statistically significantly better results than the distributed teams” and “the experiment is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects”.

With the objective of providing a catalyst for the methodology, this thesis and the presented tool is focused on Distributed Pair Programming - DPP.

2.6 Tool Support for DXP/ DPP

In this part, the tools that support Distributed Extreme Programming or Distributed Pair Programming are focused.

The tools that are supposed to be used in DXP projects should be easy to use and integrate in the working environment. They could also be supported on multiple platforms. A video conferencing and application sharing software should be used. In the hardware side, every computer should have a microphone, speakers, and a web cam. An internet connection (as fast as possible) should be established.

DXP tools are generally video conferencing & application sharing tools and distributed programming tools as seen in Table 1. Ideally, the tools supposed to support DXP should have the following properties:

1. Project coordination support,
2. Synchronous communication,
3. Active notifications and information routing,
4. Integrate process execution with knowledge management: [Maurer, 2002]

Table 1 DXP Tool Types and Examined Tools.

Tool Type	Tool Name
Distributed programming tools	MILOS, VNC, ClearCase, TUKAN, MASE
Video conferencing & application sharing tools	CUseeMe, NetMeeting.

Below, these tools will be described and comparatively evaluated with the aim of determining the requirements for the tool to be developed within the framework of this study.

2.6.1 MILOS:

MILOS is a process-support environment that helps software development teams to maintain XP practices in a distributed setting. MILOS stands for "Minimally Invasive Long-term Organizational Support". The overall goal of the MILOS approach is to support process execution and organizational learning for virtual software development teams [Bowen, Maurer, 2002].

MILOS is Web-based and accessible as a web service on the MILOS web site (http://sern.ucalgary.ca/~milos) from any machine connected to the Internet using a standard Web browser. Team members are allowed to use facilities after login to the system. Users may retrieve the list of current projects, user stories, currently available tasks, task estimation, and pair programming facilities.

Release & iteration planning is handled as follows [Maurer, Martel, 2002]: After the creation of an initial project and the assignment of a project manager to it, the

customer is supposed to enter story cards into the MILOS system. The programmers can then contact the customer, either through the MILOS framework or by conventional means, and discuss the story with them if needed. They can revise the description of the user story – creating a new version of the existing card. In addition, they can then add notes to the story card pertaining to implementation details and split up the story into several smaller stories if the scope is too large. They would then proceed to decompose the story into specific programming task that will be needed to satisfy the next build. The workflow engine of MILOS handles the creation and changes of tasks. For each user story, the MILOS system automatically creates a top-level process “Design and implement user story”. The input of this process is the newly created user story. Then the programmer can decompose the user story into smaller and more concrete tasks.

After having decomposed user stories into concrete programming task, the programmer may describe the task in more detail using the workflow engine user interface. Tasks are associated with specific projects and can be assigned to various team members. For each task, the manager enters planned start and end dates. In addition, the users are able to define the inputs and outputs of processes. The story card automatically becomes an input of all subtasks. Furthermore, the users are able to specify the information flow between tasks by defining the output of one process to become the input of another.

Specifying the information flow allows the MILOS system to provide access to input information that was created as the output of another task: the output of a task, e.g. a source code file, is transferred to the MILOS server and stored in a version management system. From there, any successor task may access the current version as well as older versions.

The programmer is able to estimate the effort and the forecasted end dates. The effort simply consists of the total number of workdays needed to complete the task (measured in ideal engineering time). To set the forecasted end date, the developer takes the required effort as well as his overall workload into account.

Synchronous communication is enabled via NetMeeting. Using Microsoft NetMeeting, MILOS provides an audio and video link between two developers and the ability to share the desktop between them. These capabilities are used to support pair programming in a distributed setting. The MILOS system keeps track of who is logged on to the system and provides the possibility to contact the responsible team member for a task or any other team member that is currently logged in. [Maurer, Martel, 2002], [Bowen, Maurer, 2002].

2.6.2 VNC:

This tool is based on the open source screen sharing application Virtual Network Computing (VNC). VNC allows a user's desktop to be replicated onto multiple computers (in particular, two in the case of pair programming). Application output is displayed on both computers, while keyboard and mouse input from either computer is sent to the applications.

Being "screen-sharing" brings the advantage of sharing single user applications by multiple users without modification. It is critical for a distributed pair of developers to be able to use their preferred development tools, and the screen sharing approach supports this requirement [Hanks, 2004].

VNC is not designed for DXP, but only supports DPP, distributed pair programming. Therefore, release and iteration planning facilities are not available. It enables synchronous communication in pair-programming via "Voice over IP" applications (like Messenger).

Despite the fact that VNC can be used as a DPP tool, it is not ideal for this. “Both users have active keyboards and mice. If both partners use the keyboard simultaneously, their keystrokes are interlaced into an unintelligible stream. There is also only one cursor, which makes it difficult for the navigator to point at areas of the screen” [Hanks, 2004]. But this disadvantage is tried to be overcome by designing a gesture mode, so the cursors are managed.

2.6.3 ClearCase:

ClearCase is a Software Configuration Management (SCM) system. “That means it keeps track of which versions of which files were used for each release (even internal releases) and also which combinations were used in builds (including engineer builds, nightly and release builds). ClearCase also provides a rich and robust set of tools to allow a company to craft their engineering environment to their own requirements”[www.timefold.com/atria/10quest.html]

Every programmer on an XP project must be able to change code and ensure that it works, not just for unit tests but also for acceptance tests. This requires frequent builds. In order for it to really work, powerful configuration management tools are needed. The ClearCase provides general guidelines for continuous integration. [http://www.therationaledge.com/content/mar_01/f_xp_gp.html]. The "anyone can change anything" philosophy in XP makes a configuration management program like ClearCase necessary, especially in distributed environments. ClearCase supports distributed development (including distributed pair programming) by managing the versions of the documents [Kircher, 2001].

Due to the fact that ClearCase is designed for managing versions of the software, the using area of it is not limited to XP or pair-programming. Therefore, the planning issues and synchronous communication facilities of XP are not available

in ClearCase. But it should not be forgotten that other synchronous/ asynchronous communication tools (like e-mail, chat, Messenger, etc.) can be used separately.

2.6.4 TUKAN:

TUKAN is “a synchronous distributed team programming environment, which applies groupware research results to the XP domain and solves the problems which arise when XP is carried out by distributed teams” [Schümmer& Schümmer, 2001].

TUKAN is an enhancement of VisualWorks / ENVY Smalltalk. ENVY provides a shared code repository with a sophisticated distributed version management and simple user management for code ownership. TUKAN adds awareness information, communication channels, and synchronous collaboration mechanisms to ENVY. It uses an extended model to arrange the software artifacts in a virtual environment.
Unlike application sharing systems like NetMeeting, TUKAN works on shared replicated objects and requires only a low network bandwidth [Schümmer& Schümmer, 2001].

Teams are supported in the planning activity by the activity explorer. It may be used simultaneously by programmers and customers to play the planning game in a distributed setting.

“TUKAN combines many of the techniques:
TUKAN provides basic communication support (chat and audio).
TUKAN is a coordination system since it helps to plan and co-ordinate activities in the planning game.
TUKAN integrates a version management system (ENVY).

In supporting the planning game, TUKAN is also a group decision support system.
TUKAN includes multi-user editors for code editing and for annotating story cards in the planning game" [Schümmer& Schümmer, 2001].

TUKAN supports Distributed Pair Programming, not only with one person, but also by inviting other people to the environment. "TUKAN uses a separate browser to provide additional information about what change caused the conflict or who is currently working nearby. If the work of another user is of interest to a programmer's task, she can decide to switch to tighter collaboration and enter a pair programming or integration session using the cooperative class browser". One can invite other to join the pair programming session by pressing the invite-Button of the browser and selecting his name from a list of available users. "When accepting the invitation, the author also sees the pair programming browser on his machine and is a partner with equal permissions within the session" [Schümmer& Schümmer, 2001].

2.6.5 MASE:

"MASE is a web-based collaboration and knowledge sharing tool for agile teams. Web technology makes the tool accessible anytime anywhere by users with a web browser in their computing environment. The tool does not distinguish users working at the same place from those who work at different places. Hence, MASE is capable of supporting collaboration for both co-located and distributed teams". [Chau, Maurer, 2004].

MASE enables any users to access, browse, create, structure, and update any web pages in real-time using a web browser only. Each of these web pages, known as a wiki page, acts like an electronic bulletin board discussion topic with a unique name.

MASE supports synchronous work through its integration with the real-time collaboration tool, Microsoft NetMeeting. Every time when a team member logs into MASE, MASE tracks the network address of that team member's computer. Through a plug-in (ListUser), MASE displays which members of a team are currently using the system, thus making all online team members aware of each other's presence. This is important for team members to establish informal and spontaneous communication with one another at ease.

MASE facilitates the following agile practices: release and iteration planning, distributed pair programming, collaborative design, and daily meetings [Chau, Maurer, 2004]

MASE supports agile practices through its library of plug-ins. "Project managers and customers can create iterations and user stories. MASE keeps track of all estimates made by the development team and suggests to both the development team and customers the appropriate size for the next iteration based on the developers' estimation accuracy from the previous iteration. Using the suggested iteration size, customers can prioritize user stories and move them from iteration to iteration or move them back to the product backlog. During the course of the project, both the customers and development team can track work progress at various granularities (project, iteration, user story) using the Whiteboard plug-in and view effort metrics for a particular individual or for the entire team" [Chau, Maurer, 2004].

"Leveraging MASE's integration with NetMeeting, developers can also perform distributed pair programming by sharing their code editor and collaborate on a design together using the shared whiteboard" [Chau, Maurer, 2004]. Using the video and audio conferencing and multi-user text-chat features of NetMeeting,

distributed team members who work at the same time can perform daily meetings.

2.6.6 CUSeeMe and MS NetMeeting:

CUSeeMe is an application that enables videoconferencing. The software was developed by Cornell University and White Pine as an attempt to create affordable and workable desktop videoconferencing. With this program, one can see and talk to someone at the same time. CUSeeMe can be used via web browsers.
[http://www.cortland.edu/flteach/methods/obj1/cuseeme.html].

NetMeeting is a Microsoft product for internet communication with multi-point data conferencing, text chat, whiteboard, and file transfer, as well as point-to-point audio and video [http://www.microsoft.com/windows/netmeeting].
NetMeeting enables distributed pair-programming by program sharing, that is, both sides open an application program, see and even edit the data displayed. [http://www.departments.dsu.edu/disted/netmeeting]

Despite the fact that CUSeeMe and NetMeeting can be used in XP / DXP projects, they are not merely intended for these methodologies, so the planning issues (like release & iteration planning) are not handled. These two tools can be used as a supporting tool in synchronous communication.

2.6.7 Ugur Oksay's DPP Plug-in:

This is not a professional tool as the others mentioned in this section. However, it has served as a starting point for the work within the scope of this study. Oksay [Oksay, 2004] has built an elementary facility that opens a dialog box from JBuilder and transmits the inputs from one side to the other. No specific editor or

JBuilder functionality is shared, but the single character-based transfer facility has provided motivation for our work.

A summary of the comparison of these tools, based on the descriptions and considerations outlined in sections 2.6.1 through 2.6.6 is summarized in Table 2.

Table 2 Distributed Extreme Programming Tools Comparison Summary

Tool Property / Tool Name	Design Architecture	Release & Iteration Planning	Synchronous Communication via Audio & Video	Pair Programming Support
MILOS	Web-based	Enabled, The list of current projects, user stories, currently available tasks and task estimation can be retrieved. The programmers can create and modify user cards.	Enabled by MS NetMeeting	Enabled by MS NetMeeting
VNC	Screen-sharing	Disabled, No planning issue is handled.	Enabled by Voice over IP applications	Supported by synchronizing the screens. The management of active keyboards and mouse is needed.
ClearCase	Software Configuration Management	Disabled, Continuous integration is accomplished only.	Disabled, Communication is not handled within the tool.	Supported by managing the versions of the software.
TUKAN	Shared replicated objects & version management	Enabled, Teams are supported in the planning activity by the activity explorer. It may be used simultaneously by programmers and customers to play the planning game in a distributed setting.	Chat & Audio enabled	Supported, It is not limited to one peer; instead more than two people can share the code. No difference between the navigator and the driver.
MASE	Web-based	Enabled, Project managers and customers can create iterations and user stories.	Enabled by MS NetMeeting	Supported, Displays which members of a team are currently using the system, thus making all online team members aware of each other's presence. Enables sharing the code editor and collaborate on a design together using the shared whiteboard.
CUSeeMe, NetMeeting	Desktop video-conferencing and program sharing.	Disabled, Planning, specifically in the context of XP, is not explicitly handled.	Enabled, Used actively for synchronous communication.	Supported by an editor which is not a compiler.

CHAPTER 3

JBuilder DPP Plug-in (JDP)

In Chapter 3, firstly, the problems of distributed pair programming and the examined tools are specified and in the next part the solutions of these problems are proposed with the purpose and scope of the study. After the functional requirements are listed, the structure of JDP is explained with design and implementation issues. This chapter finishes with the restrictions of the tool

3.1 Problem Statement

The dilemma in distributed pair programming is that; the commercial tools that support DPP including the tools mentioned in Section 2.6 do not have software development capabilities as much as the commercial software development environments (e.g. Borland JBuilder) and the software development tools do not have extended distributed pair programming facilities. MS NetMeeting, Messenger or some other communicating programs can be used in distributed pair programming, but the editors of such programs are generally dummy editors or serve the facilities such as emoticon insertion that programmers seldom need.

One of the two ways to tackle this problem is to extend communicating programs by adding DPP software development capability, while the other one is extending software development environments by adding DPP facility. The approach preferred in this study has been latter, namely, the very popular software development environment, JBuilder, has been extended with a plug-in software module to provide DPP support. Briefly, the technology of software development environments and compilers should merge with distributed pair programming methodology.

The second issue is enabling alternation of the driver and navigator roles in the DPP process. The tools mentioned in Section 2.6 do not give a specific role to the driver and navigator, and authorize them in the same level which is not appropriate in the scope of pair programming. The driver should be responsible for writing the code while the navigator should observe and evaluate the driver's code. The driver and the navigator should be able to exchange their roles in certain time intervals [Baheti, 2002], [Gehringer, 2003]. According to DPP concept, only the driver should type the code, possibly concentrating on low-level details, while the navigator should not have ability and authority to write code so that s/he can possibly ignore low level details while keeping the larger scale design and architecture issues in mind. This role differentiation should be explicit, even when they are not co-located. The main reason of such a restriction is to focus on software from different detail levels in order to prevent defects in the coding phase.

The third issue is that, the driver and the navigator need not only to communicate and share the same code, but also they must be able to work locally. The driver / navigator may want to try a small change in the code which is not needed to be seen by the navigator / driver. Enabling local changes may have benefits like saving time and trying two different ideas by both the navigator and the driver at their own editors without interrupting each other.

The fourth issue is the time wasted in the process. Other candidates for DPP, e.g. MS NetMeeting or Messenger, send the written line by a trigger which is generally a “Send” button or “Enter” key. While the driver edits, the navigator can not see the written line up to the time of executing the trigger, i.e. pressing the “Send” button or “Enter” key. This time includes not only the typing time but also implicitly the “thinking” and “text-arranging” time.

3.2 Purpose and Scope

The purpose of this study is to handle the problems stated in the previous section in the light of the DPP methodology. The software development environment to be augmented for DPP functionality is selected as Borland JBuilder 9.0.

The problem of merging a software development environment with distributed pair programming methodology is handled by adding a DPP plug-in facility to JBuilder. This plug-in enables distributed pair programming between the driver and the navigator who are using two different computers connected to Internet at two different places via a plug-in interface.

The need for differentiating the roles of driver and navigator and the ability to exchange them in certain time intervals is accomplished as follows: Two roles are defined by the system as the driver and the navigator. The sending operation is limited only from driver to navigator, i.e. every action performed by the navigator is local and does not have any effect on the driver editor. Exchanging the roles is accomplished by simply signing out and logging in to the system with the opposite role.

Working locally is handled by differentiating the way of using the facilities, i.e. by mouse or by keyboard. The most frequently used editor functions of the

driver, including cut, copy, paste and selection are replicated at the navigator side when carried out by the driver using keyboard shortcuts such as CTRL-X, CTRL-C, etc. When either side performs these operations using explicitly menu item selections, the operation is carried out locally.

Characters entered by the sender should be replicated at the driver's screen immediately.

3.3 The Summary of the Work Done

At the beginning phase of the study, what kind of a tool shall be developed was specified. Functional requirements of the planned tool were listed.

In the second phase, Oksay's work [Oksay, 2004], which has provided motivation for ours, was examined. This phase has included a meeting with Oksay.

The next phase was to improve Oksay's software which was taking the characters written into a textbox and sending it to the other side without any sequence when pressed a button. Firstly, a sequential sending operation was established. Sequential means sending the words as they are and in "First in First Out" order. At the end of the phase, the output was the software which sends the characters sequentially to the same line at the other side by clicking the "Send" button.

The goal of the next phase was to preserve the line numbers and spacing, that is, to handle the characters like "Enter", "Space" and "Backspace". After it was done, adding the "Send" button to the toolbar was intended, but because of finding a better solution, the route changed to another way; simultaneous sending. In this way, no button would be needed because the send operation should be done at the time of pressing the key.

The next phase was to cancel the text box and write to /read from JBuilder editor instead. Handling the JBuilder editor was a milestone because of beginning to use the local facilities of JBuilder.

In the next iteration, mouse events in the editor were handled. Although all mouse events were handled, only “mouseClicked” event was set to send the cursor position.

In the last phase, Cut, Copy, Paste, Select-All and text selection by mouse were handled. Cut, Copy, Paste and Select-All facilities were attended to known “Control” characters.

The approach taken in implementing this software was close to XP. General XP rules like small releases, simple design, and continuous integration were applied except inapplicable rules like pair-programming. Iterations were divided into small tasks and every week a small release was presented.

3.4 Functional Requirements for JBuilder DPP Plug-in (JDP):

1. The system shall be a plug-in to Borland JBuilder.
2. The system shall work on Microsoft Windows XP operating system.
3. The driver and navigator shall communicate over the Internet, i.e. the software shall operate over TCP/ IP.
4. Users shall be able to login to the system either as the driver or the navigator.
5. The system shall check the versions of the Borland J++ Builder of two sides; continue only if they are identical.
6. The system shall enable navigator and driver to choose the file to modify.

7. The system shall send the alphanumeric characters from the driver to the navigator.
8. The following special characters shall be sent from the driver to the navigator with their same functions:
    a. Backspace: Deletes the previous character.
    b. Enter: Go to the new/ next line.
    c. Delete: Deletes the selected characters.
    d. Space: Puts a blank character.
9. The following editor functions, when performed by the driver, shall be replicated on the navigator's editor.
    a. Select-All: Ctrl-A, selects the whole characters in the editor.
    b. Cut: Ctrl-X, deletes the selected characters and holds it in the clipboard.
    c. Copy: Ctrl-C, holds the selected characters in the clipboard.
    d. Paste: Ctrl-V, puts the characters in the clipboard to the place of the cursor.
10. The system shall send the characters synchronously, that is, at the time the characters are pressed on the keyboard.

3.5 JBuilder DPP Plug-in (JDP) Design and Implementation

JDP uses the OpenTools technology of Borland JBuilder. The OpenTools application programming interface (API) contains the classes and interfaces to extend JBuilder.

The structure of JDP is based on sending the characters and mouse events of the driver JBuilder editor to the navigator JBuilder editor. This sending operation is designed to execute at the time of the occurrence of the event.

JDP is designed as three main classes which are described in next sections in detail and two supporting classes one of which is the EditorStats class to open a new JBuilder screen and the other is an empty class to use as a default editor.

3.5.1 Decomposition Description

The JDP software is developed with one package: OpenToolsAPI. This package includes user and session related classes. Package includes the classes that do the main control, socket management and editor activities.

Package: OpenToolsAPI

Identification	:	OpenToolsAPI
Type	:	Package
Purpose	:	To provide a framework for classes of JDP.
Function	:	Performs customize operations for JDP

3.5.2 Dependency Description

Dependency description involves classes as design entities, and provides details related to classes of OpenToolsAPI package by addressing *identification, type, purpose, function, subordinates, dependencies,* and *resources* design entity attributes [IEEE, Std 1016-1998]

Figure-2, Figure-3, Figure-4, and Figure-5 shows dependencies among classes in form of a class diagram.

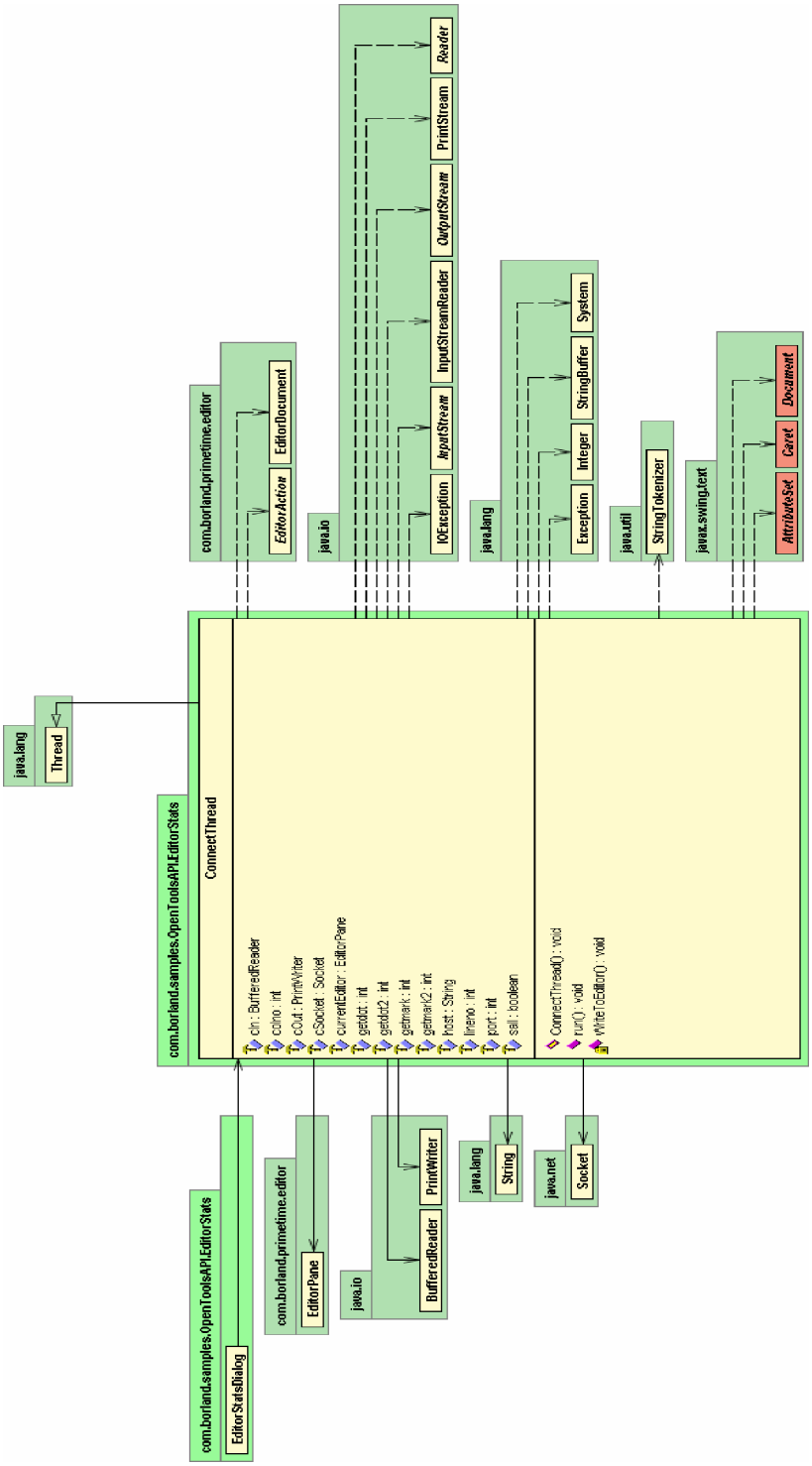

Figure 2 ConnectThread class diagram

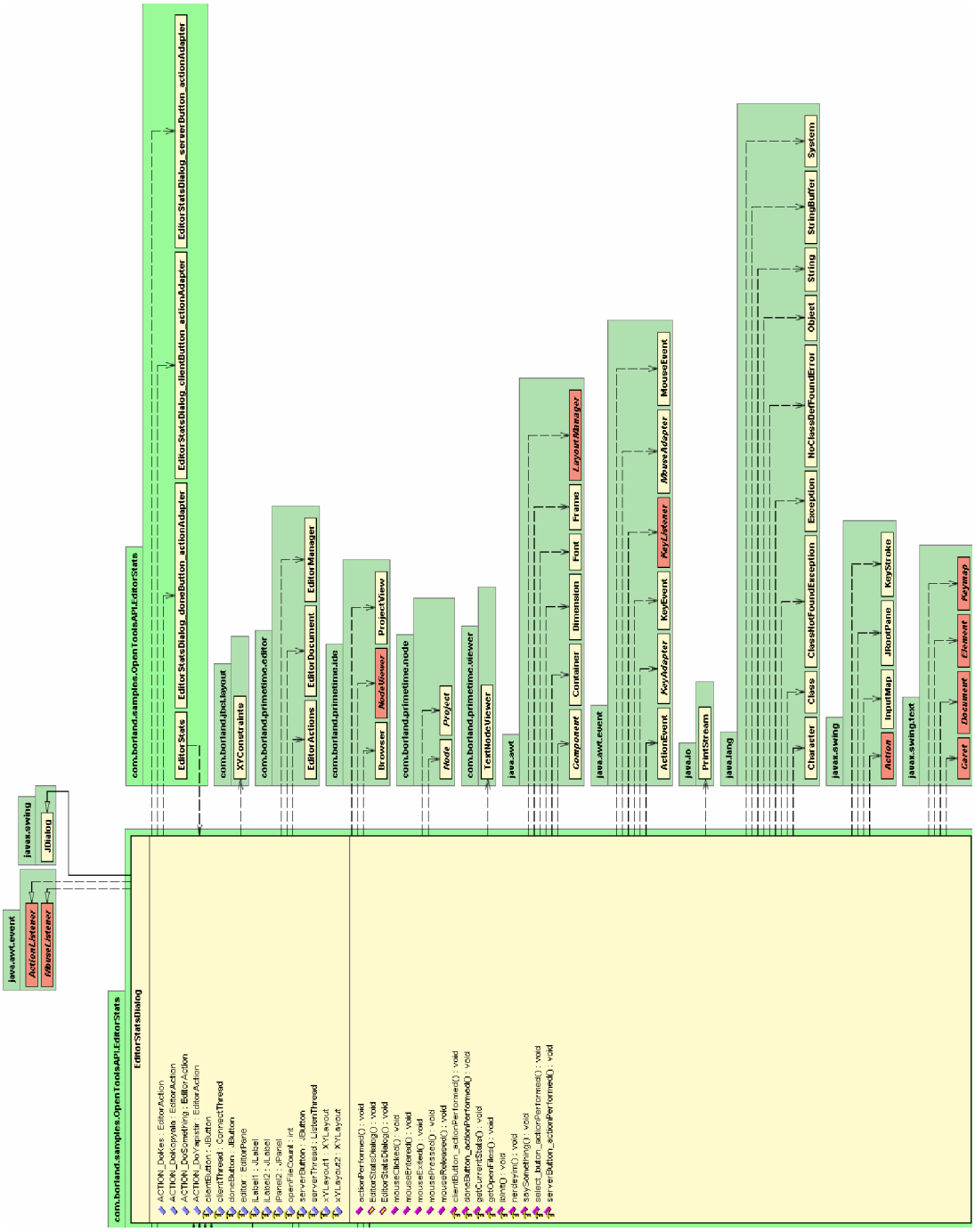

Figure 3 EditorStatsDialog Class Diagram - A

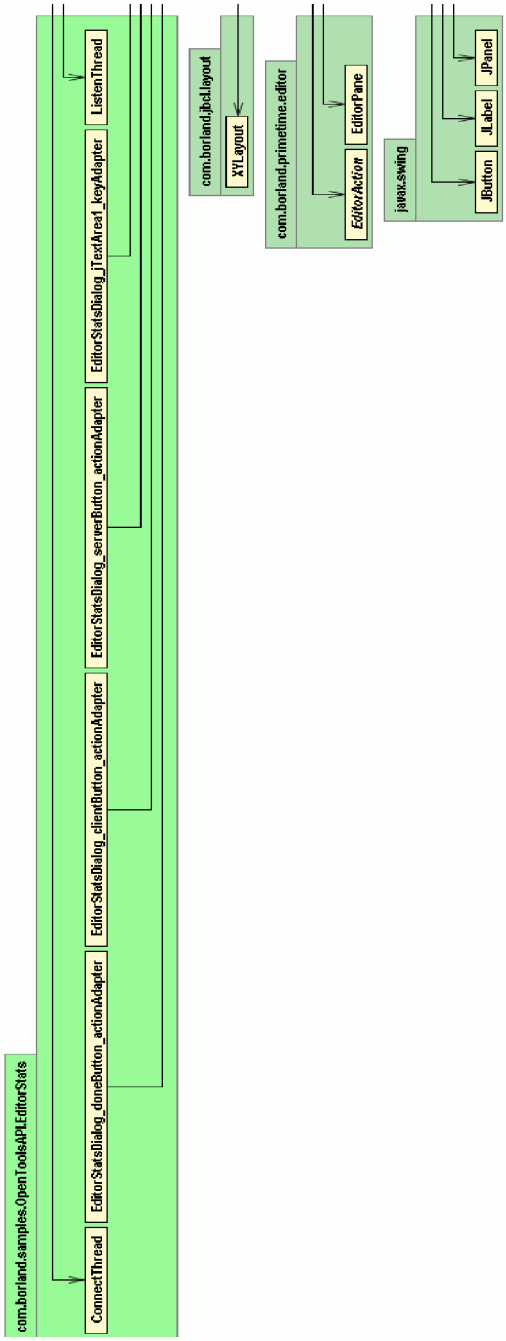

Figure 4 EditorStatsDialog Class Diagram - B

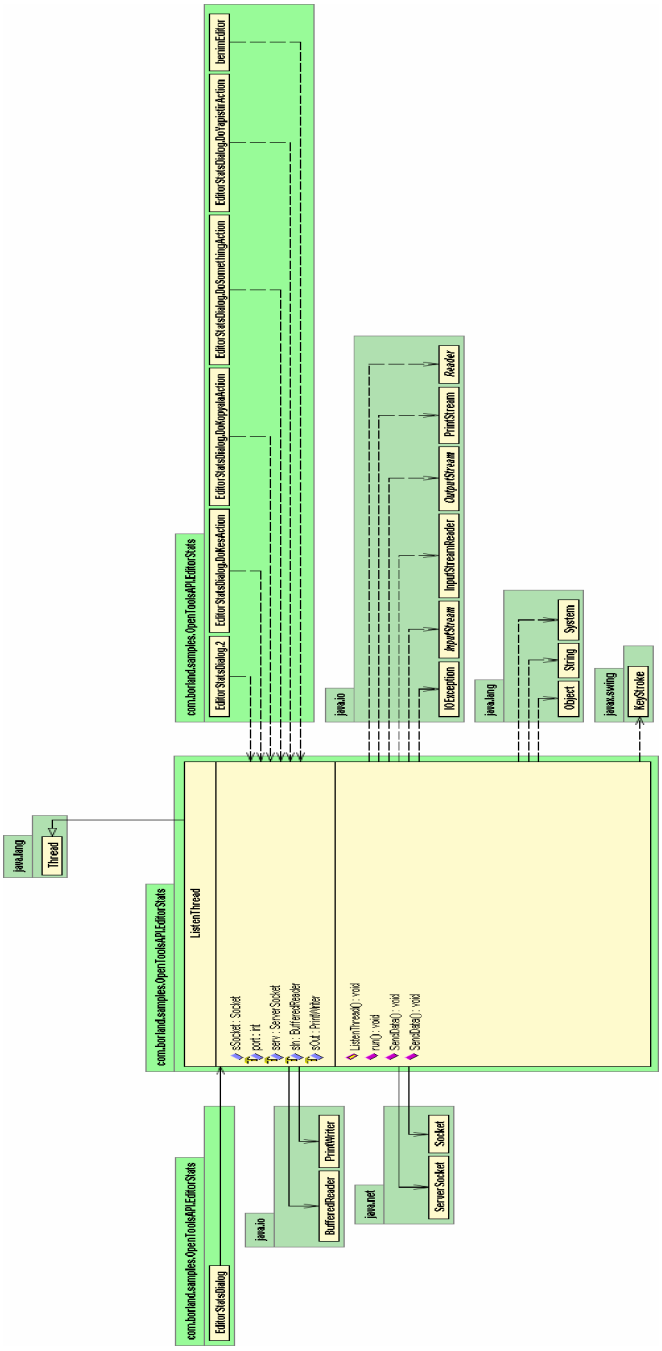

Figure 5 ListenThread Class Diagram

3.5.3 Detailed Description

a. Class: ConnectThread:

Identification	:	ConnectThread
Type	:	Class
Superclass	:	Thread class
Purpose	:	This class handles the connection to driver by the navigator and arranges the editor functions of the navigator.
Function	:	Generates a navigator demand to the listening driver, handles the editor functions in the navigator side
Subordinates	:	Properties and methods. Please refer to Section 3.5.4
Dependencies	:	OpenToolsAPI.EditorStatsDialog
Resources	:	java.io java.lang java.net java.util

b. Class EditorStatsDialog:

Identification	:	EditorStatsDialog
Type	:	Class
Superclass	:	JDialog Class
Purpose	:	This class supplies the user interface of JDP, specifies the role of the programmer, and handles editor

functionalities.

Function	:	It enables selecting the user the role of the driver or the navigator. Cut, Copy, Paste and Select-All Actions, mouse click events and driver editor facilities are handled.
Subordinates	:	Properties and methods. Please refer to Section 3.5.4
Dependencies	:	OpenToolsAPI.ConnectThread OpenToolsAPI.ListenThread
Resources	:	javax.swing java.awt com.borland.primetime.editor java.lang java.awt.event

c. Class ListenThread:

Identification	:	ListenThread
Type	:	Class
Superclass	:	Thread Class
Purpose	:	This class opens the driver socket for listening.
Function	:	Starts a thread where socket is continuously listened for the incoming messages. Handles the driver editor facilities.
Subordinates	:	Properties and methods. Please refer to Section 3.5.4
Dependencies	:	OpenToolsAPI.EditorStatsDialog
Resources	:	com.borland.primetime.editor

java.net

java.io

java.lang

3.5.4 Interface Description

Interface description involves classes as design entities, and provides access signatures related to subordinates (properties and methods) of classes in OpenToolsAPI package.

a. Class: ConnectThread:

Table 3 The interface description for ConnectThread Class

Subordinate	Description	Access allowed by	Access via signature
ConnectThread	method	All	public ConnectThread (EditorPane tempEditor)
run	method	All	Run():void
WriteToEditor	method	All	WriteToEditor():void

b. Class EditorStatsDialog:

Table 4 The interface description for EditorStatsDialog Class

Subordinate	Descrip-tion	Access allowed by	Access via signature
actionPerformed	Method	All	Public void actionPerformed (java.awt.event.ActionEvent e)
EditorStats Dialog	Method	All	public EditorStatsDialog (java.awt.Frame frame, java.lang.String title, boolean modal)
actionPerformed	Method	All	public void actionPerformed(java.awt.event.ActionEvent e)
jbInit	Method	All	void jbInit()
mouseClicked	Method	All	Public void actionPerformed (java.awt.event.MouseEvent e)
clientButton_ actionPerformed	Method	All	Public void clientButton_actionPerformed (ActionEvent e)
doneButton_ actionPerformed	Method	All	Public void doneButton_actionPerformed (ActionEvent e)
Nerdeyim	Method	All	Public void nerdeyim (String eventDescription, MouseEvent e) (java.awt.event.MouseEvent e)

Table 5 The interface description for EditorStatsDialog Class (continued)

saySomething	Method	All	(String eventDescription, MouseEvent e) (java.awt.event.MouseEvent e) (java.awt.event.MouseEvent e)
serverButton_ actionPerformed	Method	All	Public void serverButton_ actionPerformed (ActionEvent e)
DoKesAction_ actionPerformed	Method	All	Public static void DoKesAction_ actionPerformed (java.awt.event.ActionEvent e)
DoKopyalaAction _actionPerformed	Method	All	Public static void DoKopyalaAction_ actionPerformed (java.awt.event.ActionEvent e)
DoSomething Action_ actionPerformed	Method	All	Public static void DoSomethingAction_ actionPerformed (java.awt.event.ActionEvent e)
DoYapistirAction _actionPerformed	Method	All	Public static void DoYapistirAction_ actionPerformed (java.awt.event.ActionEvent e)

c. Class ListenThread:

Table 6 The interface description for ListenThread Class

Subordinate	Description	Access allowed by	Access via signature
ListenThread	Method	All	public ListenThread(EditorPane tempEditor)
run	Method	All	public void run()
SendData	Method	All	public void SendData(String sData)

3.6 Sequence Diagram

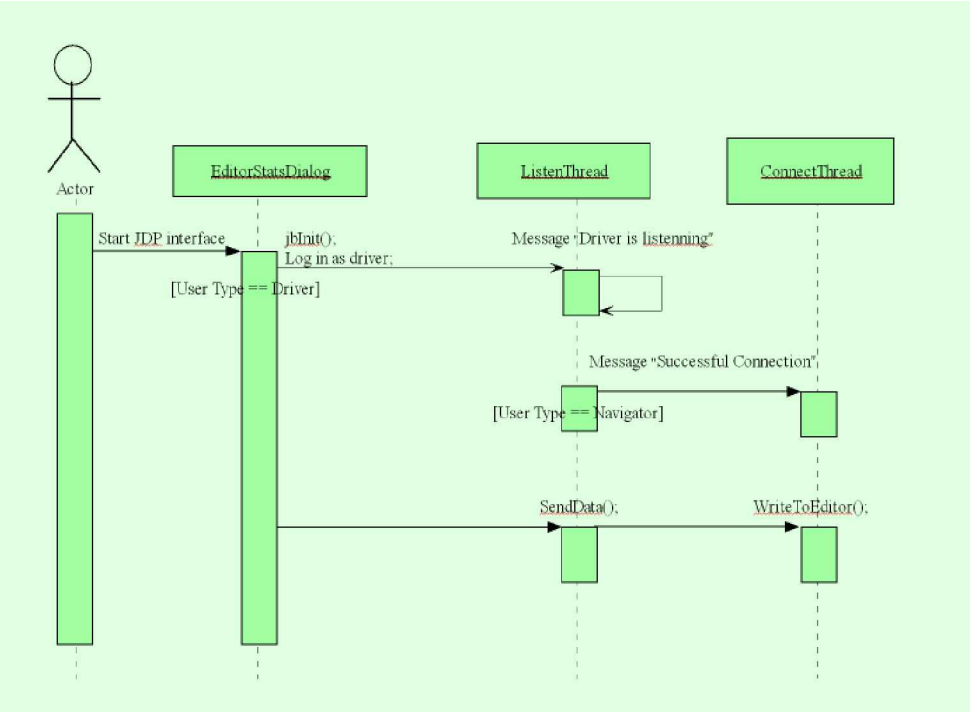

Figure 6 The sequence diagram of JDP

3.7 Restrictions

JDP is a tool to enable Distributed Pair Programming, but it does not claim to be a professional tool. Despite the facilities mentioned in Section 3.4, it has some challenges below:

1. The scope of this tool is limited to characters specified in Section 3.2 which does not include function keys (F1 to F12), page-down, page-up insert, pause, and print-screen.
2. The sending of the format (the font type, font size, bold, italic, etc.) of the code text to the other side is not implemented.

CHAPTER 4

Evaluation of JDP

In this chapter, the developed tool is evaluated. After a general evaluation, the positive and negative aspects of JDP tool are examined and a comparison between JDP and other evaluated DPP tools is given.

4.1 General Evaluation of JDP

JDP is a tool which customizes Borland JBuilder for the purpose of enabling distributed pair programming.

Distributed pair programming is a specific usage of pair programming which is a key practice of Extreme Programming. Design of JDP complies with the main methodology which advocates to generate code with two programmers one of which is the driver who is responsible for writing, editing and typing the code while the other programmer, navigator, is responsible for evaluating the larger design picture and guiding the driver. A natural result of this concept is to restrict the navigator from editing the code written by the driver. The only duty of the

navigator is to follow the code of the driver and navigate him/her by different means of communication tools, e.g. chat, teleconference, mail, etc. which are not in the scope of this study and should be supplied at the beginning of the process. The navigator can try some activities which will remain local.

Preserving the integrity of the shared code is an issue that needs to be addressed here: The driver can always synchronize the working file at both sides by simply selecting his whole file and pasting it over the editor area. This clearly gives supremacy to the driver, which is an intended feature of DPP, as discussed in Chapter 3.

File operations such as save, open, etc. are not addressed by JDP and remain within the set of functionalities offered by JBuilder. That is, both users should save, restore, and open their own files independently. This way, while code level driver-navigator hierarchy is implemented, file level independence of the two parties is preserved.

4.2 The Positive Aspects of JDP

1. JDP supports Borland JBuilder. It uses the facilities that JBuilder serves.
2. Communication bandwidth usage is limited to character transmission only. JDP uses the bandwidth at an amount of sending the characters typed by the driver.
3. The navigator and the driver can use the editor facilities locally.
4. Character entry at the driver is replicated synchronously at the navigator, serving DPP functionality needs specified in Section 3.3.
5. JDP permits users to preserve the integrity of the shared code if local changes are prevented.

4.3 The Negative Aspects of JDP:

1. Most of the JDP tools, as described in section 2.6, are supported by other types of communication facilities like chat, e-mail, audio / video communication (NetMeeting, Messenger) or Voice-over-IP applications. Because JDP focuses on coding phase, it does not handle extra communication issues. Adding such facilities to JDP would increase the effectiveness of the tool.

2. JDP has not been tested and used in any software development environment in real life by professional programmers. For this reason, possible problems while using it cannot be observed and the subjective issues like user-friendliness can not be assessed.

4.4 Comparison of JDP with Other Pair-Programming Tools

JDP is not a professional tool compared to the tools presented in Section 2.6. A brief comparison of JDP and other presented tools is presented below:

MILOS: In this tool, the system keeps track of the team members who are currently logged in. A developer is able to pair up with one of them using the application sharing and audio/video capabilities of MS NetMeeting [Maurer, 2002]. MILOS does not have a specific DPP facility, but uses MS NetMeeting facilities in Distributed Pair Programming while JDP is intended to support the operations after pair-up. In MILOS the navigator inspects the code and can comment on it using audio/video conferencing. The navigator may also take over and edit the method from his machine. JDP allows role exchange by logging off and re-logging in with a different role.

VNC: It is a screen sharing application while JDP is based on sending the characters from the driver to the navigator. “VNC allows a user’s desktop to be replicated onto multiple computers (in particular, two in the case of pair programming). Application output is displayed on both computers, while keyboard and mouse input from either computer is sent to the applications” [Hanks, 2004]. The output of the process of JDP is the code which is sent from the driver to the navigator. In VNC; both users have active keyboards and mice which make it difficult to handle keystrokes and cursors [Hanks, 2004] while such a problem does not occur in JDP.

ClearCase is a Software Configuration Management (SCM) system. Continuous integration which is one of the key practices of XP is provided with ClearCase. In pair programming, if both the navigator and the driver are authorized to access and edit the code, the versions should be managed and logged. Configuration management is kept outside the scope of JDP.

TUKAN is a synchronous distributed team programming environment. TUKAN works on shared replicated objects. It includes a version management system. JDP is character-based and does not include any version management system. TUKAN provides basic communication support while JDP should be supported by commercial tools to communicate. TUKAN enables planning and coordinating facility in the scope of “planning game” key practice of XP while JDP has nothing about the planning issues. TUKAN supports Distributed Pair Programming, not only with one person, but also by inviting other people to the environment. JDP is designed according to one pair only. In TUKAN, all users are allowed to edit the written code, i.e. there is not any difference between the navigator and driver from the point of the tool facilities, but in JDP only the driver has authority to edit the code.

MASE, as described in detail in Section 2.6.5, enables any user to access, browse, create, structure, and update any web pages in real-time using a web browser only. Each of these web pages acts like an electronic bulletin board discussion topic with a unique name. JDP uses JBuilder editor instead of web pages. MASE tracks the team member's availability to arrange pair for the people looking for a pair. JDP is designed for only one peer and selection is not available. MASE's integration with MS NetMeeting enables "sharing the code" and "collaborate on a design using the shared whiteboard", i.e. distributed pair programming [Chau, Maurer, 2004].

CUSeeMe and MS NetMeeting are commercial programs that enable synchronous communicating. CUSeeMe is audio/video conferencing software that allows Internet users to connect one-to-one, many-to-many, or any combination with the use of a reflector which allows others to join the conference. MS NetMeeting is a commercial program that enables videoconferencing, chat, file transfer, etc. NetMeeting enable distributed pair programming by program sharing which means allow one person to work in the shared program. JDP does not need any program sharing apart from JBuilder. CUSeeMe and MS NetMeeting can be used to support the communication in JDP.

CHAPTER 5

Conclusion

This chapter begins with a general discussion about the tool and ends with the improvement issues as a future work.

5.1 General Discussion about the Tool

JDP is a tool which is a plug-in to Borland JBuilder. It is based on sending the characters and some mouse events to the peer programmer who is geographically in any other part of the world. It is a synchronous tool in which the code of the programmer is sent to the other side without any trigger. JDP aims to have the same java file at both sides at the end of the distributed pair programming session.

The asymmetric character of JDP, that is; only sending information from driver to navigator, complies with the pair-programming concept as described in Chapter 3.

Another issue is real-life-usage. How effective JDP will be cannot be predicted because of lack of any testing team experiment. Like other DPP tools, JDP needs to be supported and enhanced by synchronous communication facilities like video conferencing, chat or Voice-over-IP applications. It has, however, been discussed in Chapter 4, that JDP responds to the needs stated in Chapter 3.

5.2 What Can Be Done to Improve JDP? - Future Work

This tool does not claim to be a professional one. In order to develop this tool, the following issues can be considered:

1. Handling the characters which are out of the scope of JDP, like function-keys, insert, page-down and page-up keys.
2. Capturing the menu facilities and enabling of seeing the other side menu events at the time of the call by mouse click.
3. Handling the file management operations like “Save” in order to work on the right document, provide integrity and specify the access permissions.
4. Sending of the format (the font type, font size, bold, italic, etc.) of the code text to the other side.
5. Enabling the helping facility of JBuilder editor, which lists the possible classes or alias and completes the word if pressed Enter key after the first characters are typed, at both sides.
6. Adding a module to send the necessary files to the other side.
7. Testing of JDP by a programming team and getting feedback to improve the tool, increasing the user friendliness level.
8. Adding more editor functionalities like undo, redo, etc.

JDP is only designed for distributed pair programming. It can be also enhanced by extensions towards a full scale XP tool. In order to transform JDP to an XP tool, the following facilities should be available:

1. Enable iteration and release planning,
2. Enable synchronous and asynchronous communication via commercial tools like MS NetMeeting or different means of communication.
3. Provide continuous integration by enabling code unification of more than two group members.
4. Do the acceptance tests according to the given orders.
5. Establish a database to store the necessary information about the project and integrated code.
6. Enable creation and changes of tasks.
7. Retrieve from or add to the current project user stories and available tasks.

In order to transform JDP to a tool that serves Distributed Software Engineering projects, the following facilities should be added to the above properties:

1 Enable configuration management,
2 Enable Authentication and access control ,
3 Support “Computer Aided Software engineering” (CASE) tools and source code analysis tools.
4 Support Validation and Verification tools.
5 Support Quality Measurement tools
6 Enable design and testing software.

“JBuilder Distributed Pair Programming Plug-in” (JDP) tool opened a gateway for distributed pair programming on a common commercial IDE like Borland

JBuilder. JDP enables many useful facilities that will make pair programming easier in distributed environments, however, it needs to be improved. The author hopes that the future studies on JDP will make it more adaptive, useful and easy-to-use tool.

REFERENCES

[Baheti, 2002], Baheti, P., Gehringer, E., Scotts, D. "Exploring the efficacy of distributed pair programming", NC State University, 2002

[Beck, 1999], Beck, K., "Embrace Change with Extreme Programming,", IEEE Computer, 1999

[Beck, et. al., 2001], Beck, K., Cockburn, A., Jeffries, R., and Highsmith, J., "Agile Manifesto", http://www.agilemanifesto.org, 2001

[Boehm, 2002], Boehm, B., "Get Ready for Agile Methods, With Care", Software Development, 2002

[Bowen, Maurer, 2002], Bowen, S., Maurer, F., "Using Peer-to-Peer Technology to Support Global Software Development – Some Initial Thoughts", University of Calgary, 2002

[Canfora, 2003], Canfora, G., Cimitile, A., Visaggio, C.A., "Lessons learned about distributed pair programming: What are the knowledge needs to address?", 2003

[Chau, Maurer, 2004], Chau, T., Maurer, F. "Knowledge Sharing in Agile Software Teams", University of Calgary, 2004

[Cockburn and Williams, 2000], Cockburn, A., Williams, L., "The Costs and Benefits of Pair Programming", 2000

[Cohen, 2003], Cohen, D., Lindvall, M., Costa, P., "Agile Software Development, A DACS State-of-Art Report", Fraunhofer Center for Experimental Software Engineering Maryland and the University of Maryland, 2003

[Davidson, 2002], Davidson, P., Hedrich, R., Leavy, T., Sharp, W., and Wilson, N., "Information Systems Development Techniques and Their Application to the Hydrologic Database Derivation Application", USBR Upper Colorado Regional Office, 2002

[Gehringer, 2003], Gehringer, E.F., "A Pair Programming Experiment in a Non-Programming Course", NC State University, 2003

[Glazer, 2001], Glazer, H., "Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity" Crosstalk, The Journal of Defense Software Engineering, 2001

[Hanks, 2004], Hanks, B.F., "Distributed Pair Programming; An empirical study", University of California, 2004

[Hayes, 2001], Hayes, S., "An Introduction to Extreme Programming", Khatovar Technology, 2001

[Highsmith, 2002], Highsmith, J., "Agile Software Development Ecosystems", Boston, MA, Addison-Wesley, 2002

[Highsmith, 2002b], Highsmith, J:, "What Is Agile Software Development?", Crosstalk, The Journal of Defense Software Engineering, 2002

[Highsmith, et. al., 2000], Highsmith, J., Orr, K., and Cockburn, A., "Extreme Programming", E-Business Application Delivery, 2000

[IEEE, Std 1016-1998], IEEE Std 1016-1998, Recommended Practice for Software Design Descriptions, 1998

[Kalermo, 2002], Kalermo, J., Rissanen, J., "Agile Software Development in Theory and Practice", 2002

[Kircher, et.al, 2001], Kircher, M.; Prashant, J. Corsaro, A.; Levine, D., "Distributed Extreme Programming", Siemens AG, 2001

[Maurer, 2002], Maurer F., "Supporting Distributed Extreme Programming", University of Calgary, 2002

[Maurer, Martel, 2002], Maurer F, Martel, S. "Process Support for Distributed Extreme Programming Teams", 2002

[Nosek, 1998], Nosek, J.T., "The Case for Collaborative Programming", Communications of the ACM, 1998

[Oksay, 2004], Oksay, U., "SM589 Project Report", Middle East Technical University, 2004

[Paulk, 2001], Paulk, M.C., "Extreme Programming from a CMM Perspective", Paper for XP Universe, 2001

[Paulk,2002] , Paulk, M.C., "Agile Methodologies and Process Discipline ", Software Engineering Institute, 2002

[Scacchi, 2001], Scacchi, W., "Process Models in Software Engineering", Institute for Software Research, University of California , 2001

[Schümmer& Schümmer, 2001], Schümmer, T., Schümmer, J., "Support for Distributed Teams in Extreme Programming", German National Research Center for Information Technology, 2001

[Smith, 2001], Smith, S., Stoecklin, S., "What We Can Learn From Extreme Programming", Converse College, 2001

[Sorensen, 1995], Sorensen, R., "A Comparison of Software Development Methodologies", Software Technology Support Center, 1995

[The C3 Team, 1998], The C3 Team, "Chrysler Goes to Extremes", Distributed Computing, 1998

[Wallin,2002], Wallin, C., Land, R., "Software Development Lifecycle Models, The Basic Types", ABB Corporate Research, 2002 gvu

[Williams, 2000], Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R., "Strengthening the case for pair programming", 2000

Web References

[http://www.cortland.edu/flteach/methods/obj1/cuseeme.html]. www survey, Retrieved May 8, 2005 from http://www.cortland.edu/flteach/methods/obj1/cuseeme.html

[http://www.departments.dsu.edu/disted/netmeeting] www survey, Retrieved May 8, 2005 from http://www.departments.dsu.edu/disted/netmeeting

[http://www.extremeprogramming.org] www survey, Retrieved May 4, 2005 from http://www.extremeprogramming.org

[http://www.microsoft.com/windows/netmeeting] www survey, Retrieved May 20, 2005 from http://www.microsoft.com/windows/netmeeting

[http://www.therationaledge.com/content/mar_01/f_xp_gp.html]. www survey, Retrieved May 8, 2005 from http://www.therationaledge.com/content/mar_01/f_xp_gp.html

[http://www.timefold.com/atria/10quest.html] www survey, Retrieved May 13, 2005 from http://www.timefold.com/atria/10quest.html

APPENDIX

Code CD is attached.