International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 8, August - 2014

Distributed Pair Programming: A Survey

N. Mohanraj[1] and A. Sankar[2]
Associate Professor at the Department of Computer Applications,
PSG College of Technology, Coimbatore, India.

Abstract - **In Software development practice today face the problems of low user satisfaction and low productivity. Current practice does not give the user required functionality but spend time in documentation like requirements specifications, architecture document, design document and test plans etc. The optimal solution to a better software development practice is agile software development methods like Pair programming, Extreme programming etc. But pair programming has issues like scalability as well as co-located pairs in the same physical location and this can be addressed by distributed pair programming. Several research papers are discussed in this survey paper that discusses topics like Agile Outsourcing (AO), Agile Dispersed Development (ADD) and Distributed Agile Development (DAD). Also Pair-programming environments are discussed to give the user idea about them. A variant of Extreme Programming is discussed as distributed pair programming or virtual teaming which can be defined as a group of people, who work together towards a common goal, but across time, distance, culture and organizational boundaries. Research works at various universities like the one at NC State University that is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects. It was discussed that pair programming reduces the risk of subtle errors that would make debugging excruciating; It give us a much broader code review and It provides an opportunity to communicate knowledge between coders. It has been further discussed about tools of pair programming based on the open source screen sharing application Virtual Network Computing (VNC). Also this survey suggests that distributed pair programming (DPP) can work better than solo programming. Four causes for dismissal phenomenon have been recognized: the faulty phone cause, the stranger cause, the two-minds cause, the anarchy cause. In this paper, we discussed the recent research in distributed pair programming and our intension is to attack the problem of pair dismissal where either both or one of the pair trying to omit sharing of knowledge and lead the team as a solo programmer. As a future work, we would provide a tool including usage of social networking platforms to avoid pair dismissal problem.**

1. INTRODUCTION

In traditional software development practice, we notice that there is low user satisfaction and low productivity. Also such traditional approaches produce large amounts of documentation like requirements specifications, architecture document, design document and test plans etc. instead of giving useful functionality to the end user. Due to such user unfriendly approaches sometimes projects are cancelled even before it is deployed. The solution to such thing is agile software development methods like Pair programming, Extreme programming etc. When a solo programmer uses the system, there are not much tools are required for synchronous of activities, but it is required in the case of pair-programming or extreme programming. The tools such as to replicate a user's desktop onto multiple computers in particular two in the case of pair programming. All input output methods should be shared between multiple computers and the application to be developed should also be deployed on both or multiple computers. We should notice that direct communication is better than documentation. That does not mean that document is unnecessary. In pair programming the limitations are scalability as well as co-located pairs in the same physical location. But due to the development of internet and social networking, we can foresee an approach which uses the advantages of such technologies. Hence we need to address distributed pair programming where there is a possibility of scalability as well as the users need not be co-located in the same physical location. Open source projects like Linux or the Apache Web Server where the development team members are around the world and they never met possibly as there is no requirement of such thing in software development process through extreme programming.

There is significant dependence on personal communication and customer collaboration. Agile Modeling disciplines can be difficult to apply on large teams (say 30 or more) without adequate tooling support, when team members are unable to share and collaborate on models (which would make Agile Software Development in general difficult) and when modeling skills are weak or lacking.

2. LITERATURE SURVEY

In recent times there is much attention in agile programming and was attracted by lot of researchers. An overview of research performed in agile environment is described as follows.

2.1 *Agile programming environments*

A problem arises to maintain close collaboration practices and run agile project in a distributed environment[9]. As a solution to this problem, a suitable tool support is usually employed; however, it seems insufficient at the moment. The paper [9] presents a set of general requirements that become a basis for further investigation into distributed collaboration needs and challenges. As a verification of initial assumptions, a new system was designed and part of it, that is responsible for supporting distributed pair programmers, implemented and experimentally evaluated. The first group includes conferencing applications (e.g. Microsoft NetMeeting), virtual whiteboards and desktop sharing solutions. The example of second group tool is TUKAN environment with a pair programming oriented tool consisting of voice-video

connection and other communication means. It proposes the following general requirements for discussed support, a computer system in turn:

- The system must support (preserve, stimulate, not suppress) the phenomenon of synergy which is not only the most valuable but also crucial factor, especially under the circumstances of a team and distribution of its pairs.
- The system ought to cover all functions that are recognized as necessary or useful in the geographically co-located mode, which stay in accordance with the primary requirement, including also functions which are decisive only for the friendliness of it.
- The system must fulfill all requirements for a modern computer system of its type as long as a conflict with the primary or secondary requirements (necessary ones) does not arise.

Agile Studio developed [9], which is meant for supporting selected agile practices. It has been observed that every collaboration is likely to take advantage of certain shared objects. Therefore, the editor is based on server-client architecture, where server side is responsible for sharing synchronized instances of the session objects through source files.

Three general cases for non co-located, Agile aligned development [10] is shown in Fig 1.

Agile Outsourcing (AO): Where an agile team is created at an appropriately low cost offshore location. Requirements are given by onshore team using shared documents but not shared ownership as commonly understood.

Agile Dispersed Development (ADD): It is practiced by much of the Open Source community and even by some commercial companies.

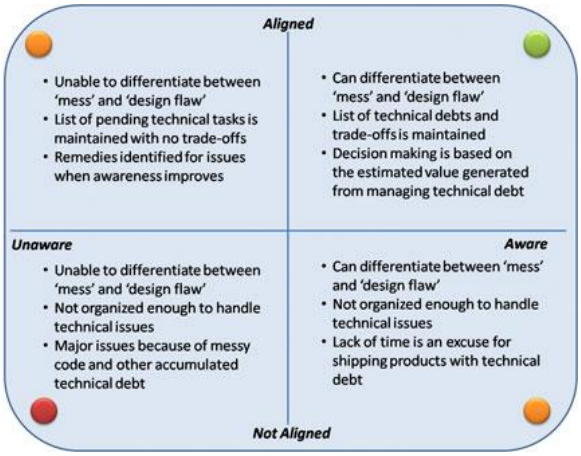

Figure 1: pair programming aligned vs. not aligned

Developers tend to be physically alone, but connected through a variety of communication channels. In the open source case, this results in practices such as Benign Dictator, and Trusted Lieutenant.

Distributed Agile Development (DAD): Customers are distributed. One development team is distributed evenly over several sites to remain close to the customers.

Team members in Distributed eXtreme Programming (DXP) as well as Distributed Pair Programming (DPP) are provided with as many communication media as possible [10]. At least these: individual and conference telephone, teleconference, video conference, email, IM, wiki, VNC. Widely separated team members need to maintain a common identity as technical problem solvers. They need to share rights and responsibilities toward each others' work, just as colocated workers do.

Members of a team in one location find it hard to understand the point of view of members in another location. Trust and cooperation break down, it is hard for one local group to work effectively with another. Team members find it hard to have faith in the good intentions of remote colleagues. Blamestorming replaces collaboration; finger pointing replaces problem solving [10].

The following Agile principles allow development teams to grow with businesses as they globalize.

- Distributed Standup
- Multiple Communication Modes
- Remote Pair
- One Team, One Codebase
- Functional Tests Capture Requirements
- One Team, One Build
- Code is Communication
- Tests Announce Intention

Convention speaks against having two people work together to develop code – having "two do the work of one", as some people see it. Managers view programmers as a scarce resource, and are reluctant to "waste" such by doubling the number of people needed to develop a piece of code and also experienced programmers are very reluctant to program with another person. Some say their code is "personal," or that another person would only slow them down. Others say working with a partner will cause trouble coordinating work times or code versions.

But it must be noticed as several well-respected programmers prefer working in pairs, making it their preferred programming style. Seasoned pair programmers describe working in pairs as "more than twice as fast". Qualitative evidence suggests the resulting design is better, resulting in simpler code, easier to extend. Even relative novices contribute to an expert' programming, according to interviews.

2.2 Pair-programming environments

Pair programming is one of the twelve practices of Extreme Programming (XP) [3]. In Pair programming it is assumed that the pairs will be working in front of the same workstation [4]. If Extreme Programming is to be used for distributed development of software, co-location becomes a limitation. A variant of Extreme Programming is used through distributed pair programming or virtual teaming. 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 [5]. This is depicted in Fig 2.

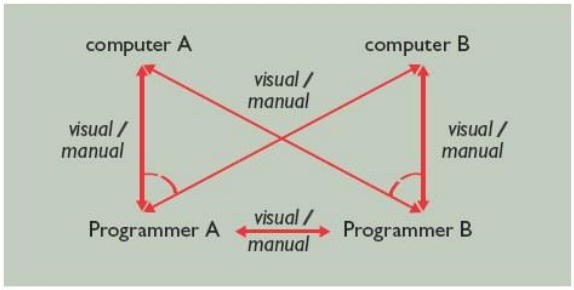

Figure 2: Typical pair programming environment

Pair programming [1] transforms what has traditionally been a solitary activity into a cooperative effort. One of the developers, called the driver, controls the computer keyboard and mouse. The driver is responsible for entering software design, source code, and test cases. The second developer, called the navigator, examines the driver's work, offering advice, suggesting corrections, and assisting with design decisions. The developers switch roles at regular intervals. Although role switching is an informal process, a typical interval is 20 minutes.

The experiment conducted at NC State University is a first indication that distributed pair programming is a feasible and efficient method for dealing with team projects [4]. It indicates the following:

- Distributed pair programming in virtual teams is a feasible way of developing object-oriented software.
- Software development involving distributed pair programming is comparable to that developed using co-located Pair programming or virtual teams without distributed pair programming.
- The two metrics used for this comparison were productivity (in terms of lines of code per hour) and quality (in terms of the grades obtained).
- Co-located teams did not achieve statistically significantly better results than the distributed teams.
- Feedback from the students indicates that distributed pair programming fosters teamwork and communication within a virtual team.

The requirements of a typical distributed pair programming tool [18] are as below.

R1. Synchronous editing of source code. As is the case for any modern source code editor it should highlight keywords based on the programming language being used and provide conventional editing tools such as: Cut, Copy, Paste, Find, and Replace.
R2. Only two programmers need to collaborate at the same time.
R3. The system should support the options of compiling and executing the source code being edited and should notify the users of the error messages reported by the compiler.
R4. The source code files to be shared should be stored in Web repositories to ensure that documents are available to all members of the development team. Furthermore, configuration control tools are increasingly being developed on top of Web servers to take advantage of the Web's ubiquity and open standards.
R5. Access to documents being edited should be controlled at the repository. Mechanisms to request and obtain shared resources need to be provided.
R6. Pair programming demands frequent communication between colleagues. The system should support text and audio-based communication, however, video is not considered necessary.
R7. Awareness of the presence and state of authors and documents, as well as access rights pertaining to shared resources should be provided to the users.

Data were analyzed in terms of productivity and quality. Also, student feedback formed an important third input for the experiment. Our goal was not to show that distributed pair programming is superior to co-located pair programming for student teams. Our goal was to demonstrate that distributed pairing is a viable and desirable alternative for use with student teams, particularly for distance education students. The results show that distributed teams had a slightly greater productivity as compared to co-located teams;

It is to be noted that pair programming should significantly reduce the risk of subtle errors that would make debugging excruciating. Also it would give us a much broader code review than we had ever had; and it would provide an opportunity to communicate knowledge between coders.

Also some investigative paths are briefly described:

- Economics: A recent controlled experiment [11] found only a small development cost for adding the second person. However, the resulting code also had fewer defects. The defect removal savings should more than offsets the development cost increase.
- Satisfaction: People working in pairs found the experience more enjoyable than working alone.
- Design quality: The study [11] also found that the pairs produced shorter programs than their solo peers, indicating superior designs.

The significant benefits of pair programming are that [12]·

- many mistakes get caught as they are being typed in rather than in QA test or in the field (continuous code reviews);
- the end defect content is statistically lower (continuous code reviews);
- the designs are better and code length shorter (ongoing brainstorming and pair relaying);
- the team solves problems faster (pair relaying);
- the people learn significantly more, about the system and about software development (lineof sight learning);

- the project ends up with multiple people understanding each piece of the system;
- the people learn to work together and talk more often together, giving better information flow and team dynamics;
- people enjoy their work more.

In [12] it is found that for a development-time cost of about 15%, pair programming improves design quality, reduces defects, reduces staffing risk, enhances technical skills, improves team communications and is considered more enjoyable at statistically significant levels.

2.3 Extreme programming environments

Using DXP(Distributed Extreme Programming) and open source processes as a baseline, the process of virtual software teams are, XP teams are usually much more closely coordinated than open source projects. Hence, project coordination support is strongly required for DXP. Here the tasks are assigned to XP team in a coordinated manner and deadlines are set as well as overview of the current state of the project is also updated. Team members access their to-do lists and to perform their tasks they retrieve relevant information in a coordinated way.

To establish synchronous communication, extensive e-mails are used as well as audio and video calls and text chat are also used. In the case of pair programming sharing of their application is also done. Both pull access as well as push access of information is done for the user.

The MILOS framework discussed in [2] nicely fits the requirements on DXP support. Nevertheless, we added the several extensions for supporting distributed XP like user stories in which a new product type that represents user stories was added.

In addition, whenever a new user story is entered, MILOS ASE automatically adds a task for implementing this story into the task list. Also release and iteration planning allows easily defining and changing releases, iterations, user stories, and tasks. In a distributed setting, the system provides awareness on what is going on in the project based on 4 task levels from XP (release, iteration, user story, and task). Further MS NetMeeting is integrated to be able to support distributed pair programming and synchronous communication.

2.4 Tools for Agile development process

The tool described in [1] is based on the open source screen sharing application Virtual Network Computing (VNC) [6]. Experiments were conducted [1] with control group as well as experiment group with students. Students in the control and experimental groups performed equally well on the final exam. Although it is not statistically significant, students who used the tool performed better on the exam than the students in the control group. Students in both experimental groups were also equally confident in their programming solutions. Comments from these students included:

- "We never had any need to. It was easier to meet in person."
- "Because we were able to find time together working on it at one person's place"
- "It was not necessary for us. It was very easy for us to meet in lab and talk face to face"

It is observed that the above remarks were given the students when VNC was used in the same lab and hence the students find meeting people is easier than using the tool. But when it is required in projects where the users don't meet because of distance, DPP (Distributed pair programming) is necessary.

The COLLECE (COLLaborative Edition, Compilation and Execution) system [7] is a groupware tool that enables users who are located in different workstations to collaborate in the same time (real time) in the building of a computer program. COLLECE was used in a study to compare the activity of Distributed Pair Programmers (DPPs) [8] and solo programmers. Here in the study particular attention was given to work productivity and program quality. It was observed that when the DPPs have enough experience in the use of the groupware tool and work collaboratively with their partner, the quality of programs is better than of those built by solo programmers. Also DPPs spent more time completing their tasks. They had to carry out additional interactions in order to coordinate and communicate in a distributed collaborative synchronous environment.

2.5 Distributed Systems and its environments:

Schumer and Schumer [13] and Maurer [14] have conducted research in this area that suggests that Distributed Pair Programming (DPP) can work. In a work by Baheti et al. [15] suggests that distributed pairing can be as effective as collocated pairing. Canfora et al. [16] studied virtual pairing by having students use a screen sharing application along with a text-based chat application. No audio channel was provided to the students.

Stotts et al. [17] provides further evidence of the potential success of distributed pairing. They describe an on-going series of experiments and case studies in which students virtually paired. Although distributed pairs successfully completed their programming assignments, they complained of their inability to point or gesture. As Stotts observed, "pairs need better capabilities for indicating areas of interest".

A representative sample of like responses [1] includes:

- "Well, besides it allowing us to work in the comforts of our own homes without ever getting out of our chairs, it also helped to overcome some schedule conflicts, and the time that would have been wasted just walking to the other person's computer was instead turned into productive programming time!"

- "You don't have to go all the way to a computer lab to pair program."
- "It made pair programming very easy and convenient. We didn't have to meet on campus or at each other's houses so we could always pair program without the effort of getting together. I think the class would have required a lot more time without the tool."

And responses about the use of the tool are as below:

- "If we do not meet in person, this combined with AIM almost perfectly emulated working side by side. We could work on it any time and take long breaks."
- "The pair programming tool allowed us to work together from two different places. The pointing function of the program also made it easier to point out errors and not accidentally type while my partner was typing."

Comments from this group(while collocated) of students includes:

- "I like the flexibility it offers in case two partners can't meet and work together. I liked being able to work on separate terminals while working side by side in the lab."
- "Easier than sharing computer"
- "Being able to switch driver/navigator easily"

Typical dislikes about the experiment are as below.

- "Communication with the partner is still awkward"
- "The tool was difficult to use when we were programming something we had never programmed before – for instance, when we first used arrays."
- "We sometimes wrote over each other's work and sometimes presenting things in person kept each others' interest."
- "AIM is not a good way to communicate, even with the headsets. Sometimes it is difficult to explain something through the air."

Also some other comments about the tool are:

- "We never had any need to. It was easier to meet in person."
- "Because we were able to find time together working on it at one person's place"
- "It was not necessary for us. It was very easy for us to meet in lab and talk face to face"
- "Too many programs to do a simple thing."
- "It was a hassle trying to get the program up and running than just simply meeting up with your partner at the lab. "

In the COPPER System [18], a synchronous source code editor that allows two distributed software engineers to write a program using pair programming. COPPER implements characteristics of groupware systems such as communication mechanisms, collaboration awareness, concurrency control, and a radar view of the documents, among others. It also incorporates a document presence module, which extends the functionality of instant messaging systems to allow users to register documents from a Web server and interact with them in a similar fashion as they do with a colleague. We report results from a preliminary evaluation of COPPER which provide evidence that the system could successfully support distributed pair programming. The Audio module establishes and maintains an audio communication channel between two clients so that their users can hold a conversation while collaborating.

Agile methodologies stress the need for close physical proximity of team members. However, circumstances may prevent a team from working in close physical proximity. For example, a company or a project may have development teams physically distributed over multiple locations. As a result, increasingly many companies are looking at adapting agile methodologies for use in a distributed environment [19].

The paper [20] describes the development and study of a technique tailored for distributed programming teams. The technique is based on an emerging software engineering methodology known as pair-programming combined with nearly 20 years of widespread and active research in collaborative software systems. Students use interactive information technology over the Internet, such as PCAnywhere and NetMeeting, to jointly and simultaneously control a programming session and to speak with each other synchronously. The earliest example of a collaborative computer system was NLS-Augment by Engelbart [21], an initial version of which was demonstrated in the early 1960.s. Engelbart.s system used shared CRTs, audio connections, mouse, and keyboard to allow crude teleconferencing and shared examination of text files by users who were not co-located. From these early beginnings, collaborative software systems became the subject of widespread research more than 15 years ago, with the creation of the PC. Ongoing research tends to focus in three main areas: hardware to provide effective communications; software concepts that allow sharing of artifacts; and conceptual models of how people want to or are able to interact effectively. The success of the simple DXP platform has led us to construct one that presents collaborators with a more significant video image, including the ability to create hyperlinks in a real-time video stream.

The following hypotheses were examined [22]:

- Distributed teams whose members pair synchronously with each other will produce higher quality code than distributed teams that do not pair synchronously. In the academic backdrop, quality can be assessed by the grades obtained by the students for their project. A statistical t-test can be performed to find whether one of the groups gets statistically significantly better results at different levels of significance ($p < 0.01$, 0.05, 0.1 etc.).

- Distributed teams whose members pair synchronously will be more productive (in terms of LOC/hr.) than distributed teams that do not pair synchronously.
- Distributed teams who pair synchronously will have comparable productivity and quality when compared with collocated teams.
- Distributed teams who pair synchronously will have better communication and teamwork within the team when compared with distributed teams that do not pair synchronously.

Five out of six students involved in distributed pair programming thought that technology was not much of a hindrance in collaborative programming. Also, about the same fraction (82%) of students involved in virtual teaming with or without pair programming felt that there was proper cooperation among team members. The experiment we conducted was a classroom experiment among 132 students, including 34 distance-learning students. To be able to draw statistically significant conclusions, such experiments have to be repeated, on a larger scale if possible. However, this experiment has given initial indications of the viability of distributed pair programming.

The statistical analysis showed a phenomenon we called the dismissal hypothesis: distributed pairs tend to stop collaboration and begin working as solo programmer [23]. Further it shows that in distributing pair programming people need a communication means that owns at least two features: vocal communication and a blackboard.

Four causes have been recognized: the faulty phone cause, the stranger cause, the two-minds cause, the anarchy cause.

- A defective communication is one of the four causes of the pair dismissal (the **faulty phone cause**).
- **The pair has to present very comparable levels of competence.** A way to obtain such a condition is to make the pairs work together in many projects. In this way it is possible also to prevent the **stranger cause**.
- The meeting should be realized with closing assessment aiming at verifying that the pair has formed an unique mind. The unique mind is intended as a uniform vision of the domain, strategies, goals, and the overall knowledge to be applied during the project. This should avoid the **two-minds cause**.

Second, knowledge needs to manage distributed pair programming have been pointed out, as listed above:

- Establish a behavioral protocol that defines clearly the roles in the pair and the switching of roles;
- Couple people with comparable experience and capabilities;
- Make people familiar in working with each others;
- Plan frequent brainstorming in order to create a common vision of the project.

Third, some requirements for an environment enabling distributed pair programming have been outlined:

- Enabling a communication as close as possible to the actual (co-located) human dialogue;
- Control the versioning of the changes made by pair programmers.

The following hypotheses were considered in [24]:

- Distributed teams whose member's pair synchronously with each other will produce higher quality code than distributed teams that do not pair synchronously.
- Distributed teams whose members pair synchronously will be more productive (in terms of LOC/hr.) than distributed teams that do not pair synchronously.
- Distributed teams who pair synchronously will have comparable productivity and quality when compared with collocated teams.
- Distributed teams who pair synchronously will have better communication and teamwork within the team when compared with distributed teams that do not pair synchronously.

3. CONCLUSION

Pair programming which is a part of Agile software development method has been one of the leading research areas. Mostly such research setup are academic setup where both the programmers in the pair co-located. This will not be case when we experiment in real time programmers in the industry. Hence the need for attempting distributed pair programming arises. In this paper, we discussed the recent research in distributed pair programming and our intension is to attack the problem of pair dismissal where either both or one of the pair trying to omit sharing of knowledge and lead the team as a solo programmer. As a future work, we would provide a tool including usage of social networking platforms to avoid pair dismissal problem.

REFERENCES

[1] Brian F. Hanks, Distributed Pair Programming: An Empirical Study, Extreme Programming and Agile Methods - XP/ Agile Universe 2004, Proceedings.
[2] D. Wells and L. Williams (Eds.): XP/Agile Universe 2002, LNCS 2418, pp. 13–22, 2002.
[3] K. Beck, "Extreme Programming Explained: Embrace Change". Reading, Massachusetts: Addison-Wesley, 2000.
[4] D. Wells and L. Williams (Eds.): XP/Agile Universe 2002, LNCS 2418, pp. 208–220, 2002.
[5] B. George., Y. M. Mansour, "A Multidisciplinary Virtual Team", Accepted at Systemics, Cybernetics and Informatics (SCI), 2002.
[6] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. IEEE Internet Computing, 2(1):33-38, January-February 1998.
[7] Bravo, C., Duque, R., Gallardo, J., García, J., García, P.: A Groupware System for Distributed Collaborative Programming: Usability Issues and Lessons Learned. In: International Workshop on Tools Support and Requirements Management for Globally Distributed Software Development, Centre for Telematics and Information Technology, pp. 50–56 (2007)
[8] Williams, L., Kessler, R.: Pair Programming Illuminated. Addison-Wesley, Reading (2002).
[9] G. Concas et al. (Eds.): XP 2007, LNCS 4536, pp. 70–73, 2007.
[10] Alistair Cockburn, Laurie Williams, The Costs and Benefits of Pair Programming,

[11] Williams, L., et al., Strengthening the Case for Pair-Programming, in IEEE Software. submitted to IEEE Software. Online at http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF
[12] Alistair Cockburn, The Costs and Benefits of Pair Programming, Internet source.
[13] Till Schummer and Jan Schummer. Support for distributed teams in extreme programming. In Giancarlo Succi and Michele Marchesi, editors, Extreme Pro gramming Examined, pages 355-378. Addison-Wesley, 2001.
[14] Frank Maurer. Supporting distributed extreme programming. In Extreme Programming and Agile Methods - XP/Agile Universe 2002, number 2418 in LNCS, pages 13-22. Springer, 2002.
[15] Prashant Baheti, Edward Gehringer, and David 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.
[16] Gerardo Canfora, Aniello Cimitile, and Corrado Aaron 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.
[17] David Stotts, Laurie Williams, Nachiappan Nagappan, Preshant Baheti, Dennis Jen, and Anne Jackson. Virtual teaming: Experiments and experiences with distributed pair programming. In Extreme Programming and Agile Methods - XP/Agile Universe 2003, number 2753 in LNCS, pages 129-141. Springer, 2003.
[18] Hiroshi Natsu, Distributed Pair Programming on the Web, Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03).
[19] D. Wells and L. Williams (Eds.): XP/Agile Universe 2002, LNCS 2418, p. 283, 2002.
[20] Prashant Baheti, Laurie Williams, Edward Gehringer, Distributed Pair Programming: Empirical Studies and Supporting Environments, Report of Department of Computer Science North Carolina State University Raleigh, NC 27695.
[21] D. C. Engelbart and W. K. English, .A Research Center for Augmenting Human Intellect,. presented at AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference, San Francisco, CA,, 1968.
[22] Prashant Baheti, Exploring the efficacy of distributed pair programming.
[23] Gerardo Canfora , Lessons learned about distributed pair programming: what are the knowledge needs to address, Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)
[24] Prashant Baheti, Exploring Pair Programming in Distributed Object-Oriented Team Projects.

N.Mohanraj received his Master degree in Computer Applications from PSG College of Technology, Coimbatore, India. He is currently working as Associate Professor at the Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore, India. Since 2008, he is working towards his PhD in Computer Science from the Faculty of Science and Humanities, Anna University. His research is focused on pair programming, agile methodologies, software engineering and OOAD.

A. Sankar received his PhD in Computer Science from Bharathiar University, Coimbatore, India, in 2003. He is currently working as an Associate Professor at the Department of Computer Applications, PSG College of Technology, Coimbatore, India. He has more than 24 years of teaching and ten years of research experience. His research interests include agile software engineering, data mining, e-learning and networks.