DISTRIBUTED PAIR PROGRAMMING: AN ANALYSIS OF THE MAIN TOOLS

ABSTRACT

This study aims to analyze the quality of software for Distributed Pair Programming. To this, the requirements for the Distributed Pair Programming were investigated in the literature. The software SAROS, PEP, SANGAN and COLLAB were analyzed. It was found that the analyzed software enable the DPP. However, they do not completely fulfill the developer needs. As a result, we propose a catalog of functional requirements to make the DPP feasible and meet the development requirements.

Keywords: Programming, Development, Distributed, Pair Programming, Software.

1 INTRODUCTION

Pair programming is a development practice that consists of two developers, working in a single computer on the same algorithm, code, project or test (McDowell et. al. 2006). This practice, very popular nowadays, gained space when implemented by the Extreme Programming (XP) agile methodology, being referred by Paulk (2001) as a basic practice of this methodology. In Pair Programming (PP), while one developer codifies, the other monitors and inspect the code, collaborating and discussing possible solutions and improvements to the proposed problem.

During the development, the roles can be inverted at any moment. In the beginning, it was very questioned as representing a loss of productivity, since there are two resources collaborating in the same activity. But Laurie Williams and Robert Kesller (2001) affirm that the pairs conclude in up to half of the time if compared to the traditional programming, and produce code with higher quality (with about 15% less errors and software defects), in addition to cost only 15% more, according to Laurie Williams and Alistair Cockburn (2001).

The distributed pair programming is a practice that makes possible to apply the pair programming in geographically distributed teams (Baheti et. al. 2002). The strengthening of the distributed software development, characterized by involving physically distant people in the project (Prikladnicki, 2013) and the appearance of the DXP (Distributed eXtreme Programming), that consists in applying XP in a geographically distributed team (Kircher et. al. 2001), made the DPP gain force.

Canfora et. al. (2006) affirm that the main difficulty to practice DPP is the lack of proper tools to support the distributed development. In this article, we analyzed four of the main tools available for this practice. To this, we searched the requirements to execute DPP in a productive and reliable way.

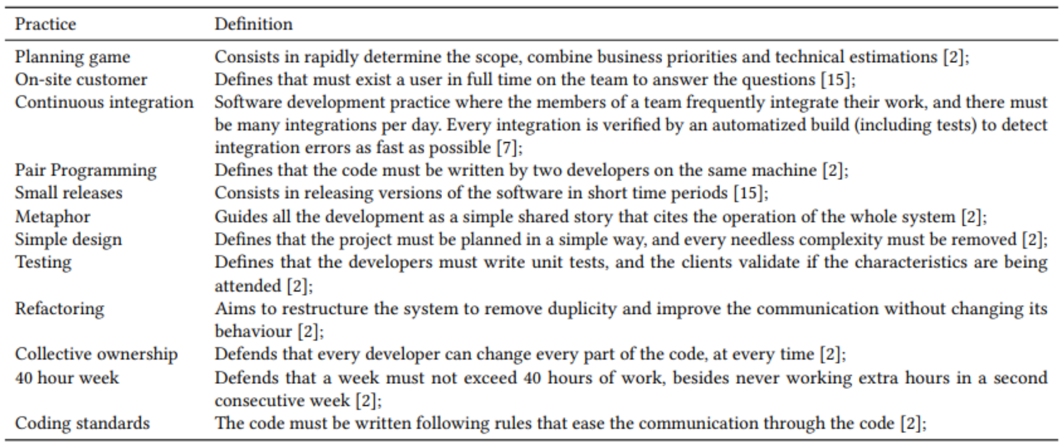
2 XP AND DXP

XP - Extreme Programming is one of the most popular agile methodologies for software development. It was created in 1996, by Kent Beck, who defined it as an efficient, low-risk, flexible and scientific methodology, as well as an amusing way to develop software (Beck, 2000). It appeared motivated by a series of software project risks and innovated by inventing a software development style which approaches these risks in a peculiar way, applying a set of principles and practices. As said by Paulk

(2001), minimalism, simplicity, an evolutive life cycle and the user involvement are principles on which XP is based, and also are principles of common sense that make part of every disciplined process, however, XP takes these principles to an extreme level. Although many practices can be applied in this methodology, Paulk (2001) classifies twelve of them as basic. These practices are shown in Table 1.

As shown in Table 1, strong and effective communication of the team members is extremely important to XP. Because of this, Beck (2000) emphasizes the need for proximity of its members. Given this limitation, Kircher et. al. (2001) defined an extension for XP that allows to execute this methodology even with team members in distinct places and named it DXP-Distributed Extreme Programming.

Table 1: Basic XP Practices



Practice	Definition
Planning game	Consists in rapidly determine the scope, combine business priorities and technical estimations [2];
On-site customer	Defines that must exist a user in full time on the team to answer the questions [15];
Continuous integration	Software development practice where the members of a team frequently integrate their work, and there must be many integrations per day. Every integration is verified by an automatized build (including tests) to detect integration errors as fast as possible [7];
Pair Programming	Defines that the code must be written by two developers on the same machine [2];
Small releases	Consists in releasing versions of the software in short time periods [15];
Metaphor	Guides all the development as a simple shared story that cites the operation of the whole system [2];
Simple design	Defines that the project must be planned in a simple way, and every needless complexity must be removed [2];
Testing	Defines that the developers must write unit tests, and the clients validate if the characteristics are being attended [2];
Refactoring	Aims to restructure the system to remove duplicity and improve the communication without changing its behaviour [2];
Collective ownership	Defends that every developer can change every part of the code, at every time [2];
40 hour week	Defends that a week must not exceed 40 hours of work, besides never working extra hours in a second consecutive week [2];
Coding standards	The code must be written following rules that ease the communication through the code [2];

They also show that, besides some XP practices do not suffer impacts from the geographic distance of the team members, it is an extremely important factor for the following practices: Planning game, On-site customer, Continuous Integration, Pair Programming, being necessary to adapt the way they are executed in XP, to execute them on DXP.

3 DISTRIBUTED PAIR PROGRAMMING – DPP

DPP is the application of pair programming with the pair members geographically distributed. Is an adaptation of the traditional pair programming to be executed on DXP (Hao, 2011). Several works indicate that pair programming is better than individual programming, but do these results apply to distributed pairs? To answer this question, it was performed an experiment in 2002 (Baheti, 2002) aiming to verify the viability and quality of software developed using pair programming with pairs geographically distributed in distinct places.

In the research, there were analyzed the quality of the produced code, the productivity, in terms of lines of code per hour, and the communication between teams, comparing distributed and collocated pairs. As result of the research, the authors concluded that:

1. Pair programming in distributed teams is a viable way to develop software;

2. Software development involving pair programming is comparable to local programming in terms of two metrics: productivity (in terms of lines of code per hour) and quality (in terms of scores obtained by correct test cases);

3. Local teams did not obtain a result statistically significant better result than distributed teams. Estácio (2013) says that on DPP the focus in infrastructure aspects must be reinforced and that as consequence of this a specific toolset must be adopted. He affirms also that the main condition for the DPP to work is the use of a specific tool.

4 METHOD

This is a research of the qualitative type, whose main characteristics are the immersion of the researcher in the context and the interpretative perspective in the conduction of the research [11]. In qualitative research, the research is an interpreter of reality (Bradley, 1993). The objectives of the research are the software to be analyzed.

This research has as aim to analyze the quality of some software that support the DPP practice. We observed that there are few programs used to this end. Some of them are SAROS, PEP, XPairtise, Rudel, SANGAM, and COLLAB. To perform the analysis, we strive to establish what software had the needed requirements for the DPP.

HAO (2011) highlights the following requirements, with high or medium priority, for a tool for this purpose:

(1) Project synchronization

(2) Editor Synchronization

(3) Support to switch the roles

(4) Visualize the actual role

(5) Chat

(6) Audio conference

(7) Video conference

(8) Real-time file edition

Besides these requirements, we list three other which we classify as having high relevance for DPP:

1. Debug (Shared with the pair): As highlighted by Baheti et. al. (2002) PP includes all the software development process stages, not only the codification but also planning, depuration, tests, etc. It became clear that whether in PP or DPP is extremely important that the programmers be able to debug the code. On PPD, it would be much more interesting that this debug could be shared in real time by the pair.

2. Inconsistency detection (Merge): During the use- of tools to support the DPP we perceive that when disconnecting, case code changes be performed when a new connection is made some of these modifications are lost. This is due to the simple substitution method used by the tools, like synchronism, instead of one which enables the merge. The authors of PEP (Aguiar et. al. 2018) cited the creation of this mechanism as an improvement for the plugin. And we consider this functionality fundamental for a DPP support software.

3. Synchronizing refactoring: As said by Paulk (2001), refactoring is one of the XP basic practices and its use is highly recommended, so we concluded that is necessary that the IDE refactoring action be shared during DPP.

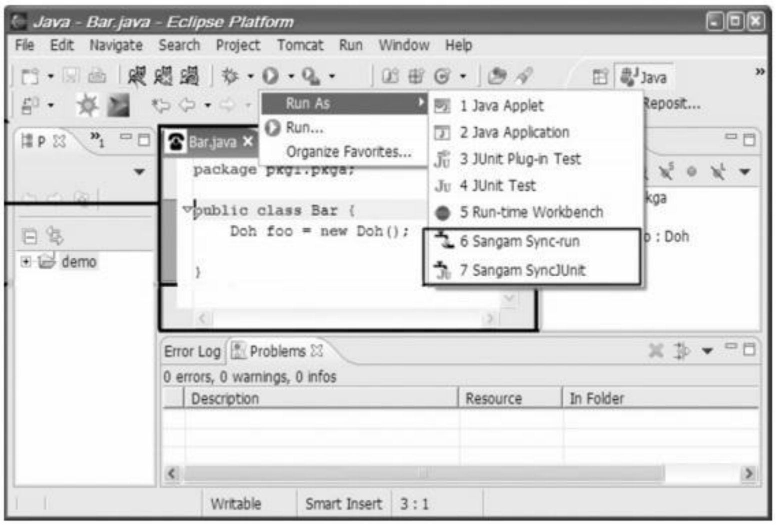
4.1 Analyzed Tools

The analyzed software products were: SAROS, PEP, SANGAM, and COLLAB. We analyzed such software, because they already appeared aiming to support the DPP, are more recent, and support more up-to-date versions of the IDE they are proposed.

4.1.1 PEP

O PEP (Figure 1) is a plugin created by students of the Federal University of Santa Catarina (UFSC), developed to work on the Eclipse IDE. According to Aguiar, Merize and Siqueira (2018) the plugin allows that while a developer write the code, other analyses it remotely, being possible also the exchange of roles at every moment, that is, who was analysing start coding, reducing considerably the number of errors and generating a gain in the development quality. It has an identical architecture on both sides and does not allow simultaneous changing in the code, and while a plugin (Server) sends the code change actions the other (Client) receives and applies these changes so that it appears locally performed.

Figure 1: PEP Plugin



```
package pkg1.pkga;

public class Bar {
    Doh foo = new Doh();

}
```

0 errors, 0 warnings, 0 infos

Source: the author.

The plugin is composed mainly of the following functionalities:

(1) Code synchronizing: It allows to synchronize all the content edited by one of the peers to the other, automatically or manually.

(2) Project sharing resource The tool is not well documented, what brings several difficulties, mainly in the plugin configuration. Furthermore, the system does not have a robust error treatment, being very fault-prone.

4.1.2 SAROS

Saros is an Eclipse plugin (Figure 2) for collaborative software development, that offers a solution for some of the problems faced on DPP. According to Salinger et. al. (2010) the plugin was initially designed to solve questions related to DPP but evolved to cover a wider scope, the distributed collaborative software development, being DPP only one of the practices the plugin allows to execute.

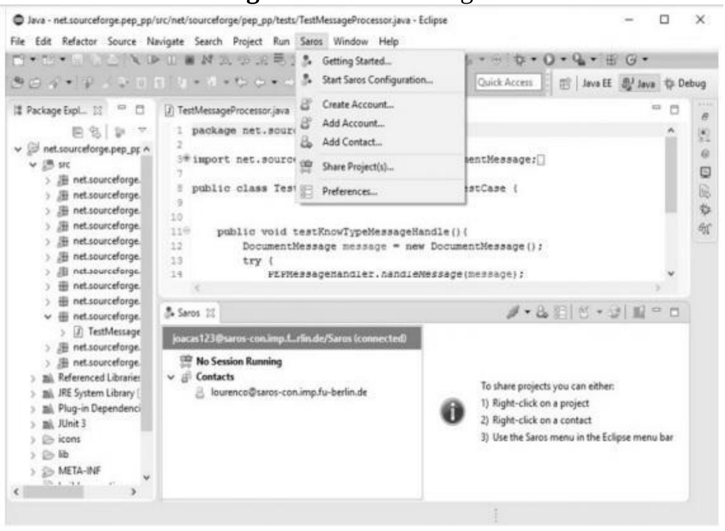
The tool allows two users geographically distributed in distinct places to shared an IDE, in a way that all the developers have an identical copy of the project and can program together. When editing documents, the developers can see immediately the modifications performed by the other participants, as well as to know the responsibility for every change. Among its functionalities related to DPP, there are:

• Real-time file edition: Allows that two users edit the same file, at the same time;

• Code synchronization: Permits to synchronize all the content edited by one of the peers, to the other;

• Communication: It has a chat tool to ease the communication between the peers;

Figure 2: SAROS Plugin.


Java - net.sourceforge.pep_pp/src/net/sourceforge/pep_pp/tests/TestMessageProcessor.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Saros Window Help
Getting Started...
Start Saros Configuration...
Create Account...
Add Account...
Add Contact...
Share Project(s)...
Preferences...
Quick Access
Java EE
Java
Debug
Package Expl...
TestMessageProcessor.java
Saros

```
package net.sourc
                 
import net.sourc                    entMessage;
                 
public class Test                   stCase {
                 
                 
    public void testKnowTypeMessageHandle(){
        DocumentMessage message = new DocumentMessage();
        try {
            PEPMessageHandler.handleMessage(message);
```

joacas123@saros-con.imp.f...rlin.de/Saros (connected)

No Session Running

Contacts

lourenco@saros-con.imp.fu-berlin.de

To share projects you can either:
1) Right-click on a project
2) Right-click on a contact
3) Use the Saros menu in the Eclipse menu bar

Source: the author.

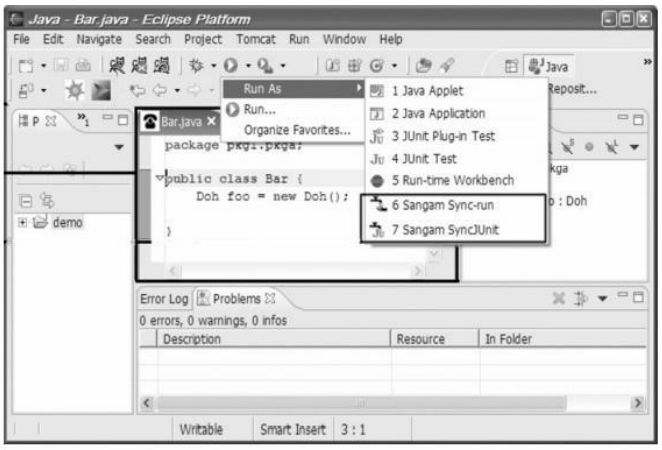
• Inconsistency detection: It allows the merging of the files, that is, case the users change the same part of the code they are notified and choose what version must be maintained.

Saros is a tool very simple to use. The installation process does not require advanced configurations, and it is well documented, having an installation manual and a tutorial.

4.1.3 SANGAM

Developed by students from the North Carolina State University in the United States, Sangam (Figure 3) was designed to fulfill the need for specific tools for DPP (Ho et. al. 2004), being conceived since the beginning to meet a series of requirements stated as minimal for this practice. It allows the users to share a development environment as they were using the same computer, allowing this way the practice of DPP.

Figure 3: Sangam Plugin



```
package pkg1.pkga;

public class Bar {
    Doh foo = new Doh();
}
```

Source: the author.

The tool was developed to synchronize the development environments of two geographically distributed programmers and aims to show that, in terms of productivity and quality, the pair programming can be such effective as the local, if executed in a correct manner. Among its functionalities, are:

• Synchronism: Allows synchronous edition (typing, selecting, opening and closing files);

• Resource synchronization Actions such as adding, erasing or modifying files are automatically reflected;

Discontinued in 2002, Sangam is not compatible anymore with the most recent version of Eclipse IDE.

4.1.4 COLLAB

Collab is a plugin (Figure 4) developed for Netbeans, the IDE developed by Sun, was developed aiming to make possible that distributed teams interact in the same way that local teams. It allows to revise, edit and create code in real time together with another developer. It was released in 2004 and referenced by specialized magazines as being the only software with this objective. Pioneer and discontinued, Collab has the following features:

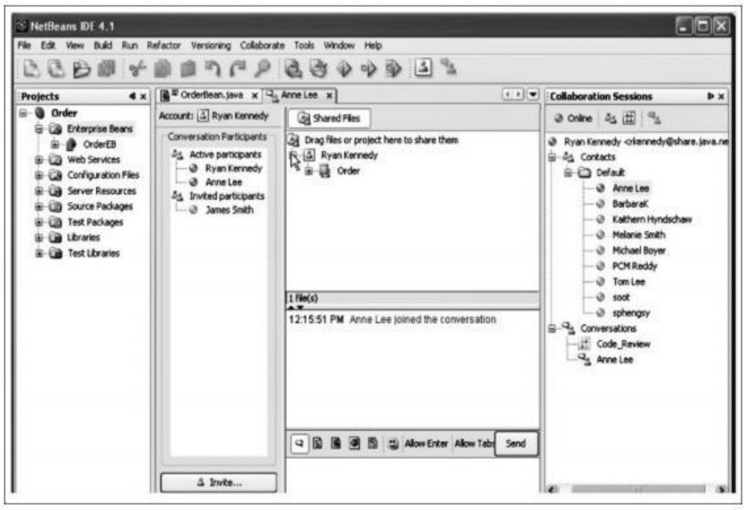
• Code synchronization;

• Operations synchronization: some IDE functionalities (e.g.: commands for class creation, deletion, etc.) can be synchronized;

• Concurrence control: Blocking of code parts, allowing simultaneous edition only for different parts of the same class.

• Project sharing: allows sharing projects or specific classes from a project. The software is discontinued and it is not available for the most recent versions of the Netbeans IDE.

Figure 4: COLLAB Plugin



12:15:51 PM Anne Lee joined the conversation

Source: the author.

5 RESULTS

We defined a catalog of functional requirements based mainly on two factors: the already existent tools (and the problems associated to them), and the minimal features needed for DPP mentioned by Hao (2011), (see Section V). Below, we list these requirements defined as essentials to practice the DPP with efficiency and high productivity:

(1) Synchronize code: this functionality allows to synchronize all the content edited by one of the peers to the other, automatically or manually;

(2) Synchronize projects: this functionality allows the user to select what projects he wants to share with the peer;

(3) Detection of inconsistencies (Merge): this functionality allows that after the synchronization of the code the users be notified about possible inconsistencies such as, for example, the case of both users making changes in the same part of the code when they were not connected;

(4) Real-time file edition: this functionality allows that a member of the peer keep a real-time tracking of the changes performed by the other;

(5) Debugging (shared with the peer): this functionality allows the pair to debug the same part of the code at the same time;

(6) Synchronize refactoring: this functionality allows that the refactoring option of the IDE be synchronized to the members of the pair, that is, when performing the refactor of a part of the code this must be reflected for the peer;

(7) Role changing: this functionality allows that at every moment the pair members change their roles, changing from controller (developer) to controlled (inspecting);

(8) Visualizing the current role: this functionality allows that at every moment the pair member see the role he is acting, either as controller (developing) or controlled (inspecting);

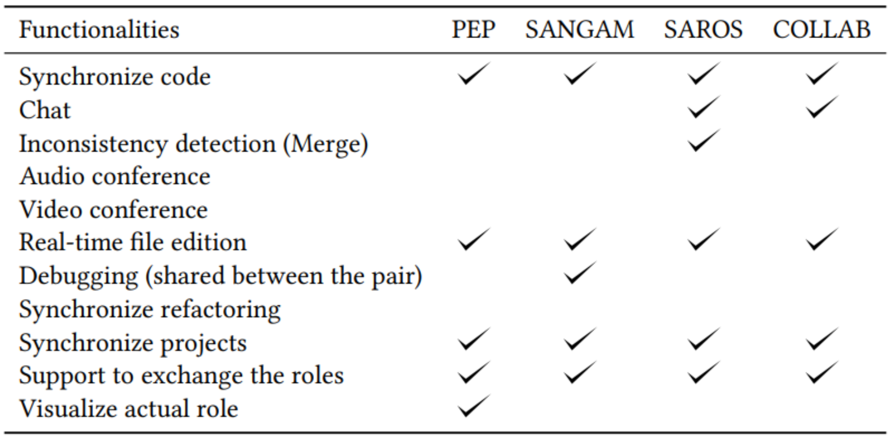
(9) Chat: This functionality allows the pair to interact through texts sent from the tool itself;

(10) Audio conference: this functionality allows the pair to interact in real time through audio sent from the tool itself;

(11) Video conference: this functionality allows the pair to interact at real time through videos sent from the own tool;

Given these essential requirements, we can observe in Table 2 that although the authors of the analyzed tools to defend their solutions as something new and that improve the agility in the use of DPP, none of these programs attend all the relevant requirements to the DPP.

Table 2: Comparison of DPP tools



Functionalities	PEP	SANGAM	SAROS	COLLAB
Synchronize code	✓	✓	✓	✓
Chat			✓	✓
Inconsistency detection (Merge)			✓	
Audio conference				
Video conference				
Real-time file edition	✓	✓	✓	✓
Debugging (shared between the pair)		✓		
Synchronize refactoring				
Synchronize projects	✓	✓	✓	✓
Support to exchange the roles	✓	✓	✓	✓
Visualize actual role	✓			

6 CONCLUSIONS

The efficiency of the DPP practice is directly related to the tools used to support this practice. The main condition of the DPP to work is the use of a specific tool (Estácio, 2013). Being the absence of a tool specific for DPP a waste of time (Ho et. al. 2004). The lack of software to attend DPP became evident during the elicitation performed in this research.

The set of requirements for a tool to support the DPP, cited in this research, is an attempt to contribute in the construction of a tool specific for the DPP which minimally attends the pair, with resources that ease the communication and allow the pair to work just as if they were in the same physical environment.

Scientifically, the legitimization of the set of requirements is due to the research process as a whole, represented in the methodology section. None of the analyzed software products attend all the needed functionalities for the DPP. Only SAROS and COLLAB, for example, have a resource to provide the pair communication. Functionalities like audio and video conference, which aims to improve even more the pair communication, are not available in any of the tools.

Among the analyzed tools, only SAROS can be used in the most recent version of the IDE to which it was proposed, and it is the only one that is not discontinued. For this reason, we indicate it to practice the DPP. However, even SAROS do not have advanced resources, such as shared debugging and refactoring synchronization. Another tool which can be used is PEP, although being very simple and do not have many of the listed resources, it showed itself very practical, attending the basic functionalities. The resources to provide and ease the pair communication are completely absent of PEP. This and the fact that it is not supported in the most recent version of the IDE to where it was proposed were the main causes to PEP be not classified by us as the best tool.

Although we can judge that the analyzed software contribute significantly for the DPP, we see as needed for a future work the creation of a new plugin to attend all the requirements to practice the PPD (see Section VI). We also highlight the importance of putting this new plugin to use in real projects, aiming to validate its quality.

REFERENCES

Alistair Cockburn and Laurie Williams. 2001. Extreme Programming Examined. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Chapter The Costs and Benefits of Pair Programming, 223–243. http://dl.acm.org/citation. cfm?id=377517.377531

Bernardo Estácio. 2013. Desenvolvimento de um conjunto de boas práticas para a programação em par distribuída. Master’s thesis. Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Programa de Pós-Graduação.

Bonnie Kaplan and Dennis Duchon. 1988. Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study. MIS Quarterly 12, 4 (1988), 571–586. http://www.jstor.org/stable/249133

Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. 2006. Pair Programming Improves Student Retention, Confidence, and Program Quality. Commun. ACM 49, 8 (Aug. 2006), 90–95. https://doi.org/10.1145/1145287.1145293

Chih-Wei Ho, Somik Raha, Edward Gehringer, and Laurie Williams. 2004. Sangam: A Distributed Pair Programming Plug-in for Eclipse. In Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (eclipse ’04). ACM, New York, NY, USA, 73–77. https://doi.org/10.1145/1066129.1066144

Gerardo Canfora, Aniello Cimitile, Giuseppe Di Lucca, and Corrado Aaron Visaggio. 2006. How Distribution Affects the Success of Pair Programming. 16 (04 2006), 293–313.

Jana Bradley. 1993. Methodological Issues and Practices in Qualitative Research. The Library Quarterly: Information, Community, Policy 63, 4 (1993), 431–449. http://www.jstor.org/stable/4308865

J. Hao. 2011. Distributed Pair Programming in Global Software Development. Master's thesis. School of Informatics, University of Edinburgh.

Kent Beck. 2000. Extreme Programming Explained: Embrace Change. AddisonWesley Longman Publishing Co., Inc., Boston, MA, USA.

KUSTERER R KALALI, M. [n. d.]. Developer Collaboration Module. ([n. d.]). https://netbeans.org/kb/articles/quickstart-collaboration.html

Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. Computer Science Education 11, 1 (2001), 7–20. https://doi.org/10.1076/csed.11.1.7.3846 arXiv:https://www.tandfonline.com/doi/pdf/10.1076/csed.11.1.7.3846

Leandro P De Aguiar, Giorgio S C Merize, and Frank Siqueira. 2018. PEP – Plugin Eclipse para Programação em Par Distribuída. (06 2018).

M Fowler. [n. d.]. Continuous integration. ([n. d.]). http://martinfowler.com/ articles/continuousintegration.html

M. C. Paulk. 2001. Extreme programming from a CMM perspective. IEEE Software 18, 6 (Nov 2001), 19–26. https://doi.org/10.1109/52.965798

Michael Kircher, Prashant Jain, Angelo Corsaro, and David Levine. 2001. Distributed eXtreme Programming. In Second international conference on eXtreme Programming and Agile Processes in Software Engineering. 66–71.

Prashant Baheti, Dr Laurie Williams, and Dr David Stotts. [n. d.]. Exploring pair programming in distributed object-oriented team projects. In In Proceedings of XP/Agile Universe 2002. Springer Verlag, 2002.

Rafael Prikladnicki. 2013. Desenvolvimento Distribuído de Software e Processos de Desenvolvimento de Software. Ph.D. Dissertação - Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Programa de Pós Graduação.

Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk. 2010. Saros: An Eclipse Plug-in for Distributed Party Programming. In Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10). ACM, New York, NY, USA, 48–55. https://doi.org/10.1145/1833310.1833319