ENHANCING ADAPTIVITY AND INTELLIGENT TUTORING IN DISTRIBUTED PAIR PROGRAMMING SYSTEMS TO SUPPORT NOVICE PROGRAMMERS

Despina Tsompanoudi and Maya Satratzemi
Department of Applied Informatics, University of Macedonia, 54006, Thessaloniki, Greece

Keywords: Pair programming, Distributed pair programming, Adaptive collaboration support, Intelligent tutoring.

Abstract: Pair programming is a computer programming technique where two programmers share one computer for software development. When applied in an educational context, pair programming has been reported as an effective teaching method, mainly improving student satisfaction, retention rates and program quality. Several systems have been developed to cover the need for pair programming over distance. We present the features of such systems including their advantages and weaknesses, and we examine if these systems eliminate known issues of pair programming, such as unequal participation. Finally, considering the limitations of current systems of distributed pair programming and taking into account common difficulties encountered by novice programmers, we propose adaptation strategies for supporting student collaboration in a distributed pair programming environment.

1 INTRODUCTION

Computer programming is still very challenging for many students, especially for novices. Novice programmers lack the skills for problem solving and show poor performance (Lister et al., 2004). As a result, introductory computer science courses have low pass rates and relatively high dropout rates. An alternative approach to teaching programming, which can be particularly valuable to novice programmers, is Pair Programming (PP). In pair programming two programmers, sitting side by side, share one computer to develop software. One of them, called the "driver", has control of the mouse and keyboard and is writing the code, while the other partner, called "navigator" or "observer" checks for syntax or logic errors and suggests alternative approaches. The roles of the driver and the navigator should be switched regularly.

Pair programming has been applied in classroom settings and evaluated for its effectiveness in several studies, as an alternative approach to teach computer programming. Compared to solo programming, research reports that PP improves program quality and reduces defects (Cockburn and Williams, 2001; Duque and Bravo, 2008; McDowell et al., 2003a; Sanjay and Vanshi, 2010; Zacharis, 2009). Students working in groups are more confident in their assignment solutions, experience less frustration and enjoy their work more (Boyer et al., 2008; Cockburn and Williams, 2001; Zacharis, 2009). Moreover, students show improved performance in exams or programming assignments (Benaya and Zur, 2006; Jun et al., 2007; McDowell et al., 2003b; Williams et al., 2002), and are more likely to continue in a computing-related major after completing the course (McDowell et al., 2003b). Since PP is a form of collaborative learning (Preston, 2005), students practicing this method get involved in discussing and negotiating problems, they learn from each other and share problem solving skills (Sanjay and Vanshi, 2010; Williams et al., 2002). Retention and pass rates were also examined with positive results (McDowell et al., 2003b; Williams et al., 2002). Finally, teachers report reduced workload during programming assignments because students are able to solve more problems on their own (Williams et al., 2002; Hanks, 2007).

Nevertheless, there are some drawbacks in PP too. Students' performance depends largely on the team composition. Various group formation strategies were tested (e.g. random assignment, based on students' preferences or students' skill levels) but none of them has shown to be more effective. When students collaborate there's always

Tsompanoudi D. and Satratzemi M..
ENHANCING ADAPTIVITY AND INTELLIGENT TUTORING IN DISTRIBUTED PAIR PROGRAMMING SYSTEMS TO SUPPORT NOVICE PROGRAMMERS.
DOI: 10.5220/0003469603390344
In *Proceedings of the 3rd International Conference on Computer Supported Education* (CeLS-2011), pages 339-344
ISBN: 978-989-8425-50-8

an issue if they contribute equally in their assigned work. Particularly in PP, the roles of the "driver" and the "navigator" should be switched regularly. Aforementioned issues depend on students' skills and their personality. Some studies also report that pairing students need more time than solo students to complete the programming assignments due to coordination effort (Cockburn and Williams, 2001; Duque and Bravo, 2008). Another drawback of PP is the collocation requirement, because students have to plan their meetings and this can lead to scheduling conflicts.

To address the problem of the collocation requirement various systems were proposed to support distributed pair programming. Distributed pair programming (DPP) is a variation of PP where the team members are at different locations and they collaborate through a real-time editor or by sharing their desktop (Sanjay and Vanshi, 2010). In this way students are more flexible and they are not required to meet physically. Compared to collocated PP, studies show that DPP does not appear to have any negative impact on students' performance (Hanks, 2008; Jun et al., 2007). Therefore DPP inherits all benefits of PP eliminating the collocation constraint, and is suitable for distance learning.

In this paper we present our study results of DPP systems which were evaluated from an educational perspective (Section 2). Then we propose some features to be embedded in systems for DPP in order to eliminate known issues of PP and effectively support students' collaboration (Section 3). We conclude in Section 4 with our study results and our suggestions.

2 SYSTEMS FOR DISTRIBUTED PAIR PROGRAMMING

In order to implement DPP some basic requirements must be met. Winkler et al. (2010) investigated the basic requirements of DPP tools, indicating among them a shared workspace, floor control, communication channels, gesturing support and awareness features. In more detail, a DPP application should provide a shared editor where students can collaborate adopting the roles of the "driver" and the "navigator". If the specific roles are not supported via a token, students could end up programming individually, which has a negative impact. Providing a means of communication is another key feature of DPP systems. Since students are not collocated, it's important to provide them a way to communicate. Well known communication means for distributed teams are e-mails, instant messaging services, discussion forums, wikis and audio/video conferences. Most DPP systems support text communication, but an audio channel would be preferable because for programmers it's more convenient to use an audio communication while typing the source code. As mentioned before, the role of the "navigator" is to review the code written by the "driver" and check for errors. He needs to point out code parts, hence gesturing support by means of a remote highlighting feature should also be provided.

Most current tools cover the basic requirements to implement PP over distance. However, we wanted to examine if they support additional features, such as logging capability and shared debugging. Although log files are not proposed by research as a DPP requirement, we believe that by saving and analysing interaction data some useful conclusions can be drawn about students' collaboration and system's DPP implementation. We also examined if collaborative debugging or executing a program is supported, since this phase is very demanding for novice programmers (Cross et al., 2002; Gries et al., 2005).

We present the following tools for DPP that were available to download and install:

- Sangam (http://sangam.sourceforge.net/)
- RIPPLE (http://research.csc.ncsu.edu/ripple/)
- DocShare (http://wiki.eclipse.org/DocShare_Plugin)
- XPairtise (http://sourceforge.net/projects/xpairtise/)
- Saros (http://sourceforge.net/projects/dpp/)
- PEP (http://pep-pp.sourceforge.net/)
- GrewpEdit (http://groupscheme.sourceforge.net/grewpedit/)

Additionally, two applications with shared text editors are presented as an alternative way to implement collaborative programming: collabedit and ec-Coffee.

We excluded out of our presentation systems that we couldn't locate and systems that in our opinion aren't suitable for novice programmers such as Jazz (http://jazz.net/) and wave-vs.net (www.wave-vs.net).

In our study we examined if the above systems meet the requirements of DPP. Specifically, we examined systems' support of the roles of the "driver" and the "navigator", how they're implemented and if there's a mechanism to force or suggest role rotation. We checked also if systems provide a communication channel and gesturing

support, if they keep statistical log files and if they support concurrent debugging and running of programs. Most tools were plugins for Eclipse which is a widely used IDE for software development.

Sangam. Sangam is an open source plugin for Eclipse and supports DPP in Java (Chih-Wei Ho et al., 2004). During one PP session, the two users take the roles of the "driver" and the "navigator" and they can communicate via a chat-window. Only the "driver" is allowed to type in the shared code document while the navigators' typing attempts are ignored and not visible for the driver. Users can switch roles whenever they want to. Another feature of Sangam is allowing users to launch or debug the same Java application at the same time.

RIPPLE. RIPPLE (Remote Interactive Pair Programming and Learning Environment) was developed in order to support DPP, distributed tutoring and data collection for research (Boyer et al., 2008). It extends Sangam by providing a logging capability. When logging is enabled, users' interactions are stored in a database and can be used for further research. RIPPLE was tested in an introductory computer science course during a laboratory assignment. Its evaluation showed that students found it easy to use, they enjoyed the lab assignment and they would use it again if given the opportunity (Boyer et al., 2008).

DocShare. ECF (Eclipse Communication Framework) includes DocShare, an Eclipse plugin that implements real-time shared editing in the Java Editor but it can be modified to support editors for other languages too (e.g. php, C/C++). DocShare offers connections through various providers such XMPP, MSN or IRC so that associated contacts can be retrieved. The roles of the "driver" and the "navigator" do not exist here because both users can type code concurrently in the shared editor. Additional features of DocShare include a chat tool, URL sharing, sending a screen capture or sharing a file. Users may also highlight parts of the code in order to indicate potential problems. Code execution is available only to the user who sends the collaboration request because the shared file is stored to a temporary folder at the second user's workspace.

XPairtise. XPairtise is another plugin for Eclipse which introduces the "spectator" role in addition to the "driver" and "navigator" roles. "Spectators" are not allowed to change or highlight code but they can participate in conversations. XPairtise supports user and session management, remote code highlighting and provides chat communication and a shared whiteboard. When entering a session, each user's local project is compared to the one that is stored in the XPairtise server to ensure that users have synchronised projects. Code saving and execution occur at both users. XPairtise was evaluated during an 18 week period in order to study its contribution in PP sessions. The results showed that students' participation was unequal and that they used rarely the whiteboard and the remote selection feature. Students communicated via an audio channel (Skype) so the chat function was used less than expected (Schümmer and Lukosch, 2009).

Saros. Saros is an Eclipse plugin which introduces some gesturing and awareness functionalities. These functionalities include code highlighting with users' assigned colours when they type or select text, displaying remote cursor and scroll-bar positions, icons indicating if Eclipse is the active window and highlighting which files are opened. (Salinger et al., 2010). The "driver" and "navigator" roles are supported, and furthermore users are allowed to be both "drivers" in order to type code concurrently. Additional features of Saros include file sharing, screen sharing, follow-mode, VoIP (in progress) and chat communication. Saving a shared file works for both users, but not running or debugging a program.

PEP. PEP (Pair Eclipse Programming) is an easy to use Eclipse plugin that implements PP by adopting the roles of "driver" and "navigator". Two users can connect into a PP session, as client or as server, simply by providing the server's IP address. PEP provides a feature to check the projects' synchronism and allows the driver to force project synchronization. An embedded chat is also provided but running and debugging programs doesn't work remotely.

GrewpEdit. GrewpEdit runs as a standalone application and supports collaborative editing and programming (Granville and Hickey, 2005). The roles of the "driver" and the "navigator" are not supported since all users can type concurrently. A shared whiteboard and chat communication are also provided. GrewpEdit supports collaborative programming in Java, C, HTML and Scheme. Program compilation and execution is not common for all users, but everyone can see each user's output in separate tabs.

Another approach to implement collaborative programming is via a shared text editor. However such applications lack important functionalities of most IDE's and are not recommended for systematic software development. Two examples are

Table 1: Comparison of DPP systems' supported features.

Tool	Floor Control	Communication Channel	Gesturing Support	Collaborative Debug & Run	Log Files
Sangam	✓	✓	✓	✓	-
RIPPLE	✓	✓	✓	✓	✓
DocShare	-	✓	✓	-	-
XPairtise	✓	✓	✓	Run	-
Saros	✓	✓	✓	-	-
PEP	✓	✓	✓	-	-
GrewpEdit	-	✓	-	Run	-

collabedit (http://collabedit.com/) and Ec-coFFEE (http://sourceforge.net/apps/trac/coffee-soft/wiki/Ec-CoFFEE).

Collabedit. Collabedit is a web-based collaborative editor which provides the simplest way to implement PP. Each collaborative programming session is given a unique URL, which can be used by other users to enter the session. Collabedit provides a chat area, document history and syntax highlighting according to the chosen programming language, but there's no option to compile or run a program.

Ec-CoFFEE. Although this plugin is not especially designed for PP, it offers very powerful and useful features for student collaborations through Eclipse. Ec-Coffee is based on local area networks and provides following collaborative tools: co-browser, threaded discussion tool, drawing tool, co-writer, chat, document sharing and voting tool. PP can implemented through the co-writer tool, but is limited to the functionalities that a simple text editor can deliver.

In Table 1 we summarize the supported features of the DPP tools we've examined. The term Floor Control indicates if the system supports the roles of the "driver" and the "navigator". We see that two systems do not support role assignments since concurrent code typing is allowed. A communication channel is provided by all systems but is limited to instant messaging exchange. Gesturing support refers here to remote code highlighting, i.e. users' ability to select code parts in order to indicate potential problems. However, three systems support this feature only for the driver (Sangam, RIPPLE and PEP). We found out that only two systems support collaborative debugging and running, while another two systems support only collaborative running. At last, as mentioned before, log files are available only in RIPPLE.

3 ADAPTIVE COLLABORATION SUPPORT

In this section we'll discuss about limitations of previously presented systems as seen from an educational perspective. Considering known issues of PP and reported problems that novice programmers encounter, we suggest how to improve DPP in introductory computer science courses. We present also which features should be embedded in systems for DPP in order to support students' collaboration, and finally we explain why IDE's can't easily incorporate these features.

A major problem in group work, and therefore in PP, is to assess each participant's contribution. To avoid unequal participation several group formation strategies were suggested, including partner assignment by comparable skill levels (Zacharis, 2009), by students preferences (McDowell et al., 2003b), random assignment (Sanjay and Vanshi, 2010) and peer evaluations (Williams et al., 2002). Diziol et al. (2010) suggest that collaborative systems should be able to assess students' contributions to a shared workspace and intervene when unequal participation is observed. In DPP this could be implemented by suggesting role switching or by displaying contribution levels. None of the systems mentioned in the previous section integrates a mechanism to motivate participation. Although RIPPLE keeps a log file, its data is not processed to estimate students' participation.

Another issue similar to equal participation is a balanced knowledge acquisition. When a system detects asymmetries in student's knowledge acquisition, it could intervene to avoid such situations (Diziol et al., 2010). This means that in DPP the system could store each student's written code in order to calculate which commands were used quite often or not at all. If a student avoids usage of a specific command then it could be assumed that he has not yet acquired the appropriate

skills. In order to help the student, the system could provide immediate feedback by showing him additional learning material.

Hanks (2007) conducted a study of problems that novice pair programmers encounter. Most of the problems reported were syntax errors and trivial mechanical problems (e.g. missing semicolons). Another study reported that when students got stuck, they used their book, the Java API or Google to look for assistance (Hanks and Brandt, 2009). Systems for DPP need to incorporate adaptive feedback features in order to cover students' needs. Retrieving respective course material or providing relevant examples could be beneficial for students who face difficulties or don't know how to proceed. Current systems for DPP do not incorporate such features.

Conclusively, none of the systems we've tested did contain a student profile with an underlying student model in order to keep track of student's interactions and collaboration history.

When applied in educational settings, DPP aims to improve students' performance and enhance the learning experience. For this reason we propose to incorporate the suggested collaboration support in an Adaptive System for Collaborative Learning (ASCL). Students' interaction data, their contributions and their skills acquisition could be stored in a student profile inside the ASCL. When adaptive feedback is provided, course material or additional resources could be immediately retrieved from course's site. Teachers could also benefit from such an implementation. Collective data about their students would be available and they could try out different group formation strategies depending on student's profile and collaboration history. Furthermore, some social interaction features could be embedded, like displaying which classmates are online or which group has already completed an assignment. Thus, students that face difficulties, even after adaptive feedback, could seek assistance from their classmates.

4 CONCLUSIONS

In this paper we examined whether current DPP systems eliminate known issues of PP and if they address common problems encountered by novice pair programmers. We found out that none of the available DPP tools incorporates such features. Based on our findings we proposed new features that should be embedded in DPP systems in order to support PP in education.

We noticed that most DPP tools are plugins for the Eclipse IDE and contain similar functionalities. Although they cover the basic requirements for pair programming, none of them keeps log files of students' interactions or provide adaptive feedback. In order to support students' collaboration we propose that systems should integrate a student model, store collaboration data and provide computer mediated adaptive assistance. An Adaptive System for Collaborative Learning seems more convenient for this purpose instead of using an IDE. Furthermore, novice programmers could benefit from such an implementation both from pair programming and from adaptive collaboration support. We aim to investigate this issue in the near future.

REFERENCES

Benaya, T., Zur, E., 2006. Collaborative Programming in an Advanced Programming Workshop Conducted in a Distance Learning Environment, *Methods, Materials and Tools for Programming Education*, 48-55.

Boyer, K. E., Dwight, A. A., Fondren, R. T., Vouk, M. A., Lester, J. C., 2008. A Development Environment for Distributed Synchronous Collaborative Programming, In *Proceedings of the 13th annual conference on Innovation and technology in computer science education,* ACM, New York, NY, USA, 158-162.

Cockburn, A., Williams, L., 2001. The costs and benefits of pair programming. In *Extreme programming examined*, Giancarlo Succi and Michele Marchesi (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA 223-243.

Cross, J. H. I. I., Hendrix, T. D., Barowski, L. A., 2002. Using the debugger as an integral part of teaching CS1, *Frontiers in Education, 2002. FIE 2002. 32nd Annual*, vol.2, no., pp. F1G-1- F1G-6 vol.2, 6-9 Nov. 2002.

Diziol, D., Walker, E., Rummel, N., Koedinger, K. R., 2010. Using intelligent tutor technology to implement adaptive support for student collaboration. *Educational Psychology Review.*

Duque, R., Bravo, C., 2008. Analyzing Work Productivity and Program Quality in Collaborative Programming. In *Proceedings of the 2008 The Third International Conference on Software Engineering Advances (ICSEA '08)*. IEEE Computer Society, Washington, DC, USA, 270-276.

Granville, K., Hickey, T. J., 2005. The design, implementation, and application of the grewpEdit tool. *Proceedings of the 2005 conference on Diversity in computing*, Albuquerque, New Mexico, USA.

Gries, P., Mnih, V., Taylor, J., Wilson, G., Zamparo, L., 2005. Memview: a pedagogically-motivated visual debugger. *Frontiers in Education,2005 FIE '05.*

Proceedings 35th Annual Conference, vol., no., pp.S1J-11, 19-22 Oct. 2005.

Hanks, B., 2008. Empirical evaluation of distributed pair programming. *Int. J. Hum.-Comput. Stud*. 66, 7 (July 2008), 530-544.

Hanks, B., 2007. Problems encountered by novice pair programmers. In *Proceedings of the third international workshop on Computing education research (ICER '07)*. ACM, New York, NY, USA, 159-164.

Hanks, B., Brandt, M., 2009. Successful and unsuccessful problem solving approaches of novice programmers. In *Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09)*. ACM, New York, NY, USA, 24-28.

Ho, C. W., Raha, S., Gehringer, E., Williams, L., 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.

Jun, S., Kim, S., Lee, W., 2007. Online Pair-Programming for Learning Programming of Novices. *WSEAS TRANSACTIONS on ADVANCES in ENGINEERING EDUCATION*, Issue 9, Volume 4, September 2007.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., Thomas, L., 2004. A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '04)*. ACM, New York, NY, USA, 119-150.

McDowell, C., Hanks, B., Werner, L., 2003a. Experimenting with pair programming in the classroom. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education (ITiCSE '03)*, David Finkel (Ed.). ACM, New York, NY, USA, 60-64.

McDowell, C., Werner, L., Bullock, H. E., Fernald, J., 2003b. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 602-607.

Preston, D., 2005. Pair programming as a model of collaborative learning: A review of the research, *Consortium for Computing Sciences in Colleges*, 39-45.

Salinger, S., Oezbek, C., Beecher, K., Schenk, J., 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.

Sanjay, G., Vanshi, K., 2010. A Novel Approach for Collaborative Pair Programming. *Journal of Information Technology Education*, USA, Vol. 9, 183-196.

Schümmer, T., Lukosch, S., 2009. Understanding Tools and Practices for Distributed Pair Programming. *Journal of Universal Computer Science*, vol. 15, no. 16, 3101-3125.

Williams, L., Yang, K., Wiebe, E., Ferzli, M., Miller, C., 2002. Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations. *OOPSLA Educator's Symposium*, pages 20-26.

Winkler, D., Biffl, S., Kaltenbach, A., 2010. Evaluating Tools that Support Pair Programming in a Distributed Engineering Environment. *Conference on Evaluation and Assessment in Software Engineering (EASE)*, Keele, Great Britain.

Zacharis, N., 2009. Evaluating the Effects of Virtual Pair Programming on Students' Achievement and Satisfaction. *International Journal Of Emerging Technologies In Learning (IJET)*, 4(3).