Informatics in Education, 2015, Vol. 14, No. 2, 291–314

DOI: 10.15388/infedu.2015.17

Distributed Pair Programming Using Collaboration Scripts: An Educational System and Initial Results

Despina TSOMPANOUDI, Maya SATRATZEMI, Stelios XINOGALOS
Department of Applied Informatics, School of Information Sciences, University of Macedonia Thessaloniki, Greece
e-mail: {despinats, maya, stelios}@uom.edu.gr

Received: August 2014

Abstract. Since pair programming appeared in the literature as an effective method of teaching computer programming, many systems were developed to cover the application of pair programming over distance. Today's systems serve personal, professional and educational purposes allowing distributed teams to work together on the same programming project. The current research focuses in distributed pair programming systems which are suitable for supporting students in learning computer programming. Systematic review of publicly available systems revealed that there is an absence of effective collaboration support for the students. The main drawbacks of pair programming, such as uneven workload distribution and infrequent role switches, cannot be addressed with available systems. While building an enhanced version of a distributed pair programming system, successful instructional strategies in similar collaborative learning systems were explored, in order to improve students' interactions when applying pair programming over distance. As a result, the new system allows students to practice distributed pair programming in the form of collaboration scripts. This paper presents the features and the underlying concepts of the system, and the results of its first evaluation. The study showed that distributed pair programming attracted positive feedback from students, and that scripted collaboration affected students' engagement in programming, and resulted in an evenly distribution of learning objectives among pairs.

Keywords: distributed pair programming, collaboration scripts, collaborative programming, adaptive collaboration support.

1. Introduction

Distributed Pair Programming (DPP) systems allow two programmers to collaborate remotely in order to apply the Pair Programming (PP) technique from separate locations. The model of PP originated from the software industry as a part of Extreme Programming (XP). It involves two programmers working on the same workstation and sharing one computer in order to develop software. While pair programming, the team

members adopt two specific roles: one programmer acts as the "driver" and the other one as the "navigator" (also called "observer"). The driver has possession of keyboard and mouse and types the programming code. The navigator reviews the inserted code and gives guidelines to the driver. Driver and navigator are in constant collaboration in order to design and develop the program code in common, while they should frequently alternate roles.

This model of programming was reported to have many benefits compared to solo programming. Most research studies from software industry and academia indicate positive effects of PP on programmers' performance and software quality. Williams *et al.* (2008) studied several years the application of PP in the classroom. They found that the collaborative nature of PP helps students to achieve advanced learning outcomes, to be more confident and to receive better grades in programming assignments. Other studies indicate that PP leads in higher program quality, continuous knowledge transfer and more student enjoyment (Faja, 2011). Another notable benefit is the fact that in PP students work in teams like professional programmers, meaning that they are trained in a professional style of programming and develop teamwork skills – an essential qualification for their future career. Despite its many benefits, research suggests that students should be trained in the proper practice of PP in order to gain the most from this methodology (Williams *et al.*, 2008). Pair incompatibility is also a factor that influences significantly the progress and outcome of the collaboration, with many studies suggesting pairing students based on similar skill levels (Faja, 2011). Examining students' experiences on PP, the majority report scheduling conflicts as a major drawback of PP (Faja, 2011).

The development of various DPP systems covered the problem of aforementioned scheduling conflicts and gave new perspectives to the application of PP in industry and education. DPP systems make remote collaboration in distance education feasible, while they preserve most advantages of PP (Hanks, 2008; Stotts *et al.*, 2003). They allow geographically distributed teams to design and develop software projects remotely, and facilitate intercommunication and collaboration among team members. In order to cover the different requirements and demands of end-users, various types of DPP systems exist. Some systems were developed only for educational purposes, while more complicated and integrated systems aim to serve the needs of professional teams. Recently, a lot of web-based collaborative editors appeared, which enable two or more users to share and edit program code in real time. Although they provide the easiest way to pair program, they have limited capabilities compared to other applications.

The problem of unequal participation is a common issue in collaborative learning and has also been faced in PP. Williams (2007) describes non-participation as the most common problem with pairing, and suggests to use peer evaluations in order to motivate students' participation. Schuemmer and Lukosch (2009) evaluated students' contributions using a DPP system and observed unequal participation levels and infrequent role switches. Since PP is also a form of collaborative learning (Preston, 2005), the adoption of successful instructional approaches from collaborative learning within the field of PP or DPP constitutes a promising way to address these issues. Research indicates that it is very important to structure the collaborative learning process in order to foster learning

and interaction (Dillenbourg and Fischer, 2007). And a promising teaching approach to achieve this, is the use of collaboration scripts. They shape collaborative learning activities by providing a framework which determines several aspects of the collaboration. Collaboration scripts have a wide range of applicability and they have been studied in the context of other problem-solving fields like mathematics and chemistry (Diziol *et al.*, 2007; Tsovaltzi *et al.*, 2010). As described by Kobbe *et al.* (2007), a collaboration script contains a number of components and mechanisms and its goal is to guide students through the collaborative learning process. In more detail, script components consist of a detailed description of participants, activities, roles, groups and resources, and script mechanisms describe group formation strategies, task distribution and task sequencing. Collaboration scripts are often embedded in Computer Supported Collaborative Learning (CSCL) systems where the computer guides students through the sequence of the script phases. In this case, the scripts are prepared within the CSCL environment and educators are responsible to define its components and mechanisms.

Considering the benefits of collaboration scripts it was attempted to incorporate them within the DPP context. For this purpose we developed SCEPPSys (which stands for "Scripted Collaboration in an Educational Pair Programming System" and is pronounced like the Greek word skepsis), an educational DPP system. Instead of allowing free collaboration during DPP sessions, the workflow was structured by means of scripted collaboration. This approach aims to address the most important drawback of PP, namely unbalanced student participation. To achieve this, collaboration scripts are used, in order to distribute programming tasks among pair members and to assign the roles of the driver and navigator. To the best of our knowledge collaboration scripts have not been studied yet as a part of DPP, therefore the current study fills a gap in this regard. In order to study the impact of this approach an evaluation study of the produced system was conducted. The findings confirmed the feasibility of the proposed methodology and showed that collaboration scripts affected students' interactions in a positive way.

This paper introduces the concepts of scripted DPP and presents preliminary results of its first evaluation. The primary contribution of the study is the combination of two different research areas, namely DPP and Collaboration Scripts. The paper includes a comprehensive overview of this novel approach, including related work, a presentation of the supporting system and the outcomes of a controlled study. The results of scripted DPP revealed an increased number of role switches and comparable contribution levels regarding students' interactions. To conclude, this work provides a framework to apply DPP in the classroom, and a possible solution to address the most common drawbacks of PP.

The remaining article is organized as follows. In the next section follows a presentation of related systems that facilitate PP or collaborative programming over distance. Collaborative programming is referred when a system supports two or more programmers simultaneously. Then, the features and the environment of SCEPPSys are presented (Section 3). Section 4 contains the experimental design and the results of the system evaluation. A discussion follows in Section 5, and in the last section we conclude with our suggestions and future directions (Section 6).

2. Related Work

This section contains the related work on DPP systems and scripted collaboration. First the several types of DPP systems are presented, and a comparison between them aims to identify some standard features of DPP systems and to highlight their limitations. Then a brief description of authoring tools for collaboration scripts follows, which presents the main characteristics of those systems.

As previously mentioned there are three types of DPP systems that serve personal, professional and educational purposes. The most easy-to-use and convenient implementations for personal use are nowadays web-based collaborative editors. Most of them are freely available and allow two or more programmers to share a real-time editor in order to develop or exchange program code. They support syntax highlighting for many programming languages but they usually do not offer compiling and execution actions. Communication is mediated via a chat area. Examples of such editors are Squad[1] and Collabedit[2]. On the other hand, professional DPP systems like Visual Studio Anywhere[3] are designed to support the lifecycle of large projects and thus more complicated for novices. Another approach to apply DPP is to use a desktop sharing application like TeamViewer[4]. Its main advantage is that it enables remote collaboration in every development environment, but the shared program is saved only in one computer and connection delays may also hinder the programming process. The current research explores systems that are more suitable for students, or were especially designed for educational purposes. Desktop sharing applications were excluded because they lack educational features.

Earlier studies of DPP in the classroom used desktop sharing applications in order to test the efficacy of DPP over co-located PP (Baheti *et al.*, 2002; Stotts *et al.*, 2003). Over the next years several DPP-oriented systems appeared. The GrewpEdit tool (Granville and Hickey, 2005) was designed to support collaborative programming in computer science courses and includes a shared code editor, a shared whiteboard and a chat. COPPER (Favela *et al.*, 2004) is another application developed to support DPP. It involves an editor which can be run in individual or synchronous collaborative mode. When working in collaborative mode, a floor control mechanism and basic awareness features facilitate the application of DPP sessions. COLLECE (Bravo *et al.*, 2013) is a powerful educational tool for collaborative programming which provides logging capabilities and analysis of users' interactions. It contains a subsystem to organize programming activities and provides feedback to the users about their activities (Duque *et al.*, 2011).

Besides the standalone applications, other implementations of DPP are embedded within Integrated Development Environments (IDE). Especially for the Eclipse IDE there is a significant number of available plugins. Among them, RIPPLE (Boyer *et al.*, 2008) and XPairtise (Schuemmer and Lukosch, 2009) were also used in academic studies. RIPPLE, which is an extension of Sangam (Ho *et al.*, 2004), was tested in an introductory computer science course during a laboratory assignment. Its evaluation showed

[1] http://squadedit.com
[2] http://collabedit.com
[3] http://vsanywhere.com
[4] http://www.teamviewer.com

that students found it easy to use, they enjoyed the lab assignment and they would use it again if given the opportunity. XPairtise was evaluated during an 18-week period in order to study its contribution in DPP sessions. The results showed that role switches did not occur as often as expected and that students used rarely the remote selection feature. Students communicated via an audio channel (Skype) so that the chat function was used less than expected. Saros (Salinger *et al.*, 2010) is a plugin still under ongoing research and XecliP (Schuemmer and Lukosch, 2009) was built by the research group of XPairtise and has similar functionalities. Most Eclipse plugins support the standard requirements of DPP, namely a real-time shared editor, floor control policies, communication tools, remote code highlighting and awareness features. RIPPLE adds a logging capability of users' interactions, allowing researchers to reconstruct user sessions for further study. However, the logged information is provided only as raw data and further processing is required in order to extract useful information. Finally, COLE-Programming (Jurado *et al.*, 2013) is a plugin which extends the COALA (Jurado *et al.*, 2009) environment by adding some collaborative tools like a chat, a forum and a voting pool but DPP is not a feature of the system.

In the field of collaborative learning, "scripted roles" are considered predefined role assignments, which are performed by the instructor, in order to equally engage students in relevant roles and activities. Compared to "emerging roles", which are spontaneous role assignments performed by the students, "scripted roles" aim to an equal distribution of tasks and roles among group members (Strijbos and Weinberger, 2010). Computer-supported collaboration scripts can facilitate task distribution and role rotation in order to engage students in diverse roles and activities. This process is supported in authoring tools for computer-supported collaboration scripts which are general purpose editors, because they facilitate the creation of pedagogical scenarios in various areas that involve collaborative learning. Such tools are RELOAD (Milligan *et al.*, 2005) and Collage (Hernández-Leo *et al.*, 2006). These tools help teachers to create a sequence of collaborative learning activities by defining learning objectives, participants, roles, tasks, resources etc. A runtime environment is then needed in order to interpret and execute the script.

Table 1 summarizes the features of current DPP plugins. As shown, DPP applications typically do not contain functionalities that enable the creation of collaboration scripts. Neither do they integrate sufficient interaction analysis and support. Therefore

Table 1
Comparison of DPP plugins

Tool	Support of the standard DPP requirements (shared editor, floor control, communication, remote highlighting, awareness)	Log Files	Interaction Analysis	Collaboration Scripts	Evaluation
RIPPLE	✓	✓	–	–	✓
Sangam	✓	–	–	–	–
XPairtise	✓	–	–	–	✓
XecliP	✓	–	–	–	–
Saros	✓	–	–	–	–

DPP plugins need further improvement and more evaluation studies in order to increase their effectiveness.

3. SCEPPSys Design and Implementation

This section contains an overview of SCEPPSys' architecture and its components. First the requirements of DPP systems are listed, and then the main characteristics of SCEPPSys' development are presented. In the last part it is demonstrated, visually and practically, how a DPP session is performed.

A typical DPP system incorporates some standard features in order to meet the requirements of distant collaboration. At first, a real-time shared editor is required to edit the source code. In order to adopt the roles of the driver and navigator, a floor control mechanism must ensure that only one user is allowed to insert and change the program code at a time. Since PP is based on close collaboration and has a specific settlement, concurrent changes in the editor is not a desired feature. In DPP systems, driver actions like editing, executing or opening a file are also replicated to the workspace of the navigator. An embedded text-based or audio-based communication channel enables pair programmers to discuss and coordinate their actions. Another basic feature of DPP systems is the support of remote code highlighting. It enables the driver or the navigator to point out code parts in order to indicate a problem or syntax error. At last, most DPP systems incorporate basic awareness features indicating the status of the users. For example, when a user saves the code or types a message, accordingly status messages appear on partner's side. Most educational DPP systems support these standard features.

For our implementation, a system which embedded the aforementioned requirements was chosen. The new system was based on an existing Eclipse plugin because standalone applications lack important features of common IDEs. Eclipse is a very popular development environment and widely used in computer science courses. Besides, students attending the Java class in our university also work with the Eclipse IDE. In addition, most Eclipse plugins are open-source and freely available. All available open-source plugins were tested and the XecliP plugin was chosen as a basis for SCEPPSys.

SCEPPSys is based on a typical client-server architecture. Its distribution consists of a server, a database and the Eclipse plugin. Students only need to install the plugin in order to run SCEPPSys. As depicted in Fig. 1, the server is responsible to dispatch messages between the clients. The database stores users' accounts, shared projects, statistics and all necessary information about groups, courses and assignments. The teacher uses the server and the database in order to set up the whole DPP system. A web-based administration environment provides access to server and database (Tsompanoudi and Satratzemi, 2014).

In its initial design, the plugin supported DPP with a floor control mechanism which adopted the roles of the driver and navigator. Only the driver was allowed to edit the source code and all actions were transferred and replicated on driver's side, including opening, closing and saving several editors. The navigator was able to highlight parts of the source code in order to point out potential problems. Both users could request a role

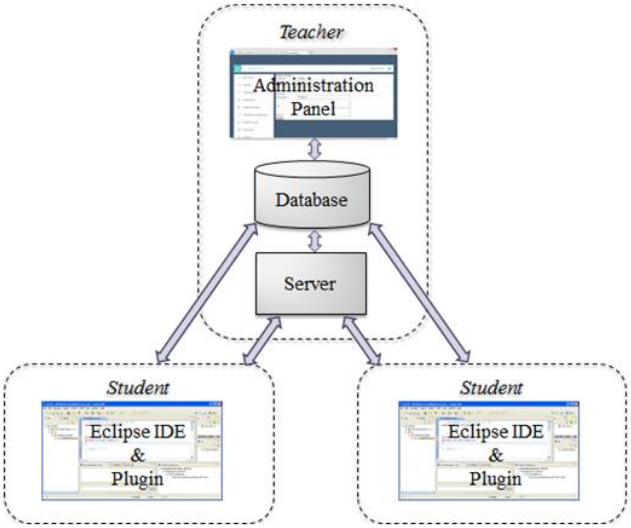

Fig. 1. System architecture.

switch and communication was mediated via a chat area. The first addition made, was the ability of synchronic program execution because it is a frequently performed action by the driver and an essential part of the coding process. As a result, this feature allows both users to view the output when the driver executes the code. Another performed change was the way users share a common project by providing a simpler solution. The former way to share a project was mediated via a CVS repository, which was considered a complicated solution for novice programmers. The remaining additions and changes made, aimed to adapt the user interface in order to pair program guided by collaboration scripts and to create an administration panel for the teacher.

The definition of collaboration scripts requires an authoring tool which was built as a part of the administration section. It allows the teacher to manage his courses and to organize programming assignments in the form of collaboration scripts. A collaboration script involves the definition of participants, groups, activities, roles and policies for group formation and task distribution. In the case of DPP, participants are all students enrolled in a course whose data is entered in the system. Students are grouped in pairs depending on teacher's selected group formation strategy. Currently the system supports four group formation policies: random groups, free selection, comparable skill levels and comparable contribution levels. The creation of activities involves a different approach of solving programming assignments. Instead of providing to the students a whole description of the assignment, the programming assignment is decomposed into smaller problem-solving tasks. This approach helps to categorize each task in learning goals and to better regulate collaboration during DPP. It also aims to assist students through the problem-solving process, because most novices face difficulties when asked to apply acquired knowledge. In order to distribute tasks among students, a task-role distribution policy must be defined by the teacher. A "rotating role switching" policy intends to obtain equal number of tasks per student and successive role alternations,

while a "balanced knowledge" policy aims to achieve symmetry in skill acquisition. The definition of all these parameters constitutes the components and mechanisms of a collaboration script which will guide the pairs during the problem-solving process.

In order for the teacher to monitor the outcomes of students' collaboration the administration panel contains a section with statistical information about submitted projects per group and user level. The system calculates students' contribution in each programming assignment and keeps track of several collaboration related factors. Contribution is calculated by means of users' driving time or the amount of inserted program code. Collaboration is indicated by the number of: exchanged messages, program executions, role switches, retrieved hints and the distribution of tasks among pairs. The calculation of such parameters aims to assist to the evaluation of group projects, where the contribution of each individual member is hard to assess. Additionally, teachers may examine various group formation strategies or task distribution policies and monitor the impact on participation and collaboration. The remaining administrative functions of the system serve the assessment of the submitted programming assignments.

SCEPPSys has a customized Eclipse perspective which handles the embedded features of the plugin. A screenshot of users' interface is depicted in Fig. 2. In the left area is located Eclipse's Package Explorer (Fig. 2 (a)) and SCEPPSys' driver/navigator chat (Fig. 2 (b)). The shared project is automatically created during session start in the Package Explorer. Each class of the project has a collaborative editor in the middle area of the workspace (Fig. 2 (c)). The right area is used to display online users (Fig. 2 (e)), running sessions (Fig. 2 (f)), users' requests and status (Fig. 2 (e) & (f)) and the console window (Fig. 2 (g)). While a DPP session is running, an additional view is created which displays the programming tasks of the current assignment (Fig. 2 (d)). In order to better understand the workflow of a typical DPP session the diagram depicted in Fig. 3 was created. As shown in the figure, a DPP session has five phases:

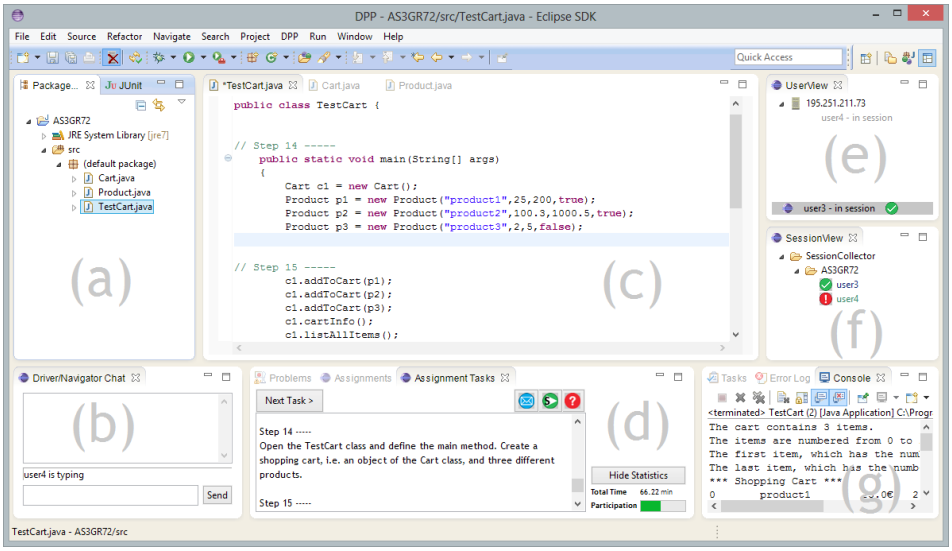

Fig. 2. Screenshot of a DPP session.

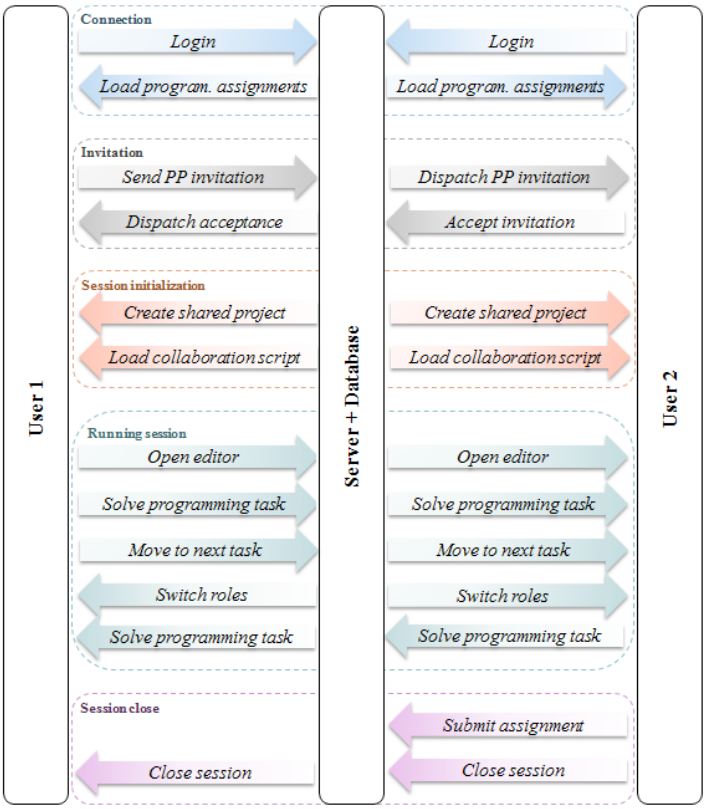

Fig. 3. Workflow of a DPP session.

- *Connection*: Students (User1 and User2 in the figure) connect to the server and after successful authentication the plugin loads an outline of course's programming assignments. If there is an unsolved or unfinished assignment, the name of the team mate is listed near the activity.
- *Invitation*: In order to start a DPP session both team members must be online. One of them sends a DPP invitation to the other and waits for his response. If the invitee accepts the invitation, his response is dispatched to his partner and the session initialization process begins.
- *Session initialization*: The shared project is created by the system in the workspace of both students. If the students resume a saved session, the respective project is loaded from the database. Meanwhile, an additional view representing the collaboration script opens, in order to display the programming tasks to be performed by the students.
- *Running session*: The student who initiates a DPP session is assigned the driver role and is responsible to open a Java file and begin programming. These actions are dispatched and performed via the server in the workspace of the navigator. When a programming task is completed, the driver moves to the next task. This

action may trigger a role switch by the system. The student in the driver position continues with the problem-solving process.

- *Session close*: A DPP session is closed when students decide to submit their solution or to continue the session at another time. On a session close message, students' interaction data is saved in the database. Users may then disconnect from the server.

During a DPP session both students may request at any time individual role switches since role assignment is not mandatory. Drivers' actions like saving and running the program code appear in both workspaces and students can communicate and exchange their ideas using the chat area. In order to help weak students, a hint of the solution for every programming task was embedded. If a user decides to retrieve a hint, a message is displayed only in his workspace and his partner is not notified about this action. Lastly, during this phase the system displays the session time and the individual contribution rate in order encourage students' participation. The visualization of individual participation rates in CSCL environments is not new and studies reported positive findings in regard to students' motivation (Avouris *et al.*, 2004; Janssen *et al.*, 2007).

Conclusively, SCEPPSys extends existing plugins by combining collaboration scripts and DPP. Furthermore, a main advantage of the system is its enhanced logging capability of users' interactions, which facilitates the evaluation process of DPP within an educational context. In fact, the current study was performed using SCEPPSys, and the presented results complement limited findings from former DPP studies.

4. System Evaluation and Results

4.1. *Experimental Design*

Prior to the evaluation presented in this section a pilot study of a prototype of the system was conducted (Tsompanoudi *et al.*, 2013). The first study aimed to test the consistency of the system and was held during one laboratory session. Twenty students, attending a Java class at that time, formed ten random groups and were asked to solve a basic programming assignment. Students were placed in separate locations in order to apply DPP. This pilot study gave a first insight of system's usability. Although no technical problems occurred, a system bug was detected and students' comments for future reference were recorded.

After minor corrections in the software and user interface, the main evaluation was conducted. The study lasted over a period of 8 weeks. Students with basic knowledge of the Java programming language volunteered to participate in the study. Forty-eight students (10 females and 38 males) attended the study, and were rewarded with extra credits, on condition that they had completed all phases of the evaluation. 10% were first-year students, 23% were second-year students, 27% were third-year students, 11% were fourth-year students and the remaining 29% were final year students, all majoring in Computer Science. Most participants had taken a Java course in the past, but failed to pass the final exams. The evaluation was organized as follows. Two meetings with the

students were scheduled, one at beginning of the study and another one during the final week. At the first meeting the DPP methodology was presented to the students along with a demonstration of the setup and the main functions of the system. Students then filled out a questionnaire which assessed their knowledge level in Java. The test contained open-ended questions which were based on given program code and some multiple-choice questions which focused on program execution and output. The following weeks, students were assigned five projects and used SCEPPSys in order to solve and submit the assignments. The projects covered the topics of Java fundamentals, object-oriented programming, collections, inheritance and polymorphism, and respective learning material was provided to the students. During the last meeting a second assessment of students' Java skills was conducted, and they were given a questionnaire in order to express their opinions about the usefulness of the system.

The forty-eight students were grouped in pairs, either based on their preferences or on similar knowledge level if they did not express a preference. They were discouraged to work in adjacent computers since the IP's of each group were regularly checked in order to ensure the proper application of DPP. Data gathered from the questionnaire revealed that most students had attended a Java course in the past and were familiar with the Eclipse IDE. Almost half of them stated to have worked in teams in programming assignments, but only two students claimed to have experience in DPP. Thus for the majority of the students the application of DPP was a new challenge.

The 24 pairs were divided in two groups which worked with different versions of the plugin. Both groups solved the same programming assignments using a collaboration script which differed in the part of the task distribution mechanism. Pairs of the first group arranged the driver and navigator roles by themselves, while in the second group the system initiated role switches based on the "balanced knowledge acquisition" policy. Additionally, the plugin of the second group displayed the total session time and the participation rate of each student. The primary goal of the study was to evaluate the impact of collaboration scripts in DPP. By dividing students into two groups it was also feasible to evaluate free collaboration in comparison to structured collaboration. Conclusively, the two groups formed two experimental conditions. The control group consisted of pairs allowed to regulate collaboration by themselves and the experimental group followed a scheme of structured collaboration.

The research questions of the evaluation study were organized in two subcategories concerning two different areas of interest. The main subject of the first category was to investigate the role of collaboration scripts in DPP, focusing in students' overall participation and performance. This part of the study involved the definition of the following research questions:

(Q1): *Does structured collaboration lead to more balanced student contributions compared to free collaboration?* (Section 4.2)

(Q2): *Did a balanced distribution of tasks among pairs achieve symmetry in knowledge acquisition?* (Section 4.3)

(Q3): *Does structured collaboration influence the completion time of an assignment compared to free collaboration?* (Section 4.4)

(Q4): *Did the use of the system have an effect on students' knowledge level?* (Section 4.5)

(Q5): *Does structured collaboration lead to better learning outcomes compared to free collaboration?* (Section 4.5)

(Q6): *Do students who are guided by structured collaboration achieve better assignment scores than students who are allowed to collaborate freely?* (Section 4.6)

The main subject of the second category was to evaluate the features of the new system and its acceptance by students. The presented results aim to answer the following research question:

(Q7): *What are students' perceptions of the system and the application of DPP?* (Section 4.7)

The results of all these studies are presented in the following subsections.

4.2. *Evaluating Interaction*

(Q1): *Does structured collaboration lead to more balanced student contributions compared to free collaboration?*

In this section students' actions in the workspace were studied in order to evaluate collaboration. This part of the study involved:

(a) the number of exchanged chat messages,
(b) the number of synchronic program executions,
(c) the number of hints requested,
(d) the number of role switches, and finally
(e) the amount of inserted program code.

Due to the fact that the majority of pairs used applications like Skype and Facebook in order to communicate with each other, it was not possible to analyze communication information. Similarly, it was observed that some groups did not use the synchronic program execution feature. Instead, they preferred to run programs locally and used the available Eclipse options. Thus, it was not possible to extract valuable information from this feature as well.

The integration of help messages (hints) in each programming task aimed to support weak students. Since the main reason of disengagement in collaborative work is lack of knowledge or incomparable skill level (Chaparro *et al.*, 2005; Plonka *et al.*, 2012), this feature intended to motivate students in active participation. For this reason, hint requests were hidden from partner's workspace in order to encourage their use. Surprisingly, only a few groups out of 24 took advantage of the embedded help. Students stated in the questionnaire that they did not use hints because they preferred to make their own efforts instead of getting a hint of the solution.

An indicator of good collaboration is a high frequency of role alternations between the pair members. The system tracks the number of role switches in each session, thus

the frequencies for each condition were compared. The experimental group, whose role switches were initiated by the system or manually, had in average a higher number of role changes in each project (Fig. 4). The control group had at least 4.8 role changes in average and 8.5 at maximum. An independent samples test for each project was conducted and a statistically significant difference between experimental and control group was found in all projects ($p < 0.05$).

System's interventions in turn taking forced students of the experimental group to be more participating. In contrast, analysis of role switches in the control condition revealed that some groups made only one or two role switches in each project. Students of the control group were asked how they did arrange role switches during DPP sessions. They indicated three different strategies. Primarily, each student selected and solved tasks that he found most undemanding. Secondly, they managed to solve approximately an equal number of tasks. The third strategy was to change roles whenever the driver got tired. Since students in the experimental group were also allowed to initiate role changes, they were asked about the main reasons requesting them. They indicated that they primarily changed roles when the navigator expressed a request to solve a programming task or to make a correction, and whenever they felt that the system made uneven role distributions. A look at logged session data revealed that 3 out of 12 pairs (25%) kept system's role assignments. This finding is not surprising since corrections in the program code may influence users' participation in each individual step. Interaction data of these three groups was further analyzed in order to study outcomes under these conditions. The results revealed optimal interaction results regarding participation, task distribution and knowledge acquisition. Each student had at least 43% participation in the program code. The difference in amount of tasks and learning goals within pairs was also minimal. The results of these three pairs confirm the proper operation of the system and illustrate the usefulness of the selected algorithms. Nevertheless, it may not always be possible to achieve ideal collaboration conditions. But an attempt towards this direction certainly approaches the desired outcomes.

The last study in this section involved an estimation of users' contribution rate. This raises the question of how students' involvement in such a collaborative activity can be measured. One can take into account the driving time of each user or the code he pro-

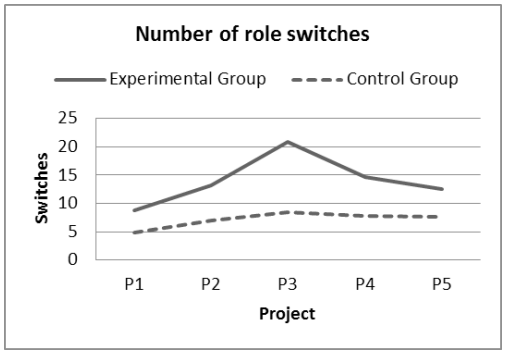

Fig. 4. Role switches per condition.

duces (Gardner, 2003). On the other hand the role of the navigator should not be underestimated. Although it is difficult to assess navigator's contribution to a group project, one possible solution is to consider the number and the quality of sent communication messages. Bravo *et al.* (2013) incorporated a structured chat (with sentence openers) within the COLLECE system. It did not gain the expected benefits since only 3% of sent messages were structured and 92% were free messages. Since most pairs in our study used other means rather the embedded chat to communicate with each other, it was not possible to count messages or to analyze the content. However, the system saved the driving and non-driving time of each user and also his contribution in the program code. It was found out that driving time and produced code were not always proportional. For this reason the total time a user spends in the driver role was not considered as an accurate value to conclude his contribution. In order to evaluate students' contribution, only the produced program code was taken into account. The goal of this part of the study was to investigate if structured collaboration led to more balanced student contributions compared to free collaboration (research question Q1).

The system was designed to save in its database the files of each submitted project. It also distinguishes the content of each file depending on the user who typed the program code. The contribution rate for each student is calculated as the ratio of the typed code to the total code. First the difference between individual contribution rates within each group was calculated. This number is limited from 0 to 1 and it indicates balanced contribution when its value is closer to zero. Then the average value of this difference over the five projects was calculated. It was found that the control group had an average difference of 0.243, while the experimental group had an average difference of 0.165. Values within the range 0 and 0.2, namely a participation rate between 40% and 60%, are sufficient to indicate symmetry in contribution. The experimental group achieved a main difference within the desired boundaries. A Mann-Whitney test (Fay and Proschan, 2010) was run in order to examine if the experimental condition had a statistically significant difference from the control group. The test indicated that the difference between control and experimental condition was not significant ($p > 0.05$). To answer the first research question, based on statistical analysis it cannot be concluded that structured collaboration leads to more balanced student contributions.

As depicted in Fig. 5 and Fig. 6, in both conditions 67% of the students had an average contribution rate within the acceptable range. A significant percentage (25%) of the control condition revealed disengagement on the part of one student, namely a participation rate less than 30%. Such asymmetries were not detected in the experimental condition due to system's interventions. Conclusively, both conditions achieved a high percentage of evenly distributed contributions except that the control group had also some extreme cases.

Summarizing the findings of interaction analysis, it can be concluded that automatic role assignments resulted to a higher number of role switches between the pairs in the experimental condition. Students of the experimental groups rotated more often the roles of driver and navigator in contrast to the students of control groups. Frequent role switches are considered to be more effective in DPP because users interact more with each other without getting tired. Furthermore, Plonka *et al.* (2011) showed that frequent

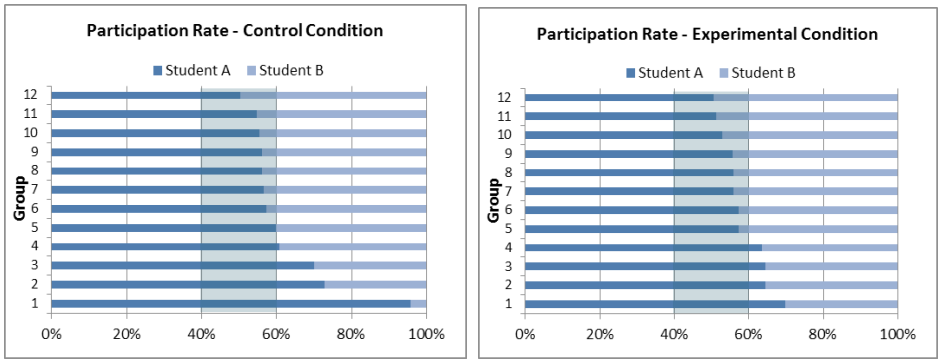

Fig. 5. Participation rates in each condition.

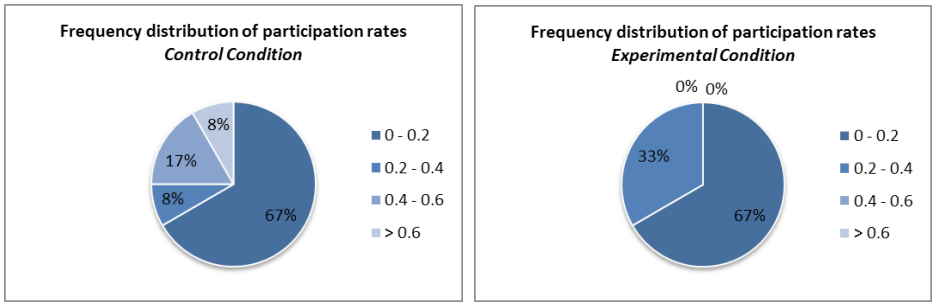

Fig. 6. Frequency distribution of differences between participation rates.

role switches indicate a high level of engagement on the part of both programmers. Despite the number of role switches, the two conditions had not a statistically significant difference in students' participation rates. Finally, it's worth mentioning that it was not requested from students to achieve equal contribution rates in order to study a natural behavior during collaboration.

4.3. *Evaluating Task Distribution*

(Q2): *Did a balanced distribution of tasks among pairs achieve symmetry in knowledge acquisition?*

As previously described, dividing the programming assignments into smaller problem-solving tasks allows the teacher to assign to each task a specific learning goal (e.g. constructor, conditional statement, getter method etc.). The system exploits this categorization in the task distribution mechanism of the collaboration script. This strategy aims to prevent students from solving tasks of the same category on a regular basis. Furthermore, each student solves practically the same amount of tasks. The goal is to achieve symmetry in skill acquisition because students tend to divide work in group projects

depending on what they already know, losing thus the opportunity to learn new skills (Barker, 2005). The "balanced knowledge acquisition" policy was applied in the experimental group, which triggered role changes during DPP sessions. The control group regulated role switches individually.

In order to investigate the second research question (Q2), first the number of tasks per student for each learning goal was counted. Then the difference from an evenly distribution was calculated. For all learning goals (summative for all projects) there was a greater difference in the control group compared to the experimental group. A Mann-Whitney test revealed that the average difference between the two groups was also statistically significant ($U = 25.5$, $p = 0.007$), meaning that system's interventions during DPP had a positive effect on task distribution. On the other side, free collaboration led to unbalanced task distribution, which was also confirmed by students' responses in the evaluation questionnaire. They revealed that each of them solved similar tasks in order to avoid difficulties. Regarding research question (Q2), the results showed that the balanced distribution of tasks among pairs achieved symmetry in knowledge acquisition.

4.4. *Examining Completion Time*

(Q3): *Does structured collaboration influence the completion time of an assignment compared to free collaboration?*

The system calculates the total time a team spends in solving a programming assignment and the driving/non-driving times for each user. Driving time is the time the user spends exclusively in typing the program code and non-driving time is the time used for other actions like communication, coordination and other interactions. Having this data available, two different subjects of interest were examined. The first aim was to compare driving and completion times of the two conditions, and the second aim was to analyze the relationship of total and driving time. Regarding the first goal the results were used in order to answer the third research question (Q3). The task distribution mechanism which was applied in the experimental group, forced several role switches during a DPP session. It was expected that these role changes would have caused an overhead in the time spent to solve the programming assignments. In contrast to our expectations, the experimental group achieved in average a shorter completion and driving time in each project (Fig. 7). However, analysis of this data showed a statistically significant difference only in the last project (t-test results: $t(22) = -3.995$, $p = 0.001$). Conclusively, regarding research question (Q3), structured collaboration does not influence the completion time of assignments compared to free collaboration.

The relationship between driving and total time (Fig. 7) showed that the control group started with 32% of the total time driving and reached 40% in the last project. The experimental group spent 29% of the total time driving in the first project and 37% in the fifth. We notice a slight increase in driving time as students use the system. This suggests that students become familiar with the system and the application of DPP over time. Still they are less productive than professional software developers. A study showed that professionals spend only a third of the total time non-driving (Plonka *et al.*, 2011).

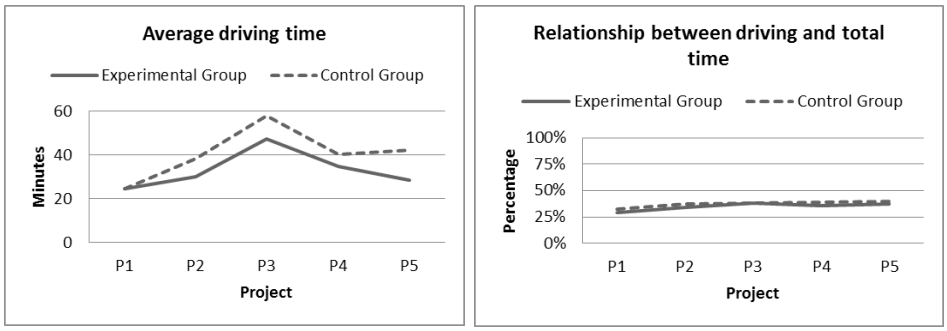

Fig. 7. Comparing driving time and its relation to total time between control and experimental condition.

4.5. *Evaluating Test Results*

(Q4): *Did the use of the system have an effect on students' knowledge level?*

(Q5): *Does structured collaboration lead to better learning outcomes compared to free collaboration?*

In order to study research questions (Q4) and (Q5) students' scores on the pre- and post-test were analyzed. A paired samples test for each condition was conducted in order to investigate if the use of the system did have an effect on students' knowledge level (Q4). All tests of the evaluation phase were run at 95% confidence level. The test revealed a statistically significant difference between the test scores of the control group ($t(22) = -5.401$, $p = 0.0005$) as well as between the scores of the experimental group ($t(23) = -5.816$, $p = 0.0005$). Both groups performed significantly better in the post-test which suggests an improvement in their knowledge level. However, it was expected to obtain such a result since the use of the system in combination with the provided learning material "forced" students to study Java and to improve their programming skills.

Comparing the pre-test scores of both conditions no statistically significant difference was found ($p = 0.637$) which suggests that both groups started with a similar knowledge level. Regarding research question (Q5), the control group ($M = 6.78$, $SD = 1.41$) achieved comparable scores to the experimental group ($M = 6.16$, $SD = 2.09$). A Mann-Whitney test between post-test scores showed that structured collaboration did not have an impact on students' performance ($U = 231.5$, $p = 0.343$) since both conditions performed equally well (Q5).

4.6. *Evaluating Project Scores*

(Q6): *Do students who are guided by structured collaboration achieve better assignment scores than students who are allowed to collaborate freely?*

The five programming assignments submitted by the students were assessed, and for each project an independent samples test was conducted in order to examine if the two groups had a difference in project scores. These tests investigate research question (Q6), namely if students who are guided by structured collaboration achieve better assignment scores than students who are allowed to collaborate freely.

It was observed that students performed very well in all projects with comparable project scores and software quality. However, a Mann-Whitney test was run for each individual project. In the first four projects no significant difference between the grades ($p > 0.05$) was found, but in the fifth project the test revealed that students of the control group performed better than the experimental condition ($U = 30.5$, $p = 0.015$). A closer look at the scores of the experimental condition revealed that the performance of two groups declined significantly in the last project and affected in that way the overall performance.

4.7. *Student Feedback*

(Q7) *What are students' perceptions of the system and the application of DPP?*

In this section the results of the evaluation questionnaires are presented, which were completed by the students during the last meeting. The aim of this part of the study was to obtain students' feedback on the system and the application of DPP. The following report summarizes the results in four subsections representing four different areas of interest.

4.7.1. *Feedback on Usability Issues*

Students were asked to evaluate system's usability answering five-point Likert (Likert, 1932) type questions (Table 2). As a whole, students found the system easy to use ($M = 3.89$, $SD = 0.60$). A significant difference between control and experimental group can be found in requesting and accepting role switches, where the experimental group rated with a lower grade the role switching mechanism, although most alternations were initiated

Table 2

Students' level of agreement on ease of use (Likert scale: 1 (strongly disagree) – 5 (strongly agree))

	Control Group	Experimental Group	Total
Installation	M = 4.65, SD = 0.49	M = 4.42, SD = 0.50	M = 4.53, SD = 0.50
Connection to server	M = 4.26, SD = 0.75	M = 4.04, SD = 0.81	M = 4.15, SD = 0.78
Session creation	M = 3.91, SD = 0.95	M = 3.75, SD = 0.74	M = 3.83, SD = 0.84
Role switches	M = 4.35, SD = 0.65	M = 3.33, SD = 1.01	M = 3.83, SD = 0.99
Chat	M = 4.22, SD = 1.09	M = 3.96, SD = 1.04	M = 4.09, SD = 1.06
Navigation in programming tasks	M = 3.96, SD = 0.98	M = 3.83, SD = 1.17	M = 3.89, SD = 1.07
Session close	M = 4.09, SD = 0.90	M = 4.08, SD = 0.72	M = 4.09, SD = 0.80
System as a whole	M = 4.00, SD = 0.60	M = 3.79, SD = 0.59	M = 3.89, SD = 0.60

by the system. In total, students agreed that they learned to use the system very quickly (M = 4.17, SD = 0.92), and that the interface was nicely designed (M = 3.60, SD = 0.74). At last, students didn't indicate a high server response time (M = 2.92, SD = 1.02).

4.7.2. *Feedback on System's Features*

In this section the students rated the usefulness of the embedded plugin features which aimed to facilitate DPP over Eclipse (Table 3). Apparently, collaboration scripts appeared quite beneficial for students since they provided a positive feedback on guided problem solving (M = 4.18, SD = 1.05) and the embedded assignment description (M = 4.00, SD = 0.83). They found less useful the chat functionality and hints (M = 3.47, SD = 1.30 and M = 3.33, SD = 1.13 respectively), a fact also confirmed by the findings of interaction analysis. The display of participation rates in the experimental group had a moderate effect on students (M = 2.83, SD = 1.03) and proved not a particularly useful feature.

4.7.3. *Feedback on Distributed Pair Programming and Collaboration*

Students' comments indicated a positive attitude towards DPP and collaboration. As an overall experience DPP was rated with an average score of 4.40 (SD = 0.58) on a scale of 1 (very poor) to 5 (very good). Learning computer programming is generally considered as a difficult procedure, and 66% of the students stated that they had faced many problems in the past while solving programming assignments. The majority of them (94%) believe that DPP can help students to overcome such difficulties. They agreed that the system is also suitable for novice programmers (M = 4.00, SD = 0.75) and confirmed to a large extent the main benefits of DPP (Table 4). Students acknowledged that a major advantage of DPP is the ability to collaborate remotely without wasting time. They indicated that co-located PP causes scheduling conflicts and sometimes can lead to distraction. The vast majority did not report any problems with the partner when asked to evaluate aspects like lack of knowledge, inconsistency and domination.

Table 3

Evaluation of system's capabilities (Likert scale: 1 (not useful) – 5 (very useful))

	Control Group	Experimental Group	Total
Saving and resuming a DPP session	M = 4.44, SD = 0.73	M = 4.04, SD = 0.93	M = 4.24, SD = 0.85
Guided problem solving	M = 4.57, SD = 0.51	M = 3.77, SD = 1.31	M = 4.18, SD = 1.05
Providing assignment description in workspace	M = 3.96, SD = 0.88	M = 4.04, SD = 0.81	M = 4.00, SD = 0.83
Concurrent file saving	M = 4.04, SD = 0.93	M = 3.75, SD = 1.23	M = 3.90, SD = 1.09
Synchronic program execution	M = 3.74, SD = 0.92	M = 3.71, SD = 0.75	M = 3.72, SD = 0.83
Remote code highlighting	M = 3.87, SD = 0.76	M = 3.75, SD = 0.74	M = 3.81, SD = 0.74
Automatic project creation	M = 3.78, SD = 0.85	M = 3.75, SD = 1.15	M = 3.77, SD = 1.01
Chat	M = 3.74, SD = 1.42	M = 3.21, SD = 1.14	M = 3.47, SD = 1.30
Hints	M = 3.22, SD = 1.17	M = 3.46, SD = 1.10	M = 3.33, SD = 1.13
Display of participation rates	–	M = 2.83, SD = 1.03	M = 2.83, SD = 1.03

Table 4
Students' level of agreement on DPP benefits (Likert scale: 1 (strongly disagree) – 5 (strongly agree))

	Control Group	Experimental Group	Total
Assignments are completed in a shorter time	M = 3.65, SD = 0.98	M = 3.21, SD = 0.98	M = 3.43, SD = 0.99
Students share knowledge and problem solving skills	M = 4.52, SD = 0.51	M = 4.00, SD = 0.93	M = 4.26, SD = 0.79
Errors in program code can be found earlier	M = 4.52, SD = 0.73	M = 4.38, SD = 0.71	M = 4.45, SD = 0.72
DPP facilitates learning programming	M = 4.13, SD = 0.55	M = 3.67, SD = 0.82	M = 3.89, SD = 0.73
Students are more confident in assignment solutions	M = 4.13, SD = 0.76	M = 4.04, SD = 0.91	M = 4.09, SD = 0.83
Learning to program is more enjoyable	M = 4.39, SD = 0.78	M = 3.92, SD = 1.06	M = 4.15, SD = 0.96
Students are able to solve more problems on their own	M = 4.13, SD = 0.82	M = 4.04, SD = 0.91	M = 4.09, SD = 0.86

4.7.4. *Suggestions and Limitations*

The evaluation questionnaire contained some open-ended questions in order to let students express their opinions about the system and to make suggestions for improvement. The main focus was given on negative experiences in order to detect system's limitations and bugs. Most students reported server crashes which caused delays and a rollback in the development process. Students of the experimental group felt that role assignments weren't always fair, because tasks had a different level of difficulty. The remaining reported problems were independent from each other, and we are looking forward to detect the circumstances under which such problems occurred.

Students' suggestions indicated an improvement in communication. The embedded chat should contain some additional features, like chat rooms in order to exchange messages with other groups. Some suggested incorporating audio or video communication because typing is more time consuming. Other comments indicated some improvements in the workspace. For example, some students would find useful to know the total time they spend in the driver role. Finally, although the majority stated that solving consecutive tasks facilitated the problem-solving process, they suggested to make available a preview of the entire assignment description at any time.

5. Discussion

In this study, the adoption of collaboration scripts aimed to structure collaboration during DPP and to facilitate role distribution among team members. For this purpose, an existing Eclipse plugin was redesigned, which keeps track of students' activities. SCEPPSys not only supports the standard requirements of DPP, it also introduces an adaptive floor control mechanism which grants floor control based on students' knowledge level. The definition of collaboration scripts inside the system allows teachers

to create a framework for adaptive DPP sessions depending on users' collaboration history. At the same time, the stored information gives an insight into students' interactions and individual contributions are estimated. This feature is particularly in group projects very useful in order to assess each member's effort. Situations like disengagement on the part of one student can be detected, which was also the case in one group of the control condition.

Enhancing DPP with the use of collaboration scripts created also a new approach to solve programming assignments. Instead of providing to students a general description of a programming project, they were asked to solve smaller programming tasks. Each task was part of a logical sequence of problem solving steps which led to the solution of a complete programming assignment. Students indicated that this way of programming facilitated the problem-solving process in contrast to traditional programming assignments. Furthermore, it seems that this approach helped students to distribute workload since the majority of the pair members in the control group tried to solve an equal number of tasks. Nevertheless, further research is needed in order to study students' learning outcomes under different problem solving approaches. In our future work we plan to investigate the impact of various task distribution strategies on students' performance and learning behavior. For instance, mandatory role assignments or successive role alternations are two different turn taking approaches which will complement the evaluation phase of our system.

The evaluation results also revealed the capability to achieve symmetry in skill acquisition. Students who were assigned tasks depending on learning goals showed a statistically significant difference in the level of balanced knowledge acquisition and each student met approximately an equal number of learning goals. The drawback of this strategy was that students didn't gain an understanding of the purpose of the role switches. A future direction for improvement would be to provide to students a feedback of their interactions and information about the programming skills they have acquired so far. This feature would support the justification of system-driven role assignments and would help students to self-regulate role switches.

This study considered the participation of each student in the program code in order to assess his contribution. A limitation of the system is that it captures only the work of the driver and not the contributions of the navigator. Content analysis is not supported by the system and also hard to achieve when students use external communication channels. Lastly, the difficulty level of each programming task was not recorded by the system which constitutes a different approach in task distribution.

6. Conclusion

In this article a new DPP system and the results of its first evaluation were presented. In addition, this study introduced collaboration scripts in the DPP methodology and initial findings were reported. The evaluation study of SCEPPSys showed that when DPP teams are guided by collaboration scripts students achieve contribution rates within satisfactory limits. In addition, when collaboration is coordinated by the system, more

balanced knowledge distribution is gained, and role alternations occur more often during DPP sessions. These benefits do not come at the cost of productivity. On the contrary, system's interventions in turn taking resulted in shorter DPP sessions. This study also demonstrates the applicability of collaboration scripts in a wide range of collaborative learning areas like the DPP model. Furthermore, the system gained positive feedback from students and some useful and encouraging remarks were reported. The alternative problem solving approach appeared to students quite beneficial, and the DPP methodology has been proved as an effective way to solve programming assignments in computer science courses. The application of DPP by novice programmers was conducted without any complications even though most students were inexperienced in this practice. The findings of the current evaluation study gave us some ideas for improvement and directions for future research. Our next step towards this direction includes minor corrections regarding the user interface and experimentation with various group formation strategies. The software will be publicly available as soon as the development and evaluation process is completed.

References

Avouris, N., Margaritis, M., Komis, V. (2004). Modelling interaction during small-group synchronous problem-solving activities: the Synergo approach. In: *Proceedings of ITS 2004 workshop on designing computational models of collaborative learning interaction*, 13–18.

Baheti, P., Williams, L., Stotts, D., Smith, J.M. (2002). Distributed pair programming: empirical studies and supporting environments. *Technical Report TR02-010*. Department of Computer Science. University of North Carolina at Chapel Hill.

Barker, L.J., (2005). When do group projects widen the student experience gap?. In: *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, 276–280. DOI: 10.1145/1067445.1067521

Boyer, K.E., Dwight, A.A., Fondren, R.T., Vouk, M.a., Lester, J.C. (2008). A development environment for distributed synchronous collaborative programming. *ACM SIGCSE Bulletin*, 40(3), 158–162. DOI:10.1145/1597849.1384315

Bravo, C., Duque, R., Gallardo, J. (2013). A groupware system to support collaborative programming: design and experiences. *Journal of Systems and Software*, 86(7), 1759–1771. DOI:10.1016/j.jss.2012.08.039

Chaparro, E., Yuksel, A., Romero, P., Bryant, S. (2005). Factors affecting the perceived effectiveness of pair programming in higher education. In: *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group*, 5–18.

Dillenbourg, P., Fischer, F. (2007). Computer-supported collaborative learning: the Basics, *Zeitschrift für Berufs- und Wirtschaftspädagogik*. 21, 111–130.

Diziol, D., Rummel, N., Spada, H., McLaren, B.M. (2007). Promoting learning in mathematics: script support for collaborative problem solving with the Cognitive Tutor Algebra. In: *The Proceedings of the Conference on Computer-Supported Collaborative Learning.*

Duque, R., Bravo, C., Ortega, M. (2011). A model-based framework to automate the analysis of users' activity in collaborative systems. *Journal of Network and Computer Applications*, 34(4), 1200–1209. DOI: 10.1016/j.jnca.2011.01.005

Faja, S. (2011). Pair Programming as a team based learning activity: a review of research. *Issues in Information Systems*, XII(2), 207–216.

Favela, J., Natsu, H., Pérez, C., Robles, O., Morán, A.L., Romero, R., … Decouchant, D. (2004). Empirical evaluation of collaborative support for distributed pair programming. In: *Groupware: Design, Implementation, and Use*, 215–222. DOI: 10.1007/978-3-540-30112-7_18

Fay, M.P., Proschan, M.A. (2010). Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics surveys*, 4, 1.

Gardner, W. (2003). Assessing individual contributions to group software projects. In: *8th Western Canadian Conference on Computing Education (WCCCE '03)*. Courtenay, BC, Canada, 33–50.

Granville, K., Hickey, T.J. (2005). The design, implementation, and application of the grewpEdit tool. In: *Proceedings of the 2005 Conference on Diversity in Computing*, 14–16. DOI:10.1145/1095242.1095249

Hanks, B. (2008). Empirical evaluation of distributed pair programming. *International Journal of Human-Computer Studies*, 66(7), 530–544. DOI:10.1016/j.ijhcs.2007.10.003

Hernández-Leo, D., Villasclaras-Fernández, E.D., Asensio-Pérez, J.I., Dimitriadis, Y., Jorrín-Abellán, I.M., Ruiz-Requies, I., Rubia-Avi, B. (2006). COLLAGE: A collaborative Learning Design editor based on patterns. *Journal of Educational Technology & Society*, 9(1).

Ho, C., 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*, 73–77. DOI:10.1145/1066129.1066144

Janssen, J., Erkens, G., Kanselaar, G., Jaspers, J. (2007). Visualization of participation : does it contribute to successful computer-supported collaborative learning ?. *Computers & Education*, 49(4), 1037–1065. DOI:10.1016/j.compedu.2006.01.004

Jurado, F., Molina, A.I., Redondo, M.A., Ortega, M. (2013). Cole-programming: shaping collaborative learning support in eclipse. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 8(4), 153–162. DOI: 10.1109/RITA.2013.2284953

Jurado, F., Molina, A.I., Redondo, M.A., Ortega, M., Giemza, A., Bollen, L., *et al*. (2009). Learning to program with COALA, a distributed computer assisted environment. *Journal of Universal Computer Science*, 15(7), 1472–1485.

Kobbe, L., Weinberger, A., Dillenbourg, P., Harrer, A., Hämäläinen, R., Häkkinen, P., Fischer, F. (2007). Specifying computer-supported collaboration scripts. *International Journal of Computer-Supported Collaborative Learning*, 2(2–3), 211–224. DOI:10.1007/s11412-007-9014-4

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Milligan, C.D., Beauvoir, P., Sharples, P. (2005). The Reload learning design tools. *Journal of Interactive Media in Education*, 2005(1).

Plonka, L., Segal, J., Sharp, H., van der Linden, J. (2011). Collaboration in Pair Programming: driving and switching. In: *Agile Processes in Software Engineering and Extreme Programming*. 43–59. DOI: 10.1007/978-3-642-20677-1_4

Plonka, L., Sharp, H., van der Linden, J. (2012). Disengagement in pair programming: does it matter? In: *34th International Conference on Software Engineering (ICSE)*. 496–506. DOI:10.1109/ICSE.2012.6227166. DOI: 10.1109/ICSE.2012.6227166

Preston, D. (2005). Pair programming as a model of collaborative learning: a review of the research. *Journal of Computing Sciences in colleges*, 20(4), 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)*. 48–55. DOI:10.1145/1833310.1833319

Schuemmer, T., Lukosch, S. (2009). Understanding tools and practices for distributed pair programming. *Journal of Universal Computer Science,* 15(16), 3101–3125. DOI: 10.3217/jucs-015-16-3101

Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., Jackson, A. (2003). Virtual teaming: experiments and experiences with distributed pair programming. In: *Extreme Programming and Agile Methods-XP/ Agile Universe 2003*. 129–141. DOI:10.1007/978-3-540-45122-8_15

Strijbos, J.W., Weinberger, A. (2010). Emerging and scripted roles in computer-supported collaborative learning. *Computers in Human Behavior*, 26(4), 491–494.

Tsompanoudi, D., Satratzemi, M. (2014). A Web-based authoring tool for scripting distributed pair programming. In: *14th IEEE International Conference on Advanced Learning Technologies*. 259–263.

Tsompanoudi, D., Satratzemi, M., Xinogalos, S. (2013). Exploring the effects of collaboration scripts embedded in a distributed pair programming system. In: *Proceedings of the 18th ACM conference on Innovation and technology in computer science education.* 225–230.

Tsovaltzi, D., Rummel, N., McLaren, B.M., Pinkwart, N., Scheuer, O., Harrer, A., Braun, I. (2010). Extending a virtual chemistry laboratory with a collaboration script to promote conceptual learning. *International Journal of Technology Enhanced Learning*, 2(1), 91–110.

Williams, L. (2007). Lessons learned from seven years of pair programming at North Carolina State University. *ACM SIGCSE Bulletin*, 39(4), 79–83. DOI:10.1145/1345375.1345420

Williams, L., McCrickard, D.S., Layman, L., Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In: *Proceedings of the Agile 2008* (AGILE '08). 445–452. DOI:10.1109/ Agile.2008.12

D. Tsompanoudi is a PhD candidate in the Department of Applied Informatics at the University of Macedonia (Greece). She received her B.Sc. degree in Informatics from the Aristotle University of Thessaloniki in 2003 and her M.Sc. degree in Information Systems from the University of Macedonia in 2006. Since 2005 she works as a teacher of Informatics in Secondary Education. Her main research interests include computer-supported collaborative learning systems and the study of distributed pair programming in education.

M. Satratzemi received the BS degree from Mathematics Department, Aristotle University of Thessaloniki (AUTh), and the PhD degree in Informatics from the Department of Applied Informatics, University of Macedonia. She is a professor at the Department of Applied Informatics. Her current main research interests lie in the area of Educational Programming Environments and Techniques, Didactics of Informatics, Adaptive and Intelligent systems, Collaborative Learning Systems, Game-based Learning. She was Conference co-chair of the 8th ITiCSE'2003.

S. Xinogalos is an assistant professor of Programming Environments and Techniques at the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece (e-mail: stelios@uom.edu.gr). He is author or co-author of more than 50 research papers on programming environments and techniques, object-oriented design and programming, educational environments and games for programming, as well as didactics of programming published in international journals, books and conference proceedings. He has also designed and implemented two educational programming environments.

Paskirstytasis porinis programavimas naudojantis bendradarbiavimo scenarijais: švietimo sistema ir pradiniai rezultatai

Despina TSOMPANOUDI, Maya SATRATZEMI, Stelios XINOGALOS

Nuo tada, kai porinis programavimas buvo aprašytas literatūroje kaip efektyvus metodas mokyti kompiuterinio programavimo, sukurta labai daug taikomųjų programų, kurios padėtų dirbti per atstumą. Šiandieninės sistemos tarnauja asmeniniams, profesiniams ir švietimo tikslams ir leidžia komandoms kartu dirbti su tuo pačiu programavimo projektu. Dabartiniai tyrimai daugiausiai dėmesio skiria paskirstytojo porinio programavimo sistemoms, kurios tinka mokyti studentus kompiuterinio programavimo. Visų laisvai prieinamų sistemų sisteminė apžvalga atskleidė, kad nėra veiksmingos bendradarbiavimo pagalbos studentams. Pagrindiniai porinio programavimo trūkumai – nėra tolygaus darbo krūvio pasiskirstymo ir retas vaidmenų įsijungimas – negali būti įveikti prieinamose sistemose. Siekiant sukurti patobulintą paskirstytojo porinio programavimo sistemos versiją, buvo ištirtos sėkmingos mokymo strategijos panašiose mokymosi bendradarbiaujant sistemose. Buvo sukurta nauja sistema, kuri leidžia studentams pabandyti porinį programavimą naudojantis bendradarbiavimo scenarijais. Straipsnyje pateikiamos sistemos savybės ir pagrindiniai konceptai bei pirmojo vertinimo rezultatai.