Design and Evaluation of a Web-based Distributed Pair Programming Tool for Novice Programmers

José Colin, Sébastien Hoarau, Christophe Declercq, Julien Broisin

Design and Evaluation of a Web-based Distributed Pair Programming Tool for Novice Programmers

José Colin
jose.colin@irit.fr
IRIT, Université Toulouse 3 Paul Sabatier
Toulouse, France

Sébastien Hoarau
Christophe Declercq
seb.hoarau@univ-reunion.fr
christophe.declercq@univ-reunion.fr
Laboratoire d'Informatique et de Mathématiques, Université de la Réunion
Saint-Denis, France

Julien Broisin
julien.broisin@irit.fr
IRIT, Université Toulouse 3 Paul Sabatier
Toulouse, France

ABSTRACT

Research on pair programming (PP) in education have shown a number of positive outcomes for learners, and especially novice programmers, such as enhanced learning, greater confidence in work quality, higher problem solving skills or enhanced interaction skills, and promotes collaborative learning. Due to these diverse advantages, pair programming in education currently follows a growing curve. Also, blended learning approaches are becoming more and more popular in education, including when learners have to learn programming. As a consequence, distributed pair programming (DPP) can be considered as a good solution to support pair programming in hybrid learning scenarios. A large number of tools from both the research community and the major integrated development environment (IDE) editors tried to study and implement DPP in their tools. However, our review of literature shows that none of them meet the requirements for delivering effective pair programming activities to novice programmers in blended learning scenarios. Based on these findings, the paper introduces a new DPP application especially designed for novice programmers. It integrates, based on some requirements identified from previous research, several features dedicated to DPP as well as other capabilities supporting extensive data collection and learning analytics. The tool has been experimented in authentic learning settings in higher education with 82 students, both in PP and DPP conditions. The experiment showed no evidence of a difference between PP and DPP on the students' perceived usability of the application, as well as on the quality of their productions.

CCS CONCEPTS

• **Software and its engineering** → **Pair programming**; • **Social and professional topics** → *CS1.*

KEYWORDS

Computer Science Education; Distributed Pair Programming; Tool Design and Evaluation; Novice Programmers

ACM Reference Format:
José Colin, Sébastien Hoarau, Christophe Declercq, and Julien Broisin. 2024. Design and Evaluation of a Web-based Distributed Pair Programming Tool for Novice Programmers. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653571

1 INTRODUCTION

Pair programming (PP) is known for its many benefits when used in education. Prior works have shown multiple positive impacts of PP such as enhanced learning, greater confidence in work quality, higher problem solving and interaction skills [9], as well as greater student's perceived satisfaction and enjoyment [21]. PP is frequently studied in higher education, but research conducted on high school students [16] and in introductory computer science courses at the university level [31] have demonstrated its effectiveness when employed with novice programmers.

Blended learning approaches are increasingly being employed by educational institutions and teachers at various levels of education. In those learning settings, distributed pair programming (DPP) enables students to remotely engage in pair programming activities. The effectiveness of DPP in education compared to PP and solo programming seems to be very promising concerning code quality, code comprehension and academic performance, as research has shown that no difference were observed between those programming practices [22]. However, due to the multiple challenges raised by the online component of blended learning [19], current tools dedicated to distributed pair programming suffer from several weaknesses such as lack of features to change roles between students, poor support of activities management by teachers, or low data collection preventing advanced features offered by learning analytics.

In this paper we introduce a new tool for DPP and report on the results of an experiment conducted in higher education with first-year students. We explore the tool's capacity to support DPP over regular PP by comparing student's perceived usability and the quality of their productions in these two contexts. After introducing the related work in the field of DPP, we present the tool that was designed. Then we describe the experimental settings, the data

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07.
https://doi.org/10.1145/3649217.3653571

collected and the analysis aimed at evaluating the tool. The results are presented and discussed, and the limitations of this work are highlighted. Finally, we present our conclusions and outline the main short-term perspectives focused on the design of data-driven scaffolding tools for students and teachers.

2 RELATED WORK

The objective of this section is to identify the requirements for a distributed pair programming application dedicated to novice programmers, according to previous findings of the literature. Existing tools are then evaluated regarding these requirements in order to highlight the main issues that still need to be addressed.

2.1 Distributed pair programming requirements

According to the definition by Schenk et al. [24], distributed pair programming takes place in a virtual environment where a shared editor is provided to the users so that all modifications of the source code made by a participant are transferred to the other participant's screen. The virtual environment also ensures that only one user can modify the source code at any time. As in traditional pair programming, this user is called the Driver and its partner is called the Navigator. The role of the Driver is to actively work on the programming task by editing the source code, while the Navigator observes and assists the Driver by providing information, suggesting improvements and identifying problematic parts of the code.

Requirements for DPP tools have been discussed in prior works. Winkler et al. [30] developed a systematic evaluation process for DPP tools and listed different properties for such tools. In particular, they identified several requirements with high priority. *Workspace awareness* refers to the ability to know who is participating in a DPP session, to understand the role of everyone, and to be aware of the interactions within the shared space. *Floor control* relates to the restriction on who can edit the source code. It should allow modifications from the Driver only. *Role changes* provides users with a mechanism to change their role during a DPP session. *Communication tools* offer communication channels like voice channels or textual chat between the participants of a DPP session. Finally, *Gesturing features* help users in visualizing others' actions on the shared workspace. For example, the shared pointer feature allows the cursor of one user to be visible by other participants. The shared highlighting feature, which makes it possible for all users to see others' text selection, is another example of gesturing features.

In addition to these features, Schummer et al. [25] emphasized another capability that DPP tools should implement. The *Spectator mode* feature allows users to view and follow the interactions in an ongoing collaborative session without influencing the participants of that session. This feature is very important in a (distant) learning context, for instance to allow a teacher to follow what is going on during the different DPP sessions of her students. Besides, the literature also shows that tools designed for novice programmers require additional essential educational capabilities.

2.2 Educational requirements specific to novice programmers

For learning how to program, it is common not to use the same programming software as in the professional world, but rather learning environments adapted to the needs of the learners. Our review of literature on previous research aiming at supporting novice programmers led us to identify a number of main requirements. For a long time, there have been efforts to provide beginners with *Simple interfaces* in the form of basic code editors [3, 17], so that learners can focus on computational concepts [11] instead of struggling with complex graphic user interfaces. Also, education tools always provide various mechanisms to manage the *Pedagogical activities* they are designed for and to assign students to these activities [12]. When it comes to programming learning, programming software propose very rich debugging support but they still require too much efforts for beginners. The research community has thus widely studied how to enhance *Testing and debugging* performance [4] of novice programmers, as well as understanding of compiler error messages [6]. Another important supporting capability of computer education tools is *Tutoring*, whether provided by a human or artificial intelligence [2, 14]. Finally, more and more educational tools stand on *Data collection* to enhance the support provided to learners. These systems collect data resulting from the interactions of users in the system to better understand their behaviors [18, 26], predict their performance [15], or provide them with the educational resources they need [10].

2.3 Existing systems for distributed pair programming

A wide range of tools and applications support the distributed pair programming activity. In this section we review the main initiatives before exposing a synthesis of these tools regarding the set of criteria identified in the previous sections.

2.3.1 Integrated Development Environments. Most Integrated Development Environments (IDEs) can provide DPP natively as a feature, or by the use of extensions including Live Share for Visual Studio and Visual Studio Code, Code with me for Intellij, or Code Together for Eclipse. These tools offer extensive features regarding awareness, gesturing and communication. Sometimes, they also implement floor control. For instance, Live Share integrates a read-only mode that prevents other users from editing the code. However, none of these extensions enforces a strict separation of roles between the Driver and the Navigator, like it is usually done in traditional pair programming.

Also, IDEs are often considered unsuitable for novice programmers due to their complexity [13]. A study by Rigby et al. [20] tends to show that the use of an IDE creates more frustration and less satisfaction for novice programmers compared to a tool designed specifically for them.

2.3.2 Web-based editors. Distributed pair programming is also implemented in web-based tools specifically designed to support collaborative work such as Replit, Codeshare or Codefile. These tools offer a less complex environment than most IDEs and thus implement a limited number of features, which is an advantage for novice programmers. However, just like IDEs, they do not enforce a strict separation of pair programming roles. Also, these tools do not collect, or do not provide access to the interaction data between the users and the system. It is thus nearly impossible for teachers to analyze learners' behavior and to bring them the support

Table 1: Evaluation of existing tools against educational and DPP requirements

	Simple interfaces	Pedagogical activities	Testing and debugging	Tutoring	Data collection	Workspace awareness	Floor control	Role changes	Communication tools	Gesturing features	Spectator mode
Live Share (Visual studio and VSCode)	+	-	++	-	-	++	+	-	++	++	++
Code With Me(IntelliJ)	-	-	++	-	-	++	-	-	++	++	++
Code Together (Eclipse)	-	-	++	-	-	++	-	-	++	++	?
Replit	++	-	++	-	-	++	-	-	+	++	-
Codeshare	++	-	-	-	-	+	-	-	-	+	-
Codefile	++	-	-	-	-	++	-	-	-	++	-
Xpairtise	-	-	++	+	?	++	++	++	+	++	++
XecliP	-	-	++	-	?	++	++	++	+	++	-
SCEPPSys	-	++	++	++	+	++	++	++	+	++	++

Legend: requirement fully supported (++), partially supported (+), not supported (-), unknown (?)

they need, and for researchers to understand how programming learning occurs. Also, none of these tools are specifically designed for education, so they do not provide pedagogical capabilities.

2.3.3 Research initiatives. Xpairtise [25] and XecliP [28, 33] are two Eclipse plugins designed by the research community. Both are very complete regarding the DPP requirements identified in section 2.1, but they lack pedagogical and tutoring capabilities as they have not been especially designed for education.

SCEPPSys [27] extends XecliP to support pair programming in classes, and is thus much more complete in terms of educational features. In particular, an administration panel allows teachers to manage learning activities and pair programming groups. SCEPPSys also collects a wide range of data to assist the evaluation of students, like the duration a student spends on the Driver and Navigator roles, the amount of code produced or the usage of communication tools. However, the data collection regarding code execution and testing by students seems limited. Also, like Xpairtise and Xeclip, this tool extends the Eclipse IDE, which is not suitable for beginners.

2.4 Synthesis

Table 1 summarizes the evaluation of the above-mentioned tools regarding the set of DPP and educational requirements identified in Sections 2.1 and 2.2. It highlights the main existing gaps that drove the design of our application.

3 OUR APPLICATION FOR DISTRIBUTED PAIR PROGRAMMING

This section introduces the architecture of our tool for learning Python, and focuses on the features implementing the DPP and educational requirements.

3.1 Global overview and technologies

The application we developed is based on web technologies to allow easy access from any web browser, with the objective of encouraging its use by teachers and students. The main motivating factor in its design was the pursuit of **simplicity**.

The overall client-server architecture of our application and the communications between its various components are illustrated in Figure 1. The backend side comprises several NodeJS APIs mainly dedicated to the educational features, as well as a ShareDB-based WebSocket server providing duplex communications between the front and the backend to implement the synchronization of the shared editor between users and many awareness/gesturing features. The backend side also integrates a Learning Record Store (LRS) compliant with the xAPI standard to store data resulting from the interactions between the users and the application. This component has been developed with NodeJS and stands on MongoDB for the database, but some works are in progress to use the TraxLRS system instead of this home-made store.

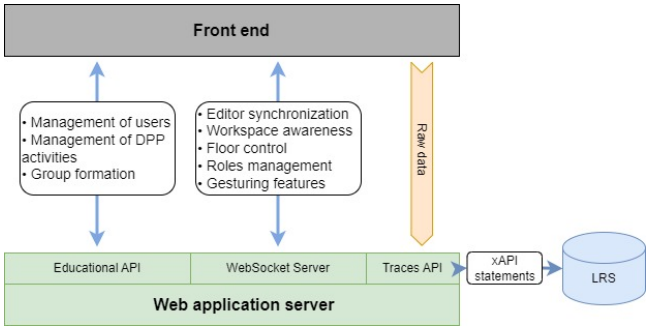

Figure 1: Overall architecture of our tool dedicated to distributed pair programming for learning Python.

We implemented the frontend using ReactJS. Also, it is important to note that the frontend embeds Pyodide as a Python interpreter running inside the web browser, thus bringing scalability of the application by minimizing the load of the server.

3.2 Distributed pair programming features

The application user interface implementing the DPP requirements is illustrated in Figure 2. It comprises 5 main components.

The shared editor (Component 1 in Figure 2) contributes to the **workspace awareness**, as it is synchronized between all members of the same DPP session to allow everyone to see the same source code. It also implements two important **gesturing features**: the shared pointer and the shared text highlighting use colors to identify the different users. Finally, this component implements **floor control**, as only the user with the Driver role can edit the source code; the other participants of the session have read-only access to the code. The objective of the code written by the students in the shared editor is to solve the problem exposed in Component 3.

Component 4 enhances the **workspace awareness** by showing the users currently connected to the DPP session, and associates them with a color. That color is then used by other visual elements associated with this user; for instance, the color associated to a user matches with the color of her cursor in the shared editor. Also, Component 4 shows the user currently in the Driver role through a specific icon (i.e., a pencil). Finally, when a teacher creates an activity (see the next section), she is automatically granted permission to join the DPP session of all pairs of students registered for that activity. When the teacher connects to a specific DPP session with students, her name also appears in Component 4. This ensures the integration of the **spectator mode** into our application and

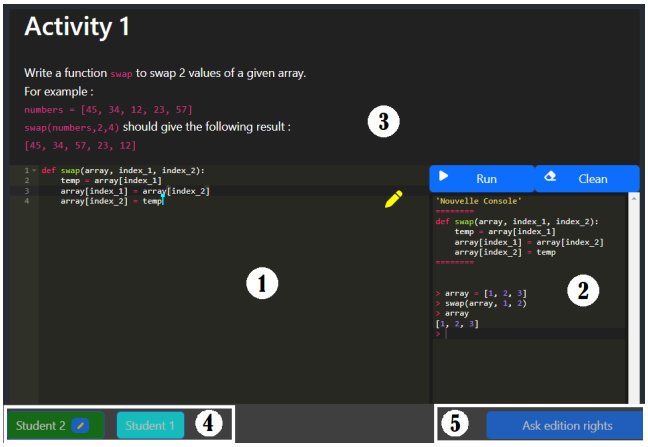

Figure 2: The application user interface dedicated to the DPP requirements.

contributes to the **tutoring** requirement, as teachers can be aware of the difficulties students face and provide the appropriate support.

Component 5 is dedicated to **role changes**, and allows users in the Navigator role to request a switch towards the Driver role. The current Driver then receives a notification of the request, and has the opportunity to accept or reject it. According to the request's response, the role switch occurs or not.

Finally, this user interface implements the **code testing** educational requirement through the Python console of Component 2. Each user has its own console which is not shared to the other participants of the DPP session. It allows participants of a session to execute the whole Python code written in the shared editor and run any Python commands to test the code. The other educational requirements are offered through other artifacts and user interfaces described below.

3.3 Educational features

The educational features are mainly implemented on the backend side of the application through two APIs.

The first API is dedicated to the management of **pedagogical activities**. It ensures the creation, modification and deployment of pair programming activities consisting of (i) a problem to be solved specified by the teacher, and (ii) a list of pairs of students. This API is available to teachers through a user interface illustrated in Figure 3. It shows the Markdown editor to specify the problem statement, and a form to set up the pairs of students. It is the responsibility of the teacher to choose how to form these pairs. Once the activity is created, the teacher can share a specific link to her students. When they access this link, the system starts a specific session of the activity for each pair of students, and assigns the Driver role to the first of the two students connecting to the activity.

The other API is responsible for **data collection**. It gathers data from both the client and the server sides, formats the data according to the xAPI standard [32], and stores the resulting statements into a the learning record store (see Figure 1). We adopted the xAPI format as it is widely used in the Technology Enhanced Learning community, thus allowing for rich analysis of the students' behavior

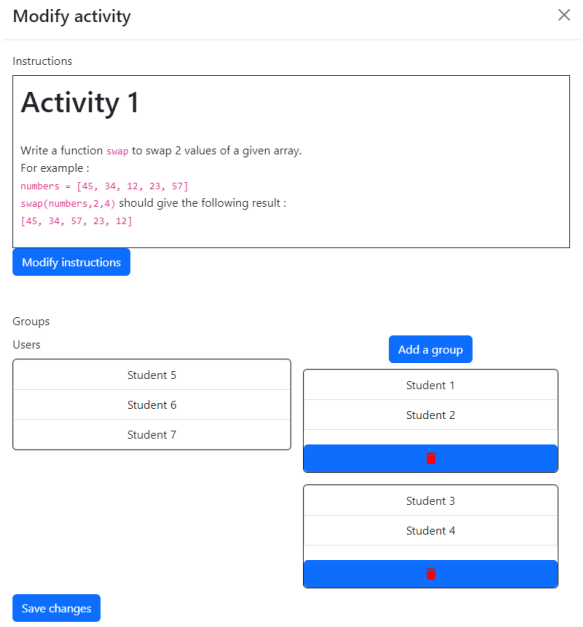

Figure 3: The teacher user interface dedicated to the creation of a pair programming activity.

combining both, actions performed within our application, as well as those performed within other education tools such as learning management systems. We collect seven types of xAPI statements: connection and disconnection to a session, role switch requests and responses, source code writing, code execution (including the result of the execution), and interactions with the Python console. All statements comprise, among other data, a timestamp and the user that performed the action. Therefore, by processing those statements, it is possible to model the whole sequence of interactions that occurred during a DPP session. The goal behind this data collection is to design dashboards based on indicators such as the time spent or the amount of code produced by students, but also to propose more complex features such as (intelligent) tutoring capabilities supporting both the teachers and the students in their respective tasks.

In summary, our application integrates features for each of the DPP and educational requirements identified in Section 2, excluding the *Communication tools* requirement. Indeed, for a first version of the application, we assumed that audio and video conference software such as Discord, Zoom or Teams were sufficient.

4 EXPERIMENT AND EVALUATION METHODS

Our DPP application has been experimented in authentic learning settings, with the objective of assessing the students' perceived usability on the one hand, and the quality of their productions in different conditions on the other hand.

4.1 Experimental setup

The experiment involved 82 first-year students from the Université de la Réunion, in the context of a beginner Python programming

course (CS1). It was distributed over two weeks, and each week comprised one hands-on lab session of one hour. In both weeks, students were asked to solve exercises about the concept of two-dimensional list; the exercises were different for the two learning sessions. Let us note that students were not trained on the application before the experiment; the teacher just briefly introduced the application the first week, and required the students to use it. As student were novices programmers, they had little or no previous experience with PP.

We proposed two experimental conditions to assess our application: the traditional pair programming and the distributed pair programming conditions. In the former condition, a single computer was used and the students were seated next to each other. In that experimental condition, a single access code was assigned to the pair of students so they could connect to the application. Instead, in the DPP condition, each student of a pair had access to a computer, and they were both connected to the same activity session. They were free to change roles between Driver and Navigator. In that experimental condition, students of a pair were also seated next to each other in the classroom so that they could talk. However, they could not see their partner's screen, hence reproducing a setting close to what two remote programmers can experience in DPP using a communication tool.

The first week involved 26 pairs of students (Group A) in the DPP condition and 15 pairs of students (Group B) in the PP condition. As we wanted that each pair of students experienced the two experimental conditions, students of Group A were assigned to the PP condition in the second week of the experiment, while students of Group B were assigned to the DPP condition. The pairs of students remained unchanged between the two weeks. As all participants were exposed to identical conditions, this experimental design enabled us to isolate the potential effects of other factors that characterize students and could potentially influence their performance. This was particularly important as we lacked prior information about the participants.

4.2 Data collection

Both the students' perceived usability of the application and the quality of their productions were evaluated on the basis of quantitative data.

To assess the students' perceived usability of the application, all students were *asked* to answer the System Usability Scale (SUS) questionnaire [7] after each hands-on lab session. Students thus gave their perceived usability of the application in both PP and DPP conditions. The SUS is recognized as a quick and reliable tool to assess the user's perceived usability of a system [8]. Also, previous studies have shown that it is a valid and reliable instrument to assess web applications [29]. This questionnaire comprises 10 statements, and the respondent gives her level of agreement on the basis of a 5-point Likert scale. The SUS score of each respondent is then calculated according to [7], and the final score corresponds to the average of all respondents' scores. It ranges from 0 to 100, where higher scores indicate better usability. To add to the general questions of the SUS questionnaire, we delivered to the students in the DPP condition a questionnaire related to the main features of the application. The questionnaire is available online at recherche.data.gouv.fr. It comprises four statements associated to a 5-point Likert scale agreement. The four statements respectively tackled the partner's cursor visualisation (S1), the Python console (S2), the distribution of the roles within the pair (S3), and the mechanism for switching roles (S4). We also delivered an open-ended question asking what other features students would like to be integrated in the application.

To assess the work achieved by the pairs of students, we collected the whole set of their productions. More precisely, for each pair of students and each week of the experiment, we collected the final version of their source code, that is the source code they left when disconnecting from the application. These productions were then evaluated on a scale from 0 to 5 by a single researcher in computer science, to avoid disparities in student grading.

Table 2: Mean, SD and median of the SUS score in both groups of students, for each week of the experiment.

		Mean	SD	Median
Group A (N = 37)	Week 1 (DPP)	67.97	14.99	67.5
	Week 2 (PP)	69.19	16.33	70
Group B (N = 18)	Week 1 (PP)	63.47	13.15	67.5
	Week 2 (DPP)	65	10.96	66.25

4.3 Data analysis

The mean, standard deviation and median of the SUS score were calculated for each group of students, so as to compare their perceived usability in the traditional pair programming condition versus the distributed pair programming condition. Also, to investigate whether or not there was an evidence of a statistical difference in terms of perceived usability between the two experimental conditions, we ran a paired t-test for each experimental group. To get more insights on the DPP features, we computed the mean, standard deviation and median of the Likert-scale questions delivered to students in the DPP condition.

To assess the quality of learners' productions, we compared the mean, standard deviation and median of the students' grades in the two experimental conditions. However, in contrast to the assessment of the perceived usability, our analysis were carried out independently for each week of the experiment because the activities were different over the two weeks. We considered that comparing the grades of the two groups on the same programming task ensures a fair comparison. For that reason, we used an independent t-test, to investigate whether or not there was a significant difference on the quality of students' productions between the 2 groups on each week.

5 RESULTS AND DISCUSSION

5.1 Perceived usability

The results of the SUS questionnaire are based only on students who completed the questionnaire each week, i.e., who experienced the two experimental conditions. We collected 37 responses from Group A, and 18 from Group B. Results appear in Table 2.

Regarding Group A, the mean scores of the SUS are very similar in both experimental conditions, indicating a slight increase in the

Table 3: Mean, SD and median of the responses to the questionnaire delivered only to the students in the DPP condition.

	Group A (N = 32)			Group B (N = 15)			Group A and B		
	Mean	SD	Median	Mean	SD	Median	Mean	SD	Median
S1	3.59	1.32	4.0	3.73	1.29	4.0	3.64	1.31	4.0
S2	3.78	1.11	4.0	3.47	1.15	4.0	3.64	1.13	4.0
S3	3.84	1.12	4.0	3.73	0.85	3.0	3.81	1.04	4.0
S4	3.63	1.36	4.0	4.07	0.85	4.0	3.77	1.24	4.0

PP condition (score = 69.19) compared to the DPP condition (score = 67.97). Regarding Group B, the scores are lower: 63.47 in the PP condition, and 65 in the DPP condition. According to [5], the application has been qualified as "OK" by both groups of students.

For the questionnaire administered exclusively to students in the DPP condition, we obtained 32 responses from Group A and 15 from Group B. The results given in Table 3 are aligned with those of the SUS questionnaire. Students of both groups homogeneously assessed the DPP features of the application, the results indicating that students tend to "agree" that each feature fulfill its objective. Among the 47 answers to the open-ended question, 6 students asked for the integration of a chat, which highlights the need for textual communication even when students can communicate orally.

Those results are encouraging for several reasons. First, the average SUS score for Group A, which comprises more students than Group B, is 68.58. It is near the "Good" threshold (i.e., 71 according to [5]), and just above the average SUS score for web applications (i.e., 68.05) according to previous research based on an empirical evaluation of 1180 existing SUS survey results [1]. Second, results show that in both groups, the mean SUS score increases between the first and the second week of the experiment. We assume that if the students had been trained before the experiment, the mean SUS scores would have reached higher values. This hypothesis is confirmed by 3 students who answered in the open-ended question of the second questionnaire that they would have liked some help to understand the interface and the console. Third, the results of the paired t-tests show no evidence of a difference in the SUS scores between both experimental conditions in Group A ($t = 0.67$, $p = 0.50$), as well as in Group B ($t = 0.46$, $p = 0.64$). These results suggest that using the tool in distributed pair programming condition has no negative impact regarding students' perceived usability compared to using it in traditional pair programming condition.

5.2 Quality of students' productions

Table 4 gives the results of the analyses of student pairs' productions. In both weeks, student pairs in the DPP condition achieved higher grades compared to those in the PP condition. This tendency is particularly noteworthy during the second week of the experiment. This is attributed to the increased difficulty of the programming task that week, where 10 pairs of students in Group A invested too much time thinking on the solution without success.

However, these results are tempered by the findings of the independent t-tests, which show no evidence of a difference in grades between the two experimental conditions in both week 1 ($t = 0.44$, $p = 0.66$) and week 2 ($t = 1.81$, $p = 0.08$). Nevertheless, they once again suggest that working in a DPP condition had no negative impact on the students' programming achievement, compared to the traditional PP condition.

Table 4: Mean, SD and median of the grades of the student pairs' productions, per week and for each experimental condition.

		Mean	SD	Median
Week 1	DPP (Group A, 26 pairs)	2.57	1.02	2.25
	PP (Group B, 15 pairs)	2.43	0.89	2.0
Week 2	DPP (Group B, 15 pairs)	2.23	1.70	2.0
	PP (Group A, 26 pairs)	1.29	1.48	0.5

6 LIMITATIONS

The main limitations of the presented work are related to the experiment. Concerning the DPP experimental condition, we aimed to replicate the experience of two learners engaged in remote learning. However, this condition was not genuinely 'distributed' because the pairs of students were seated next to each other. Other experiments in real remote settings, using DPP in MOOCs for example, are required to strengthen the results presented in this paper. Also, we collected a limited number of answers for the two questionnaires, making it difficult to strongly emphasize the features requiring improvements and/or to identify the need for new features. To facilitate those large scale experiments and collect a greater amount of data, we will improve the hosting of the application which is currently deployed on a single virtual machine of our lab, and extend the use of our tool to other institutions and high schools.

7 CONCLUSION AND FUTURE WORKS

This paper proposes a new distributed pair programming application dedicated to novice programmers. It implements all the required features identified through our review of literature, excluding a communication tool. The experiment we conducted using the presented tool in distributed pair programming conditions shows no evidence of a negative impact in terms of students' perceived usability and quality of their productions, compared to using it in traditional pair programming conditions. All the anonymized data collected during the experiment are available online at recherche.data.gouv.fr for replication of this study and/or further research by the community.

On the basis of the data collected by the application, our future works will first focus on tutoring. Indicators related to both the individual and collective engagement will be designed. For students, being aware of their individual participation rate within a pair could help them balance their overall participation [23], while the level of activity of each pair should help teachers to intervene accordingly. Second, we plan to explore automatic pair formation. Indeed, research suggest that pair compatibility is often not analyzed in pair programming studies [21, 22]. Finally, we plan to release the application as open source to allow teachers to use it in other contexts, as well as researchers to contribute to its improvement.

REFERENCES

[1] Philip T. Kortum Aaron Bangor and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction* 24, 6 (2008), 574–594. https://doi.org/10.1080/10447310802205776

[2] Umair Z Ahmed, Nisheeth Srivastava, Renuka Sindhgatta, and Amey Karkare. 2020. Characterizing the pedagogical benefits of adaptive feedback for compilation errors by novice programmers. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*. Association for Computing Machinery, New York, NY, USA, 139–150.

[3] Eric Allen, Robert Cartwright, and Brian Stoler. 2002. DrJava: a lightweight pedagogic environment for Java. *SIGCSE Bull.* 34, 1 (feb 2002), 137–141. https://doi.org/10.1145/563517.563395

[4] Pasquale Ardimento, Mario Luca Bernardi, Marta Cimitile, and Giuseppe De Ruvo. 2019. Reusing bugged source code to support novice programmers in debugging tasks. *ACM Transactions on Computing Education (TOCE)* 20, 1 (2019), 1–24.

[5] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.

[6] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584

[7] John Brooke. 1996. Sus: a "quick and dirty'usability. *Usability evaluation in industry* 189, 3 (1996), 189–194.

[8] John Brooke. 2013. SUS: a retrospective. *Journal of usability studies* 8, 2 (2013), 29–40.

[9] Silvana Faja. 2011. Pair Programming as a Team Based Learning Activity: A review of research. *Issues in Information Systems* XII (01 2011), 207–216.

[10] Xinyu Fu, Atsushi Shimada, Hiroaki Ogata, Yuta Taniguchi, and Daiki Suehiro. 2017. Real-time learning analytics for C programming language courses. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (Vancouver, British Columbia, Canada) *(LAK '17)*. Association for Computing Machinery, New York, NY, USA, 280–288. https://doi.org/10.1145/3027385.3027407

[11] Brian Hempel, Justin Lubin, Grace Lu, and Ravi Chugh. 2018. Deuce: a lightweight user interface for structured editing. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 654–664. https://doi.org/10.1145/3180155.3180165

[12] Amey Karkare and Purushottam Kar. 2022. Prutor: an intelligent learning and management system for programming courses. *Commun. ACM* 65, 11 (2022), 62–64.

[13] Ioannis Karvelas. 2019. Investigating Novice Programmers' Interaction with Programming Environments. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 336–337. https://doi.org/10.1145/3304221.3325596

[14] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2021. A tutoring system to learn code refactoring. In *Proceedings of the 52nd ACM technical symposium on computer science education*. Association for Computing Machinery, New York, NY, USA, 562–568.

[15] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2022. Time-on-task metrics for predicting performance. *ACM Inroads* 13, 2 (2022), 42–49.

[16] Stamatios Papadakis. 2018. Is Pair Programming More Effective than Solo Programming for Secondary Education Novice Programmers? A Case Study. *International Journal of Web-Based Learning and Teaching Technologies* 13 (01 2018). https://doi.org/10.4018/IJWLTT.2018010101

[17] Jarred Payne, Vincent Cavé, Raghavan Raman, Mathias Ricken, Robert Cartwright, and Vivek Sarkar. 2011. DrHJ: a lightweight pedagogic IDE for Habanero Java. In *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java* (Kongens Lyngby, Denmark) *(PPPJ '11)*. Association for Computing Machinery, New York, NY, USA, 147–150. https://doi.org/10.1145/2093157.2093180

[18] Filipe D Pereira, Elaine HT Oliveira, David BF Oliveira, Alexandra I Cristea, Leandro SG Carvalho, Samuel C Fonseca, Armando Toda, and Seiji Isotani. 2020. Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming. *British journal of educational technology* 51, 4 (2020), 955–972.

[19] Rasheed Abubakar Rasheed, Amirrudin Kamsin, and Nor Aniza Abdullah. 2020. Challenges in the online component of blended learning: A systematic review. *Computers & Education* 144 (2020), 103701. https://doi.org/10.1016/j.compedu.2019.103701

[20] Peter C. Rigby and Suzanne Thompson. 2005. Study of Novice Programmers Using Eclipse and Gild. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange* (San Diego, California) *(eclipse '05)*. Association for Computing Machinery, New York, NY, USA, 105–109. https://doi.org/10.1145/1117696.1117718

[21] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 37, 4 (2011), 509–525. https://doi.org/10.1109/TSE.2010.59

[22] Maya Satratzemi, Xinogalos Stelios, and Despina Tsompanoudi. 2023. Distributed Pair Programming in Higher Education: A Systematic Literature Review. *Journal of Educational Computing Research* 61, 3 (2023), 546–577. https://doi.org/10.1177/07356331221122884 arXiv:https://doi.org/10.1177/07356331221122884

[23] Maya Satratzemi, Stelios Xinogalos, Despina Tsompanoudi, and Leonidas Karamitopoulos. 2018. Examining Student Performance and Attitudes on Distributed Pair Programming. *Scientific Programming* 2018 (Oct. 2018), e6523538. https://doi.org/10.1155/2018/6523538 Publisher: Hindawi.

[24] Julia Schenk, Lutz Prechelt, and Stephan Salinger. 2014. Distributed-Pair Programming Can Work Well and is Not Just Distributed Pair-Programming. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 74–83. https://doi.org/10.1145/2591062.2591188

[25] Till Schümmer and Stephan G Lukosch. 2009. Understanding tools and practices for distributed pair programming. *Journal of Universal Computer Science, 15 (16), 2009* 15, 16 (2009), 3101–3125.

[26] Donggil Song, Hyeonmi Hong, and Eun Young Oh. 2021. Applying computational analysis of novice learners' computer programming patterns to reveal self-regulated learning, computational thinking, and learning performance. *Computers in Human Behavior* 120 (2021), 106746.

[27] Despina Tsompanoudi and Maya Satratzemi. 2014. A Web-Based Authoring Tool for Scripting Distributed Pair Programming. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*. IEEE, Athens, Greece, 259–263. https://doi.org/10.1109/ICALT.2014.81

[28] Despina Tsompanoudi, Maya Satratzemi, and Stelios Xinogalos. 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* (Canterbury, England, UK) *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 225–230. https://doi.org/10.1145/2462476.2462500

[29] Thomas Tullis and Jacqueline Stetson. 2004. A Comparison of Questionnaires for Assessing Website Usability. *UPA Presentation* (2004), 12 p.

[30] Dietmar Winkler, Stefan Biffl, and Andreas Kaltenbach. 2010. Evaluating tools that support pair programming in a distributed engineering environment. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering* (UK) *(EASE'10)*. BCS Learning & Development Ltd., Swindon, GBR, 54–63.

[31] Krissi Wood, Dale Parsons, Joy Gasson, and Patricia Haden. 2013. It's never too early: pair programming in CS1. In *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136* (Adelaide, Australia) *(ACE '13)*. Australian Computer Society, Inc., AUS, 13–21.

[32] xAPI 2023. xAPI specification. https://github.com/adlnet/xAPI-Spec/tree/master.

[33] XecliP 2023. XecliP. https://xeclip.sourceforge.net/.