OmniaScience

Journal of Technology and Science Education
JOTSE, 2019 – 9(3): 357-367 – Online ISSN: 2013-6374 – Print ISSN: 2014-5349
https://doi.org/10.3926/jotse.638

UTILISING PAIR PROGRAMMING TO ENHANCE THE PERFORMANCE OF SLOW-PACED STUDENTS ON INTRODUCTORY PROGRAMMING

Mewati Ayub, Oscar Karnalim, Risal Risal, Wenny Franciska Senjaya, Maresha Caroline Wijanto

Faculty of Information Technology, Maranatha Christian University (Indonesia)

mewati.ayub@it.maranatha.edu, oscar.karnalim@it.maranatha.edu, laurentius.risal@it.maranatha.edu, wenny.fs@it.maranatha.edu, maresha.cw@it.maranatha.edu

Received January 2019
Accepted March 2019

Abstract

Due to its high failure rate, Introductory Programming has become a main concern. One of the main issues is the incapability of slow-paced students to cope up with given programming materials. This paper proposes a learning technique which utilises pair programming to help slow-paced students on Introductory Programming; each slow-paced student is paired with a fast-paced student and the latter is encouraged to teach the former as a part of grading system. An evaluation regarding that technique has been conducted on three undergraduate classes from an Indonesian university for the second semester of 2018. According to the evaluation, the use of pair programming may help both slow-paced and fast-paced students. Nevertheless, it may not significantly affect individual academic performance.

Keywords **–** Pair programming, Introductory programming, Slow-paced students, Quasi experiment, Computing education.

To cite this article:

Ayub, M., Karnalim, O., Risal, R., Senjaya, W.F., Wijanto, M.C. (2019). Utilising pair programming to enhance the performance of slow-paced students on introductory programming. *Journal of Technology and Science Education,* 9(3), 357-367. https://doi.org/10.3926/jotse.638

---

1. Introduction

Introductory Programming has become a main concern in Computing education due to its high failure rate (Vihavainen, Airaksinen & Watson, 2014). As a result, many teaching interventions have been proposed such as encouraging collaboration (Lasserre & Szostak, 2011), bootstrapping the materials (Mullins, Whitfield & Conlon, 2009), or contextualizing the materials with interactive media (Elvina, Karnalim, Ayub & Wijanto, 2018; Karnalim & Ayub, 2018).

Pair programming is a collaboration technique where two programmers work together to solve a particular task (Beck & Gamma, 2000). One of them acts as the driver –who has a control of the shared resources (e.g., the computer, mouse, and keyboard)– whereas other acts as the navigator or observer, whose main responsibilities are to provide advices and find errors. During the completion process, the programmers' roles are typically switched after a certain period of time (Williams & Kessler, 2003). According to

(Livermore, 2006), this technique –which is originally a part of extreme programming (Hannay, Dybå, Arisholm & Sjøberg, 2009)– has been applied on 72% of industry organisations.

There are a lot of research about pair programming as a pedagogical tool (Salleh, Mendes & Grundy, 2011). Kavitha and Ahmed (2015) evaluated the impact of pair programming on master students. Their findings suggest that pair programming may be useful for knowledge sharing.

Braught, Eby and, Wahls (2008) shows that pair programming improves paired students' individual programming skills. Further, it boosts the students' confidence about their work. In addition, they are more likely to succeed on the course.

Hannay et al. (2009) performed a meta-analysis on comparing pair programming with solo programming. The study depicted that the former is faster than the latter when the task is simple. Further, the former yields higher code quality when the task is complicated.

Saltz and Shamshurin (2017) applied pair programming on data science context. They found that pair programming can also be used in that context where the driver is a software programmer and the observer is a data science researcher.

Salleh, Mendes and Grundy (2014) investigated the impact of personality traits based on five empirical studies with a total of 594 undergraduate students. They were focused on three of five-factor personality framework: Conscientiousness, Neuroticism, and Openness. According to their study, only the last factor affects significantly in terms of resulted academic performance.

Gómez, Aguileta, Aguilar, Ucan, Rosero and Cortes-Verdin, (2017) evaluated the impact of programming workspace on pair programming. They found that, in the context of pair programming, the use of programming workspace is better to be avoided.

Lewis and Shah (2015) showed that pair programming teams with incomparable members tend to complete given task quickly due to the domination of a member. The finding is determined based on audio records while the students do pair programming.

Villamor and Rodrigo (2018) conducted a dual eye tracking on paired novice students with Cross-Recurrence Quantification Analysis (CRQA). They found that CRQA alone is not adequate to predict the successful collaboration on pair programming.

In computing education, pair programming can become a collaboration-based teaching intervention; two students collaboratively solve a particular programming assessment (Salleh et al., 2011). It may lead to several benefits such as improving the quality of works (Williams & Kessler, 2000), increasing course completion rate (Nagappan, Williams, Ferzli, Wiebe, Yang, Miller et al., 2003), and boosting up enjoyment (Mendes, Al-Fakhri & Luxton-Reilly, 2005) & confidence level (Berenson, Slaten, Williams & Ho, 2004).

Pair programming can also be used to help slow-paced students by pairing each of them with a fast-paced student and encouraging the latter to teach the former as a part of grading system. Even though it may seem to be demanding for the fast-paced one, it actually benefits them. They will gain more knowledge since, each time they teach, they re-learn the material. To our knowledge, no works have applied pair programming in such manner.

This paper reports an experiment about applying pair programming to help slow-paced students. It was conducted on three Introductory Programming classes for one semester (with 13 teaching weeks). In total, fifty-seven computing undergraduates from an Indonesian university are involved.

2. Methodology

Applying pair programming with the aim to help slow-paced students can be conducted in fourfold. At first, the students will be classified to two groups –fast-paced and slow-paced students– based on their academic performance. The academic performance can be defined either from previously-taken courses

or academic levels. All students are ranked based on their academic performance; the top-half of the list is considered as the fast-paced students while the counterpart is considered as the slow-paced students.

Secondly, each Slow-paced Student (S-Stud) is paired with a Fast-paced Student (F-Stud). To balance the proportion of each pair, F-Stud with top-K highest academic performance will be paired with a S-Stud with top-K lowest performance. For instance, if a student is ranked as the 3rd based on their academic performance, they will be paired with a student whose rank is the 3rd from the bottom of the list. It is important to note that the balancing mechanism assures that all pairs will have the same degree of academic skill to complete given assessment.

Thirdly, resulted pairs are asked to complete an assessment as a group. Unique to this pair programming, we let the pairs to use two computers each. S-Stud may need it to read given assessment problems, practise to rewrite the solution, or observe some parts of the solution. During the assessment, if F-Stud acts as the observer, S-Stud can learn through transferring F-Stud' knowledge to the computer. Otherwise, S-Stud can learn through observing how F-Stud solves the problem.

Fourthly, at the end of assessment, N S-Studs will be selected randomly from the pairs where N is defined based on lecturer' preference. Each S-Stud will be asked to explain some (or all) parts of their pair's solution. Their explanation will be scored, and that score will affect their final score and their fast-paced student's final score for that assessment. The worse their explanation is, the lower the final score will be. This step is required to assure that the fast-paced student will try to teach their counterpart about the solution.

In our case, the last step is aligned to our Introductory Programming course where each assessment has five programming tasks. Four S-Studs will be selected randomly to explain the first four programming tasks; one S-Stud is dedicated for one task. The last task is excluded since it is commonly the most difficult task, dedicated for students with outstanding logic. Each selected S-Stud will be asked to explain their pair's solution of a task. If they cannot explain it comprehensively, the final score for that student and their fast-paced student will be reduced up to 50%. To assure that the S-Stud does not memorise the solution's explanation, the lecturer usually asks one or two questions related to their code.

3. Evaluation

For evaluation, two main research questions are proposed. First, does pair programming enhance the academic performance of slow-paced students? Second, does pair programming enhance the individual academic performance of slow-paced students? These questions are referred as Q1 and Q2 respectively.

Three supplementary research questions related to pair programming are also proposed. First, does pair programming enhance students' academic performance in general? Second, does pair programming enhance students' individual academic performance in general? Third, does pair programming enhance the academic performance of fast-paced students? These questions are referred as Q3, Q4, Q5 respectively.

3.1. Evaluation Scenario

To answer the aforementioned research questions, an evaluation scenario involving three Introductory Programming classes (referred as A, B, and C respectively) were conducted on the second semester of 2018. It involves two learning techniques –pair and solo programming– which were conducted alternately in most occasions. Solo programming refers to an activity completing a programming assessment individually. It is commonly used in many Introductory Programming classes.

Table 1 shows the distribution of pair and solo programming for the Introductory Programming classes. All of them were conducted on in-class laboratory sessions where internet and removable disk were not accessible. Due to implementation issues, several adjacent sessions have the same learning technique and a class has fewer number of sessions.

Teaching Week	Class A	Class B	Class C
1st week	Solo	Solo	Solo
2nd week	Solo	Pair	Pair
3rd week	Pair	Solo	Solo
4th week	Solo	Pair	Pair
5th week	Pair	Solo	Solo
6th week	Solo	Pair	Pair
7th week	Pair	Solo	Solo
8th week	Solo	Pair	Pair
9th week	Pair	Pair	Pair
10th week	Solo	Solo	Solo
11th week	Solo	Solo	Solo
12th week	Pair	Pair	Pair
13th week	–	Solo	Solo

Table 1. Pair and Solo Programming Session Distribution

Teaching Week	Teaching Materials
1st week	Introduction to Programming Workspace
2nd week	Sequential Actions
3rd week	Branching
4th week	Nested Branching
5th week	Looping
6th week	Nested Looping
7th week	Review
Mid-Test	–
8th week	Void Function
9th week	Function with a Return Value
10th week	Array
11th week	Array Combined with Function
12th week	Matrix
13th week	Searching and Sorting
Final Test	–

Table 2. Teaching Materials

For each pair programming session, the students were divided to two groups: slow-paced and fast-paced students. Both groups were based on the students' scores on previous solo programming session. The scores were sorted in descending order where students whose score is on the top-half would be considered as fast-paced students and remaining students would be considered as the slow-paced ones.

All classes followed the same teaching roadmap (which materials can be seen on Table 2). The first seven weeks were conducted prior mid-test while the remaining weeks were conducted upon mid-test.

Each week, the students should complete five programming tasks in 150 minutes with Python as its programming language and PyCharm as its programming workspace. In terms of difficulty, the first two tasks are the most basic ones. They were only about implementing given weekly material. The following two tasks were the intermediate ones, which could be completed by applying a little logic on given material. The last task was the most advanced one. It could only be completed with outstanding logic.

The statistics of each class can be seen on Table 3. Several students were out of consideration since their course participation was less than 75%. We would argue that those students' performance may obfuscate the impact of pair programming as they did not participate in most teaching sessions.

Feature	Class A	Class B	Class C
The Number of Included Students	18	25	14
The Number of Excluded Students	5	2	2
The Total Number of Students	23	27	16

Table 3. The Statistics of Introductory Programming Classes

In this study, three research questions (Q1, Q2 and Q5) require only a particular group of students (either slow-paced or fast-paced students). For that purpose, we define the slow-paced students as those whose average assessment score (for all teaching weeks) is at bottom-half rank list while the fast-paced ones as those whose score is at top-half rank list.

3.2. Q1: Does Pair Programming Enhance the Academic Performance of Slow-Paced Students?

To evaluate whether pair programming enhances the academic performance of slow-paced students, the students' score for each pair programming session will be compared with their score on previous solo programming session. The detail of all comparisons can be seen on Table 4. There are seventeen comparisons taken from three Introductory Programming classes.

Comparison ID	Class	Pair Programming Week	Solo Programming Week
CR101	A	3rd week	2nd week
CR102	A	5th week	4th week
CR103	A	7th week	6th week
CR104	A	9th week	8th week
CR105	A	12th week	10th week
CR106	B	2nd week	1st week
CR107	B	4th week	3rd week
CR108	B	6th week	5th week
CR109	B	8th week	7th week
CR110	B	9th week	7th week
CR111	B	12th week	11th week
CR112	C	2nd week	1st week
CR113	C	4th week	3rd week
CR114	C	6th week	5th week
CR115	C	8th week	7th week
CR116	C	9th week	7th week
CR117	C	12th week	11th week

Table 4. Comparisons for Evaluating the Impact of Pair Programming on the Academic Performance of Slow-Paced Students

A pair programming session is considered effective to help slow-paced students if the students' scores are improved and the improvement is statistically significant. The significance is measured with t-test when the scores are normally distributed (with mean as its determiner). Otherwise, Wilcoxon Signed Rank will be used. All tests are based on 95% confidence level where p-value is considered significant if its score is lower or equal to 0.05.

The result can be seen on Table 5. The bolded p-values are those which are statistically significant. Six comparisons show statistically significant result where all of them correspond to positive improvement. Hence, it can be stated that pair programming may enhance the academic performance of slow-paced students. In average, by considering only statistically significant improvements, it boosts up by 20.34 points.

Comparison ID	Pair Programming Mean Score	Solo Programming Mean Score	Significance Test	P-Value	Improvement
CR101	87.56	98.44	Wilcoxon	0.813	-10.89
CR102	95.89	88.22	t-test	0.001	7.67
CR103	100.00	81.89	t-test	0.117	18.11
CR104	99.78	88.00	Wilcoxon	0.250	11.78
CR105	99.78	66.67	Wilcoxon	0.125	33.11
CR106	93.58	86.00	t-test	0.178	7.58
CR107	90.50	79.83	Wilcoxon	0.004	10.67
CR108	89.17	80.17	t-test	0.139	9.00
CR109	67.67	46.67	t-test	0.130	21.00
CR110	90.82	46.67	t-test	0.025	44.15
CR111	75.04	80.58	t-test	0.194	-5.54
CR112	95.00	93.00	t-test	0.733	2.00
CR113	96.14	90.14	t-test	0.018	6.00
CR114	89.71	90.14	t-test	0.901	-0.43
CR115	82.00	63.00	t-test	0.012	19.00
CR116	97.57	63.00	t-test	0.001	34.57
CR117	75.00	59.43	t-test	0.457	15.57

Table 5. The Improvements Gained by Utilising Pair Programming for Slow-Paced Students

3.3. Q2: Does Pair Programming Enhance the Individual Academic Performance of Slow-Paced Students?

To evaluate whether pair programming enhances the individual academic performance of slow-paced students, for each pair programming session, the students' score on its previous solo programming session will be compared with their score on the next solo programming session. The detail of all comparisons can be seen on Table 6. Fourteen comparisons are considered.

A pair programming session is considered effective to enhance the individual performance of slow-paced students if the students' score upon pair programming is higher than their score prior pair programming and the improvement is statistically significant. Similar with previous evaluation, the significance is measured with t-test when the scores are normally distributed and Wilcoxon Signed Rank if the scores are not normally distributed. Both tests are based on 95% confidence level.

The result can be seen on Table 7. The bolded p-values are those which are statistically significant. Six comparisons yield statistically significant result. However, only two of them show positive improvement. It may be caused by the increasing difficulty of teaching materials where the next solo programming session was commonly conducted two weeks after the previous solo programming session. Another possible cause is that several slow-paced students were not able to take advantages from their pair programming session.

Comparison ID	Class	Next Solo Programming Week	Previous Solo Programming Week
CR201	A	4th week	2nd week
CR202	A	6th Week	4th week
CR203	A	8th Week	6th week
CR204	A	10th Week	8th week
CR205	B	3rd Week	1st week
CR206	B	5th Week	3rd week
CR207	B	7th Week	5th week
CR208	B	10th Week	7th week
CR209	B	13th Week	11th week
CR210	C	3rd Week	1st week
CR211	C	5th Week	3rd week
CR212	C	7th Week	5th week
CR213	C	10th Week	7th week
CR214	C	13th Week	11th week

Table 6. Comparisons for Evaluating the Individual Impact of Pair Programming on the Academic Performance of Slow-Paced Students

Comparison ID	Next Solo Programming Mean Score	Previous Solo Programming Mean Score	Significance Test	P-Value	Improvement
CR201	88.22	98.44	t-test	0.001	-10.22
CR202	81.89	88.22	t-test	0.559	-6.33
CR203	88.00	81.89	Wilcoxon	0.047	6.11
CR204	66.67	88.00	Wilcoxon	0.125	-21.33
CR205	79.83	86.00	t-test	0.096	-6.17
CR206	80.17	79.83	t-test	0.970	0.33
CR207	46.67	80.17	t-test	0.013	-33.50
CR208	68.55	46.67	t-test	0.210	21.88
CR209	71.50	80.58	t-test	0.029	-9.08
CR210	90.14	93.00	t-test	0.297	-2.86
CR211	90.14	90.14	t-test	1.000	0.00
CR212	63.00	90.14	t-test	0.001	-27.14
CR213	79.29	63.00	t-test	0.023	16.29
CR214	48.71	59.43	t-test	0.526	-10.71

Table 7. The Individual Improvements Gained by Utilising Pair Programming for Slow-Paced Students

3.4. Q3: Does Pair Programming Enhance Students' Academic Performance in General?

To evaluate whether pair programming enhances students' academic performance in general, each pair programming session's score will be paired with their previous solo programming session's. It results in 17 comparisons (listed on Table 4) with both slow-paced and fast-paced students are considered. The effectiveness of pair programming is measured based on statistical significance test; t-test is used for normal distribution. Otherwise, Wilcoxon Signed Rank will be used. All of them are calculated under 95% confidence level.

The result can be seen on Table 8. The bolded p-values are those which are statistically significant. Eight of nine statistically significant results are positive. It can be therefore stated that pair programming may positively affect the students' academic performance. By averaging only statistically significant results, it boosts up by 11.61 points.

Comparison ID	Pair Programming Mean Score	Solo Programming Mean Score	Significance Test	P-Value	Improvement
CR101	93.56	98.22	Wilcoxon	0.993	-4.67
CR102	97.30	93.44	t-test	0.016	3.86
CR103	100.00	87.94	Wilcoxon	0.001	12.06
CR104	99.78	93.33	Wilcoxon	0.188	6.44
CR105	99.78	83.33	Wilcoxon	0.188	16.44
CR106	93.35	88.65	Wilcoxon	0.062	4.70
CR107	94.52	86.26	Wilcoxon	0.002	8.26
CR108	89.96	84.39	t-test	0.107	5.57
CR109	72.74	63.43	t-test	0.221	9.30
CR110	93.23	63.43	Wilcoxon	0.002	29.79
CR111	77.80	86.13	t-test	0.002	-8.33
CR112	96.86	94.29	Wilcoxon	0.117	2.57
CR113	97.36	84.36	Wilcoxon	0.003	13.00
CR114	83.29	93.43	Wilcoxon	0.079	-10.14
CR115	82.57	71.07	t-test	0.015	11.50
CR116	96.71	71.07	t-test	0.001	25.64
CR117	82.21	73.50	Wilcoxon	0.030	8.71

Table 8. The Improvements Gained by Utilising Pair Programming in General

3.5. Q4: Does Pair Programming Enhance Students' Individual Academic Performance in General?

To evaluate whether pair programming enhances students' individual academic performance in general, for each pair programming session, the students' score on its previous and next solo programming session will be compared to each other. It results in 14 comparisons displayed on Table 6 with both slow-paced and fast-paced students are considered. Pair programming's effectiveness is defined using statistical significance test –t-test for normal distribution and Wilcoxon Signed Rank for unnormal distribution– with 95% confidence level.

The result can be seen on Table 9. The bolded p-values are those which are statistically significant. Only two of six statistically significant results are positive. It may be caused by two possible reasons: the increasing difficulty on the next solo programming session and the incapability to benefit from pair programming.

Comparison ID	Next Solo Programming Mean Score	Previous Solo Programming Mean Score	Significance Test	P-Value	Improvement
CR201	93.44	98.22	t-test	0.009	-4.78
CR202	87.94	93.44	Wilcoxon	0.358	-5.50
CR203	93.33	87.94	Wilcoxon	0.006	5.39
CR204	83.33	93.33	Wilcoxon	0.063	-10.00
CR205	86.26	88.65	Wilcoxon	0.513	-2.39
CR206	84.39	86.26	Wilcoxon	0.235	-1.87
CR207	63.43	84.39	t-test	0.004	-20.96
CR208	73.36	63.43	t-test	0.335	9.93
CR209	77.52	86.13	t-test	0.001	-8.61
CR210	84.36	94.29	Wilcoxon	0.115	-9.93
CR211	93.43	84.36	Wilcoxon	0.144	9.07
CR212	71.07	93.43	t-test	0.001	-22.36
CR213	84.43	71.07	t-test	0.002	13.36
CR214	63.43	73.50	t-test	0.219	-10.07

Table 9. The Individual Improvements Gained by Utilising Pair Programming in General

3.6. Q5: Does Pair Programming Enhance the Academic Performance of Fast-Paced Students?

To evaluate whether pair programming enhances the academic performance of fast-paced students, each pair programming session's score will be paired with their previous solo programming session's. In total, it leads to 17 comparisons (as displayed on Table 4) with only fast-paced students are considered. The effectiveness of pair programming is defined using statistical significance test –t-test for normal distribution and Wilcoxon Signed Rank for unnormal distribution– with 95% confidence level.

The result can be seen on Table 10. The bolded p-values are those which are statistically significant. Four of five results are statistically significant and positive. It can be therefore stated that pair programming may help fast-paced students to get higher score. In average, by considering only statistically significant results, it increases these students' score by 6.18 points.

Comparison ID	Pair Programming Mean Score	Solo Programming Mean Score	Significance Test	P-Value	Improvement
CR101	99.56	98.00	Wilcoxon	0.125	1.56
CR102	98.71	98.67	Wilcoxon	0.250	0.04
CR103	100.00	94.00	Wilcoxon	0.004	6.00
CR104	99.78	98.67	Wilcoxon	0.500	1.11
CR105	99.78	100.00	Wilcoxon	1.000	-0.22
CR106	93.09	91.55	t-test	0.394	1.55
CR107	98.91	93.27	Wilcoxon	0.008	5.64
CR108	90.82	89.00	t-test	0.570	1.82
CR109	78.27	81.73	t-test	0.484	-3.45
CR110	95.64	81.73	t-test	0.010	13.91
CR111	80.82	92.18	t-test	0.001	-11.36
CR112	98.71	95.57	t-test	0.251	3.14
CR113	98.57	78.57	t-test	0.193	20.00
CR114	76.86	96.71	t-test	0.189	-19.86
CR115	83.14	79.14	t-test	0.465	4.00
CR116	95.86	79.14	t-test	0.012	16.71
CR117	89.43	87.57	t-test	0.627	1.86

Table 10. The Improvements Gained by Utilising Pair Programming for Fast-Paced Students

4. Conclusions

This paper proposes a learning technique that involves pair programming for helping slow-paced students on Introductory Programming. The technique has been applied on three classes (with fifty-seven computing undergraduates from an Indonesian university on board). According to our evaluation, the technique may help both slow-paced students and fast-paced students, even though the impact is more salient on the slow-paced ones. Despite its positive impact, pair programming may not affect the students' individual academic performance due to the incapability to benefit from pair programming.

For future work, we plan to replicate this study on advanced courses (such as Object-Oriented Programming course) and check whether these results are consistent. Further, we also plan to propose another learning technique which are more focused on enhancing individual academic performance.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors received a financial support for the publication of this article from Maranatha Christian University, Indonesia.

References

Beck, K., & Gamma, E. (2000). *Extreme programming eXplained : embrace change*. Addison-Wesley.

Berenson, S.B., Slaten, K.M., Williams, L., & Ho, C.W. (2004). Voices of women in a software engineering course: reflections on collaboration. *Journal on Educational Resources in Computing*, 4(1), 3. https://doi.org/10.1145/1060071.1060074

Braught, G., Eby, L.M., & Wahls, T. (2008). The effects of pair-programming on individual programming skill. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education - SIGCSE '08* (40, 200). New York, New York, USA: ACM Press. https://doi.org/10.1145/1352135.1352207

Elvina, E., Karnalim, O., Ayub, M., & Wijanto, M.C. (2018). Combining program visualization with programming workspace to assist students for completing programming laboratory task. *Journal of Technology and Science Education*, 8(4), 268. https://doi.org/10.3926/jotse.420

Gómez, O.S., Aguileta, A.A., Aguilar, R.A., Ucan, J.P., Rosero, R.H., & Cortes-Verdin, K. (2017). An Empirical Study on the Impact of an IDE Tool Support in the Pair and Solo Programming. *IEEE Access*, 5, 9175-9187. https://doi.org/10.1109/ACCESS.2017.2701339

Hannay, J.E., Dybå, T., Arisholm, E., & Sjøberg, D.I.K. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7), 1110-1122. https://doi.org/10.1016/J.INFSOF.2009.02.001

Karnalim, O., & Ayub, M. (2018). A Quasi-Experimental Design to Evaluate the Use of PythonTutor on Programming Laboratory Session. *International Journal of Online Engineering (IJOE)*, 14(02), 155-164.

Kavitha, R.K., & Ahmed, M.S.I. (2015). Knowledge sharing through pair programming in learning environments: An empirical study. *Education and Information Technologies*, 20(2), 319-333. https://doi.org/10.1007/s10639-013-9285-5

Lasserre, P., & Szostak, C. (2011). Effects of team-based learning on a CS1 course. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE'11* (133). New York, New York, USA: ACM Press. https://doi.org/10.1145/1999747.1999787

Lewis, C.M., & Shah, N. (2015). How Equity and Inequity Can Emerge in Pair Programming. In *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER'15* (41-50). New York, New York, USA: ACM Press. https://doi.org/10.1145/2787622.2787716

Livermore, J.A. (2006). What Elements of XP are being Adopted by Industry Practitioners? In *Proceedings of the IEEE SoutheastCon 2006* (149-152). IEEE. https://doi.org/10.1109/second.2006.1629340

Mendes, E., Al-Fakhri, L.B., & Luxton-Reilly, A. (2005). Investigating pair-programming in a 2nd-year software development and design computer science course. *ACM SIGCSE Bulletin*, 37(3), 296-300. https://doi.org/10.1145/1067445.1067526

Mullins, P., Whitfield, D., & Conlon, M. (2009). Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges*, 24(3), 136-143.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C. et al. (2003). Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin*, 35(1), 359-362. https://doi.org/10.1145/611892.612006

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 37(4), 509-525. https://doi.org/10.1109/TSE.2010.59

Salleh, N., Mendes, E., & Grundy, J. (2014). Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments. *Empirical Software Engineering*, 19(3), 714-752. https://doi.org/10.1007/s10664-012-9238-4

Saltz, J.S., & Shamshurin, I. (2017). Does pair programming work in a data science context? An initial case study. In *2017 IEEE International Conference on Big Data (Big Data)* (2348-2354). IEEE. https://doi.org/10.1109/BigData.2017.8258189

Vihavainen, A., Airaksinen, J., & Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research - ICER'14* (19-26). New York, New York, USA: ACM Press. https://doi.org/10.1145/2632320.2632349

Villamor, M., & Rodrigo, M.M. (2018). Predicting Successful Collaboration in a Pair Programming Eye Tracking Experiment. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization - UMAP'18* (263-268). New York, New York, USA: ACM Press. https://doi.org/10.1145/3213586.3225234

Williams, L.A., & Kessler, R.R. (2000). The effects of "pair-pressure" and "pair-learning" on software engineering education. In *Thirteenth Conference on Software Engineering Education and Training* (59-65). IEEE Comput. Soc. https://doi.org/10.1109/CSEE.2000.827023

Williams, L., & Kessler, R.R. (2003). *Pair programming illuminated*. Addison-Wesley.

Published by OmniaScience (www.omniascience.com)

Journal of Technology and Science Education, 2019 (www.jotse.org)