Revista Iberoamericana para la Investigación y el Desarrollo Educativo
ISSN 2007 - 7467

https://doi.org/10.23913/ride.v10i19.549

Artículos Científicos

Comparison of Pair and Solo Programming through software metrics in University Students' Projects

Comparación de la programación por pares e individual a través de las métricas de software de proyectos de estudiantes universitarios

Comparação da programação entre pares e individual através de métricas de software para projetos de estudantes universitários

Ramón Ventura Roque Hernández
Universidad Autónoma de Tamaulipas, México
rvHernandez@uat.edu.mx
https://orcid.org/0000-0001-9727-2608

Jesús Cárdenas Domínguez
Universidad Autónoma de Tamaulipas, México
jesus.cardenas.d@gmail.com
https://orcid.org/0000-0001-9962-796X

Adán López Mendoza
Universidad Autónoma de Tamaulipas, México
aLopez@uat.edu.mx
https://orcid.org/0000-0003-4801-640X

Juan Antonio Herrera Izaguirre
Universidad Autónoma de Tamaulipas, México
jaHerrera@uat.edu.mx
https://orcid.org/0000-0002-4088-7772

Carlos Manuel Juárez Ibarra
Universidad Autónoma de Tamaulipas, México
cJuarez@docentes.uat.edu.mx
https://orcid.org/0000-0003-4310-8938

Revista Iberoamericana para la Investigación y el Desarrollo Educativo
ISSN 2007 - 7467

Abstract

Introduction: Pair Programming is an agile practice that can be used both in software development for business and in the teaching of programming in university courses. **Objective:** This paper presents a research that was conducted to compare pair programming and solo programming in university courses from the perspective of the metrics of the programs created by freshmen enrolled in the Bachelor Degree in Information Technologies. **Method:** The participants were divided into two groups: those who applied pair programming and those who programmed individually. Both developed the same program under the same conditions. The following metrics were analyzed in their programs: Number of Statements, Percentage of Comments, Maximum Depth, Average Depth, Maximum Complexity, Number of methods per class, Number of calls per method and Number of sentences per method. The values of the metrics were obtained by the *Source Monitor* software. Then, *Mann-Whitney* tests were performed in SPSS. **Results:** Results showed that students that worked in pairs wrote code with more statements (p=0.038, U=17.00) and a higher level of depth (p=0.032, U=18.00) compared to solo programmers. **Conclusions:** This paper contributes to the field of software development teaching by providing quantitative empirical evidence on the effectiveness of pair programming. It is concluded that pair programming can be an appropriate educational approach for the beginner's software development university courses.

Keywords: Computer Programming, Software, Higher Education, Measurement.

Resumen

Introducción: La programación por pares es una práctica ágil que puede ser utilizada tanto en el desarrollo de software en los negocios como en la enseñanza universitaria de la programación. **Objetivo:** Este artículo presenta una investigación que se realizó para comparar la programación por pares y en solitario en cursos universitarios considerando las métricas de los programas creados por estudiantes de reciente ingreso a una carrera universitaria en tecnologías de la información. **Método:** Se dividió a los participantes en dos grupos: uno aplicó la programación por pares y otro programó individualmente. Ambos desarrollaron el mismo programa bajo las mismas condiciones. Las siguientes métricas fueron analizadas en sus programas: número de sentencias, porcentaje de comentarios, profundidad máxima, profundidad promedio, complejidad máxima, número de métodos por clase, número de llamadas por método y número de sentencias por método. Los

valores de las métricas fueron obtenidos con el software *Source Monitor*. Posteriormente se realizaron pruebas *Mann-Whitney* en SPSS. **Resultados:** Se observó que quienes trabajaron en pares escribieron código con más sentencias (p=0.038, U=17.00) y mayor nivel de profundidad (p=0.032, U=18.00) que quienes programaron individualmente. **Conclusiones:** Este artículo contribuye a la enseñanza del desarrollo de software al proveer evidencia empírica cuantitativa de la efectividad de la programación por pares. Se concluye que la programación por pares puede ser un enfoque educativo apropiado para los primeros cursos universitarios de desarrollo de software.

Palabras clave: Programación informática, Software, Enseñanza superior, Medición.

Resumo

Introdução: A programação por pares é uma prática ágil que pode ser usada tanto no desenvolvimento de software nos negócios quanto no ensino universitário de programação. Objetivo: Este artigo apresenta uma investigação realizada para comparar a programação entre pares e solo em cursos universitários, considerando as métricas dos programas criados por estudantes recentes que ingressam em uma carreira universitária em tecnologia da informação. Método: Os participantes foram divididos em dois grupos: um aplicado por pares e outro programado individualmente. Ambos desenvolveram o mesmo programa sob as mesmas condições. As métricas a seguir foram analisadas em seus programas: número de frases, porcentagem de comentários, profundidade máxima, profundidade média, complexidade máxima, número de métodos por classe, número de chamadas por método e número de frases por método. Os valores métricos foram obtidos com o software Source Monitor. Posteriormente, os testes de Mann-Whitney foram realizados no SPSS. Resultados: Observou-se que aqueles que trabalhavam em pares escreviam código com mais sentenças (p = 0,038, U = 17,00) e maior nível de profundidade (p = 0,032, U = 18,00) do que aqueles que programavam individualmente. Conclusões: Este artigo contribui para o ensino do desenvolvimento de software, fornecendo evidências empíricas quantitativas da eficácia da programação por pares. Conclui-se que a programação por pares pode ser uma abordagem educacional apropriada para os primeiros cursos universitários de desenvolvimento de software.

Palavras-chave: Programação de computadores, Software, Ensino superior, Medição.

Revista Iberoamericana para la Investigación y el Desarrollo Educativo
ISSN 2007 - 7467

Fecha Recepción: Marzo 2019 **Fecha Aceptación:** Septiembre 2019

Introduction

In the beginning of software engineering there were methodologies that set order to the developing process, but they were not flexible nor adaptable to the needs of the users, which were more demanding every time. They were unsuitable for environments with changing requirements and high priority for quality and delivery time. The developmental, incremental and agile methodologies emerged later with simplified visions focused on people with the objective of quickly obtaining good quality programs to satisfy the users' requirements.

Extreme programming, or XP is an agile approach to create software that proposes a model guide of development. XP eliminates the need to spend time in tedious and rigorous tasks, such as creation and extensive revision of documents and handling of huge volume of requirements lists. Among the variety of practices of XP, pair programming stands out, which consists of a couple of programmers that always use the same computer with defined and changing roles.

The opinion about the use of this approach can be controversial. There are authors that have found positive results and recommend it; on the other hand, other researchers prefer other means to work. It has been found that the perceptions about the effectiveness of pair programming vary according to the amount of time that the programmers have worked with it. For example, those who have more experience working in pairs are convinced that this technique helps to reduce costs, while those who have used it less, perceive the contrary (Sun, Marakas, & Aguirre-Urreta, 2016). In the university area, previous research suggests that an agile approach will be useful in university courses and that working in pairs could be beneficial for students. However, there are no studies that lead to conclusive results. If pair programming is applied to beginner programming university courses, in which way will it be useful? How will it affect the programs developed by the students? The objective of this research was to compare pair programming and solo programming in a beginner programming university course. For this comparison, the metrics of the projects that were developed by freshmen using both work methods were taken into account.

The paper is organized as follows: the next section shows the software background and its development; it emphasizes the agility, Extreme Programming and Pair Programming. Then, the materials and methods used to carry out the investigation are explained. The results and their discussion are presented afterwards. Finally, the conclusions and the possibilities for future works are explored.

Background

Software Development

Software is a basic component of computer systems, which includes programs and data that make the hardware work. According to Forouzan (Forouzan, 2003), software is divided into two categories: system and application. System software allows the computer to efficiently manage the resources such as memory, storage and processor. The application software provides features to perform a concrete task oriented to directly help the final user. Regardless of the type of software in question, creating it means to develop a program from instructions or statements that are written using a programming language; its purpose is to tell the hardware what to do (Sánchez-Montoya, 1995). The set of instructions written by the programmer is called source code. Software development means much more than only writing these instructions; it also includes the participation of the work team in the different activities of the program creation and the management of the process itself.

Software crisis is known as the phenomenon which main characteristic is the failure in software development projects due to exceeded budgets, requirements and deadlines that are not met, and the work team's lack of skills. Software development is risky and hard to control due to the multiple factors that intervene in the process. Brook (Brooks, 1987) acknowledges that complexity is part of the software essence and not product of fate. In recent years, the methodologies to ensure better process control have evolved with the aim of solving the crisis previously described. Software Engineering is the discipline that covers processes, methods and tools that are used to produce computer programs. Thanks to this field of study, the activities of the work team can be efficiently organized, and repeatable approaches can be applied to ensure the quality of the development process and the final software.

Agility in software development

Agility is a combination of philosophy and development guidelines (Pressman, 2014), where change is accepted and perceived as a regular phenomenon; therefore, it is possible to continuously provide an adequate response to it. The agile philosophy is specified in the agile manifest for software development (Agile Alliance, 2018), where fast software development has a higher priority than documentation and people have greater value than processes. There are different agile approaches, and each one accentuates the philosophies of the manifest in a larger or smaller scale; however, all of them offer different ways to achieve the same objective. Some of the agile methods emphasized by Martin (Martin, 2011) are: Extreme Programming (XP), Adaptive Development, Scrum, Dynamic Systems Development Method, and Crystal.

Extreme Programming: an agile approach

Extreme programming is an agile approach used to develop software (Fowler, 2018) that includes twelve practices aimed at obtaining working software in the shortest time. It is focused on people that produce and use the software (Beck & Andres, 2004). One of its main advantages is that it reduces the cost of implementing changes during the entire life cycle of the system. Software starts in a small scale, and then it becomes more functional as a result of the client's feedback, who is part of the work team. Efforts are determined to obtain what is needed and not wasted in developing additional features.

Pair programming: a practice of Extreme Programming

Pair programming is one of the fundamental practices of Extreme Programming. Two programmers work together in the same space and with the same computer with the purpose of producing software collaboratively through all the activities involved in this process. One of the programmers takes the keyboard and mouse and plays a guiding role; the other one is the navigator, who is in charge of observing, make timely revisions, manage tasks, locate faults, and see beyond the source code (Beck & Andres, 2004). Both of them act as a single intelligent unit that adopts the responsibility of everything that it does (Williams, Kessler, Cunningham, & Jeffries, 2000). Both roles are periodically interchangeable.

Pair programming in university

The agile practices to develop software are important nowadays in the business world since they have been proved to have a positive effect in projects. Experts consider that their use can be encouraged from within the programming courses (Kropp & Meier, 2013; von Wangenheim, Savi, & Ferreti Borgatto, 2013), where potential present and future software developers are trained. Smith, Giugliano y DeOrio (2017) think that encouraging pair programming in university produces relevant benefits for students. They studied the long term effects of using pair programming in beginner courses and found its positive effect on academic performance in more advanced courses. Pair programming promotes confidence, course completion and pass rates; this approach can be beneficial for all students, especially for women because it overcomes many factors that may prevent women's participation in computer science (Werner, Hanks, & McDowell , 2005).

Pair programming has been researched from different perspectives; however, its application within the classrooms has not been studied enough (Prabu & Duraisamy, 2015) and there are still discrepancies in the results and opinions about its true effectiveness (Coman, Robillard, Silliti, & Succi, 2014; Salleh, Mendes, & Grundy, 2014). For this reason, adopting an objective criteria in the pair programming research against solo programming is relevant. Software metrics have been used previously for this purpose. For instance, in Hulkko & Abrahamsson (2005) the defects in projects made in C++ and Java were analyzed. The least amount of defects was found in one of the projects that was developed in pairs.

On the other hand, the works of Tsompanoudi, Satratzemi, & Xinogalos (2016), Zacharis (2011) and Mohd Zin, Idris, & Kumar Subramaniam (2006) have studied pair programming in university courses using online tools; they have found positive results that recommend this approach as an alternative to solo programming. In the same way, the work of McDowell, Werner, Bullock, & Fernald (2006) found favorable results with pair programming in regards to the performance, confidence, and collaborative learning developed by the students.

Measurement and metric analysis

Measurement is a process through which a value is assigned to a programming feature with the purpose of obtaining useful references to evaluate the software quality. On the other hand, the metrics are features of software that can be objectively measured. There are metrics oriented to the development process for example the average effort to perform a task; there are also metrics

oriented to product, for example, the number of lines of source code and the level of cyclomatic complexity.

Sommerville (Sommerville, 2015) explains the use of a measurement performed in two different scenarios to determine the usefulness of a tool. Measurements performed on software are used in the decision making process oriented to resource optimization; they are also fundamental elements for empirical software engineering, an area of study that uses experimentation and data gathering for hypotheses testing in the software development field.

Method

Participants

This study had the participation of 26 freshmen obtaining a bachelor's degree of Information Technologies at a Mexican University (Note: This information was removed for confidentiality reasons). They were taking the course "Fundamentals of Computer Science and Methodology of Programming". The students were randomly distributed as follows: 12 students were assigned to work in 6 pairs and 14 students were assigned to work individually.

Scenario

The students worked in the programming lab at the University campus, where this study was conducted. This lab was chosen because the students work there regularly. This lab has 30 computers with the following features: i5 Intel Processor, 8GB of RAM memory, 500 GB of storage capacity in hard drive and a 21" flat screen monitor. They used Visual Studio 2013 as Integrated Development Environment (IDE) with Visual Basic. NET and a Console Project.

Instrument

The instrument used to evaluate the differences between the software development through pair programming and solo programming was defined by the following metrics: number of sentences, percentage of comment lines, maximum depth, average depth, maximum complexity, number of methods per class, number of calls per method and number of statements per method, which were obtained with the SourceMonitor software (Campwood, 2018) for each of the projects developed by the participants.

Procedure

The research was conducted in a single regular two-hour session of the course "Fundamentals of Computer Science and Programming Methodology". Before starting, there was a waiting period of 15 minutes; students who arrived late were not allowed to participate. Details from this experiment were not disclosed previously; thus, the students did not know they were going to be a part of it. The participants were not given any compensation or incentive. First, an introductory twenty minute talk was given. They were explained the way they would be working, and researchers avoided encouraging trends in the participants' perception of the studied approaches. Then, the students were organized to work in one of two modes: pairs or solo. This was done using a list of the attendees and assigning each student one of the two ways of working with the help of the evenly distributed random numbers found in the Coss Bu (Coss Bu, 1995) materials. After that, they were informed about the programming problem that they would have to solve (see Table 1). Everyone was asked to develop the same program under the same conditions.

Table 1. Description of the program developed by students.

Develop a program to request a number from the user and perform the following operations with it: 1)Add the same number to it. 2)Multiply it by the same number. 3)Divide it between (the same number plus 1). 4)Subtract (the same number minus 1) to it. The program must also provide the sum of all these results plus the number that the user entered. If this total sum is less than 30, it must print the message "the sum is too small". If the total sum is bigger than 50, it must print the message "the sum is too big". No other message should be printed otherwise. Finally, the total sum must be stored and listed in a log that contains all the operations performed so far; it must include the date and hour of execution as well.

Source: Own preparation

Those who worked in pairs were assigned only one computer, in which both students had to program. Those who worked individually were assigned a computer for each one. Due to the design of the facilities where the experiment was performed, all the students used adjacent computers, but pairs and individual programmers were alternately distributed. The maximum time to develop the program was one hour. For those who worked in pairs, the time to switch roles between guide and navigator was five minutes. Such periods were timed, publicly announced and supervised to be fulfilled. Copying code from other students was not allowed. Participants had free internet access for surfing, but chats and social media were not allowed.

It was also presented to them a descriptive illustration of the program's execution. This way, the requirements were more evident. Finally, the participants were asked to compress their projects into a single ZIP format file and upload it to the Blackboard Learning System (UAT, 2018). Then, the research team downloaded and processed the projects with the *Source Monitor* software. Finally, the results were entered in a Microsoft Excel spreadsheet and exported to the SPSS Software (Wagner, 2014), where the statistical processes were performed.

Type of study

This study had a "Posttest-only control group design" or "After-only with control design" as described in the book of Zikmund, Barry, Carr, & Griffin (2013).

Conceptual and operational definition of variables

The studied variables in this research with their conceptual definition are presented in Table 2. Their operational definition is described by the measurement of the metrics in each one of the projects according to the *Source Monitor* Software (Campwood, 2018).

Table 2. Definition of variables in this research.

Variable	Conceptual Definition according to the documentation in Campwood (2018).
Number of sentences	Defines the reserved words and sentences of the language such as if, foreach, do/until, for and while, the operations to assign values to variables, calls to methods, definition of variables (Dim and Redim), methods, attributes, and the exception sentences: try, catch, finally.
Percentage of comments	It is the proportion of the number of lines of the program that are marked as comments compared to the number of total lines of the program file.
Maximum depth	It refers to the maximum level of nesting in the source code (this is the deepest level of code blocks within others).
Average depth	It is the weighted average of the depth of the blocks of all the sentences in a program.
Maximum complexity	It is the biggest complexity value observed in the methods of the analyzed project.
Number of methods per class	This metric is the total number of methods divided by the total number of classes, interfaces and structures.
Number of calls per method	It is the result of dividing the total number of calls to other methods by the number of methods in that project.
Number of sentences per method	It is the result of dividing the total number of sentences within all the methods of the project by the number of methods of that project.

Source: Own preparation

Data Analysis

The projects developed by the students were analyzed with the *Source Monitor* software, and the metrics indicated in Table 2 were obtained for each project. The results were entered in a Microsoft Excel spreadsheet and then exported to SPSS, where a preliminary analysis of the data was performed. Then, *Mann-Whitney* tests were conducted to see if the arithmetic differences observed between the metrics of the groups were statistically significant with a 95% confidence reference.

Results

As a result of the analysis performed, the descriptive data of the metrics for each group was obtained first. This information is summarized in Table 3, where the median and inter-quartile range are presented for Solo and Pair Programming groups.

Table 3. Descriptive data of the analyzed metrics.

Metric	Solo Programming		Pair Programming	
	Median	Inter-quartile range	Median	Inter-quartile range
Number of Sentences	17.50	12	28	3
Percentage of comments	0	1.5	0	1.6
Maximum depth	2.00	2	4.00	0
Average depth	1.86	.30	2.14	.30
Maximum complexity	1.00	3	3.00	4
Number of methods per class	0	0	0	0
Number of calls per method	0	0	0	2.25
Number of sentences per method	0	0	0	6.50

Source: Own preparation

The results of the *Mann-Whitney* test are presented in Table 4. This test was performed to find significant differences in the metrics of both groups.

Table 4. Results of the *Mann-Whitney* test.

Metric	PValue	*Mann-Whitney* U
Number of Sentences	.038	17.00
Percentage of comments	.768	39.50
Maximum depth	.032	18.00
Average depth	.025	15.00
Maximum complexity	.198	27.00
Number of methods per class	.526	38.00
Number of calls per method	.127	35.00
Number of sentences per method	.476	37.50

Source: Own preparation

Finally, the mean ranks for the metrics with statistical significant differences (PValue <0.05) according to the *Mann-Whitney* tests are shown in Table 5. It can be noted that pair programmers used more sentences and wrote code with higher level of depth than solo programmers.

Table 5. Mean ranks obtained in *Mann-Whitney* tests for statistical significant results.

Metric	Individual	Pairs	Conclusion
Number of sentences	8.71	14.67	The participants that worked in pairs used more instructions in their programs than those who worked alone.
Maximum depth	8.79	14.50	The participants that worked in pairs wrote programs with more code blocks than those who worked alone.
Average depth	8.57	15.00	

Source: Own preparation

Discussion

The *Mann-Whitney* tests revealed that only the number of sentences and the level of depth can be considered significant. It was noted that the participants that worked in pairs wrote a higher number of sentences and their code had higher levels of depth. This means that students that applied pair programming used the reserved words of the programming language more frequently, and they were also capable of writing source code with more structures of nested blocks. It is true that the highest levels of nesting produce a more complex code because it can be more difficult to read and analyze. Nevertheless, it must be considered that the selection and iteration instructions like the ones needed to solve the exercise presented to the students in this research increase the depth metrics naturally. We consider the values obtained by pair programmers as positive results of using pair programming in a beginner university course. When working in pairs, the students produced more elaborated programs that imply a better use of the programming language and a higher performance in the participants. These findings suggest that the work in pairs could be more efficient and give better results than solo programming, such as expressed by the theory of Kent Beck (Beck & Andres, 2004). This is also consistent with the benefits of pair programming found

by Werner (Werner, Hanks, & McDowell , 2005) and with the opinions in the work of Smith, Giugliano y DeOrio (2017).

It must be taken into consideration that we did not conduct an additional analysis of the individual projects to investigate if the code written by the students could be improved to increase the performance of the programs or the legibility of the source code. On the other hand, it should be also contemplated that the participants were students without previous experience on collaborative development; their only experience was on solo programming, since it is the way they usually work.

Conclusions

This paper presented a study based on the analysis of software metrics to compare the development results of solo and pair programming in a university programming course. Statistical significant differences were found between both groups in the number of sentences written and the level of depth in the source code. Pair programmers wrote code with a higher number of statements and a higher level of depth than solo programmers. These findings allow to foresee that the implementation of pair programming in university courses could be appropriate to motivate students to write more exhaustive programs with more structural richness. As future work, it is suggested to increase the number of metrics studied in the projects developed by the participants and to conduct a further analysis on each of the projects to evaluate the quality of the code. We recommend that pair programming continue being used and studied in educational settings. This will continuously generate more specific knowledge and will help to deeply understand how Pair Programming **contributes** to learning.

Acknowledgements

Authors would like to express their deep gratitude to the Autonomous University of Tamaulipas for providing them with valuable support and helpful resources while this research was being conducted.

References

Agile Alliance (2018). Manifiesto por el Desarrollo Ágil de Software. Obtained from http://agilemanifesto.org/iso/es/

Beck, Kent, & Andres, Cynthia (2004). eXtreme Programming explained. Embrace change. United States: Addison Wesley.

Brooks, Fred (1987). No Silver Bullet: Essence and Accidents of Software Engineering. Computer, 20(4), 10-19. doi: 10.1109/MC.1987.1663532

Campwood (2018). Source Monitor. Obtained from http://www.campwoodsw.com/sourcemonitor.html

Coman, I., Robillard, P., Silliti, A. & Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. The Journal of Systems and Software, 91(3), 124-134. doi: 10.1016/j.jss.2013.12.037.

Coss Bu, Raúl (1995). Simulación: Un enfoque práctico. México D.F., México: Limusa Noriega.

Forouzan, Behrouz (2003). Introducción a la ciencia de la computación. México, D.F., México: Thomson.

Fowler, Martin (2018). Martin Fowler. Obtained from http://www.martinfowler.com/articles/newMethodology.html

Hulkko, Hanna, & Abrahamsson, Pekka (2005). A multiple case study on the impact of pair programming on product quality. Proceedings of International Conference on Software Engineering. Saint Louis, MO, USA. doi: 10.1109/ICSE.2005.1553595

Kropp, Martin & Meier, Andreas (2013). Teaching Agile Software Development at University Level. IMVS Fokus Report, 7(1), 15-20.

Martin, Robert (2011). Agile Software Development, Principles, Patterns, and Practices. International Edition. United States: Prentice Hall.

McDowell, C., Werner, L., Bullock, H. & Fernald, J. (2006). Pair Programming Improves Student Retention, Confidence, and Program Quality. Communications of the ACM, 49(8), 90-95. doi:10.1145/1145287.1145293

Mohd Zin, A., Idris, S. & Kumar Subramaniam, N. (2006). Implementing Virtual Pair Programming in E-Learning Environment. Journal of Information Systems Education, 17(2), 113-117. Obtained from: http://jise.org/Volume17/n2/JISEv17n2p113.pdf

Prabu, P. & Duraisamy, S. (2015). Impact of Pair Programming for Effective Software Development Process. International Journal of Applied Engineering Research, 10(8), 18969-18986.

Pressman, Robert (2014). Ingeniería del Software: un enfoque práctico (8th ed.). México, D.F.: McGrawHill.

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. doi: 10.1007/s10664-012-9238-

Sánchez-Montoya, Rafael (1995). Ordenador y discapacidad. Madrid: CEPE.

Smith, M., Giugliano, A., & DeOrio, A. (2018). Long Term Effects of Pair Programming. IEEE Transactions on Education, 61(3), 1-8. doi:10.1109/TE.2017.277302

Sommerville, Ian (2015). Software Engineering. Estados Unidos: Pearson.

Sun, W., Marakas, G. & Aguirre-Urreta, M. (2016). The effectiveness of pair programming. IEEE Software, 33(49), 72-79. doi: 10.1109/MS.2015.106

Tsompanoudi, D., Satratzemi, M. & Xinogalos, S. (2016). Evaluating the Effects of Scripted Distributed Pair Programming on Student Performance and Participation. IEEE Transactions on Education, 59(1), 24-31. doi:10.1109/TE.2015.2419192

UAT. (2018). Blackboard UAT. Obtained from https://campusenlinea-uat.blackboard.com/

von Wangenheim, C., Savi, R. & Ferreti Borgatto, A. (2013). SCRUMIA - An educational game for teaching SCRUM in computing courses. The Journal of Systems and Software, 86(10), 2675-2687. doi:10.1016/j.jss.2013.05.030

Wagner, William (2014). Using IBM SPSS Statistics for Research Methods and Social Science Statistics. United States: SAGE Publications.

Werner, L., Hanks, B. & McDowell, C. (2005). Want to Increase Retention of your Female Students?. Computing Research News, 17(2).

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case For Pair Programming. IEEE Software, 17(4), 19-25. doi:10.1109/52.854064

Zacharis, Nick (2011). Measuring the Effects of Virtual Pair Programming in an Introductory Programming Java Course. IEEE Transactions on Education, 54(1), 168-170. doi:10.1109/TE.2010.2048328

Zikmund, W., Barry, B., Carr, J. & Griffin, M. (2013). Business Research Methods. Mason, Ohio, United States: Cengage Learning.