Solo, pair or Mob programming: Which should be used in university?

Programación individual, por pares o colectiva: ¿cuál conviene utilizar en la universidad?

http://dx.doi.org/10.32870/Ap.v12n1.1791

Ramón Ventura Roque Hernández*
Sergio Armando Guerra Moya**
Adán López Mendoza***

ABSTRACT

Keywords
Software development; university education; teaching technology; teaching strategies; programming

The aim of this research is to compare individual, pair and Mob programming in university programming courses taking into account the perceptions of the students. 24 students participated in a course of Intermediate programming with Visual Basic. Net. They worked with the three modalities in the development of software projects during regular class sessions. A sequential explanatory mixed research design was used. For the quantitative component, questionnaires were administered. For the qualitative section, interviews were conducted. The results showed that students prefer pair programming because they perceive it as a midpoint between programming alone and doing it with a large group. Solo programming may cause stress and intellectual blockage, and doing so with too many people at the same time may generate distraction and imbalance of work among the participants. A limitation of the study is the sample size. However, the work makes quantitative and qualitative contributions in an area of knowledge little explored in formal literature. The findings suggest promoting work in pairs in university programming courses, as it is easily implemented with few resources and good results.

RESUMEN

Palabras clave
Desarrollo de software; educación universitaria; enseñanza de la tecnología; estrategias de enseñanza; programación

El objetivo de esta investigación fue comparar la programación individual, por pares y colectiva a través de las percepciones de 24 estudiantes del curso universitario Programación intermedia con Visual Basic.Net. Los participantes fueron expuestos a las tres modalidades de trabajo en el desarrollo de proyectos de software durante sesiones regulares de clase. El diseño de investigación fue mixto explicativo secuencial. Para el componente cuantitativo se aplicaron cuestionarios, y para el apartado cualitativo se realizaron entrevistas. Los resultados mostraron que los alumnos prefieren la programación por pares porque la perciben como un punto medio entre no programar con nadie y hacerlo con un grupo numeroso. Programar de manera individual puede provocarles estrés y bloqueo intelectual, mientras que hacerlo con demasiadas personas al mismo tiempo les genera distracción y desbalance de trabajo. Una limitación del estudio es el tamaño de la muestra; sin embargo, hace aportaciones cuantitativas y cualitativas a un área de conocimiento poco explorada en la literatura formal. Los hallazgos sugieren promover el trabajo por parejas en los cursos universitarios de programación, pues se implementa con facilidad, con pocos recursos y buenos resultados.

Received: September 26, 2019
Accepted: December 11, 2019
Online Published: March 30, 2020

* PhD in Telematics Engineering, University of Vigo, Spain. PhD in Education, José Martí University of Latin America, Mexico. Research professor at Autonomous University of Tamaulipas, Mexico. ORCID: https://orcid.org/0000-0001-9727-2608
** PhD in Philosophy with a specialization in Administration Autonomous University of Nuevo Leon, Mexico. Research professor at Autonomous University of Nuevo Leon, Mexico. ORCID: https://orcid.org/0000-0002-3369-8527
*** PhD in International Education, Autonomous University of Tamaulipas, Mexico. Research professor at Autonomous University of Tamaulipas, Mexico. ORCID: https://orcid.org/0000-0003-4801-640X

INTRODUCTION

Software development is an activity that is to be performed supported by a methodological approach that would facilitate the organization of tasks, collaborative work and understanding of the requirements of the applications. There is no universally recognized approach to be the best for every setting, therefore the different ways to work ought to be evaluated to choose the most appropriate one in accordance with the needs of each project and team of persons.

In college programming courses it is customary to work individually to carry out exercises and class practice (Umapathy & Ritzhaupt, 2017). This is due to the physical setting of computer laboratories and the number of students who take these subjects. In some cases, the students share a single computer because of the lack of computer equipment availability or because they do joint tasks; however, it is easy to see, in these cases, that one person conducts the process indefinitely, while the other person watches, poorly makes an opinion, or adopts a passive role.

Karthiekheyan, Ahmed & Jayalakshmi (2018) explain that pair programing is a practice consisting of two persons who work in the same computer equipment and who exchange the keyboard and the mouse in regular intervals. Both persons also exchange the roles of leader-follower, or driver-navigator, between themselves, as expressed in pair programming terminology. On the other hand, Zuill (2015) defines collective programming (Mob) as three or more persons working on the same computer equipment, space and project. This type of work requires image amplification from the computer, which may be done on a large size display or on a projector, as described by the author.

Although the Mob programming term was first used at the beginning of the past decade, the study and application thereof are recent (Balijepally, Chaudhry & Sridhar, 2017). Some companies have employed this approach with good results; however, if the bibliography on this subject is scarce to the world of business, it is even more scarce for the educational setting. These two environments are different; for example, there is more freedom at companies to adapt workspaces as required. At universities, on the contrary, the traditional facilities of computer laboratories for teaching are used by many students; they are furthermore limited to a fixed space, which could make the implementation of Mob programming difficult as a learning strategy.

If individual work may be done in college pair programming courses, or under the Mob mode, the following questions arise: Which approach is most appropriate? What is the perception of students with respect to these three ways of work? What are the differences between these modes? In this article we present our experience upon the application of these three approaches in a group of college students with the purpose of comparing their perceptions and of providing answers to these questions. We hypothesized that there are differences among the perceptions of students about the three modes, and that collaborative approaches (pair and Mob) are preferred over individual programming.

This work is organized in four sections: the first one contains background information previously reported in bibliography, the second one describes the methodology steering our research, the third one shares the results and discussion, and the fourth one presents the conclusions, recommendations and future works.

BACKGROUND INFORMATION

Pair programming

Pair programming or by pairs was proposed as part of the agile approach titled "extreme programming" at the end of the 90s; since then, its popularity has increased and its study has been done from different angles of interest. Aottiwerch & Kokaew (2018), for example, consider that every pair of programmers ought to be carefully chosen for an effective job; for this reason, al algorithm was created to select the best partner from a person based on the following criteria: attitudes, programming competences and learning behaviors.

To the above, Poonam & Yasser (2018) add that human factors and personality characteristics also are important for successful projects when programming by pairs. During their research, they found that technical aspects of projects have been more extensively studied than human factors. As an experiment was conducted, the authors found a significant relationship among the personalities of programmers, their job location, whether local or remote, and the performance of the partners.

Another less studied aspect or pair programming, is the use of Integrated Development Environments (IDE). Gomez & Aguileta

(2018) made contributions to this area doing a research the results of which specify that, in simple projects, using IDE increases the number of defects to single programmers and pairs; however, in more complicated programs, partners have significantly less defects with IDE. These authors conclude that students ought to do their initial practice with a simple word processor and a compiler, however, they ought to incorporate the use of an IDE, especially if they work within the pair mode.

Sadath, Karim & Gill (2018) emphasize on the gap existing between teaching and software engineering in the classroom and in the current reality of the world of business. For this reason, they center their research in college education and propose a work framework gathering the best practices which have proven to be effective in the industry. The work framework includes pair programming as a fundamental practice to enhance the knowledge shared among participants.

Smith, Giuliano & DeOrio (2018) agree on the relevance of doing pair work in engineering and programming college studies, as this prepares students for real world activities that are not frequently done alone. These authors performed a research to determine the effect of pair programming in the academic performance of students in the long term. Their results show that this way of programming had a positive effect: people who work in pairs in introductory courses later obtained the highest grades in more advanced studies.

Lee *et al.* (2016) carried out a research to relate study times and academic performance of students who do pair programming. They found that a better performance of students is obtained who are at the risk of failing with more hours of independent study before pair programming in the lab. They conclude that individual training along with pair programing is beneficial to prevent low performance in these courses.

In accordance with Du *et al.* (2015), pair programming is a useful tool to improve communication among the students of each pair, as well as the understanding of academic topics, regardless of the grades of each member. These authors also found that pair work triggers new ideas. For their research, they used the C language with topics of control flow, functions, pointers and files. Their evaluations were based on interviews to students and on teachers' perception. On the other hand, Saltz & Shamshurin (2017) highlight the positive results of pair programming as it is applied to analyze data with R.

These authors performed a research with students organized by pairs and saw that the participants improved their communication levels; in addition, they wrote a better code in less time and obtained good perceptions on the results they obtained.

Swamidurai & Umphress (2015), on the other hand, argue that software development does not always have to be in the company of another person; therefore, they propose an approached called "inverted pair programing", which consists of two programmers who start by designing solutions together, but they separate during the implementation and then they gather together again to do tests. These authors validated their proposal by means of two experiments, which resulted in the fact that traditional pair programming may be costly and that inverted pair programming achieves greater or similar quality levels at a lower cost.

To Meyer (2018), pair programming is an interesting practice that ought to be used from time to time, especially in more complicated sections of the software under development. He says that there is no reason to impose it as a sole mode to create software and recommends that this should not be confused with tutorship or accompaniment, which is a different activity. In this sense, Meyer explains that agile methods are not a panacea, for there still are disadvantages and challenges to be faced. Likewise, he highlights the importance of actual measures and evaluations of these approaches so that we will not exaggerate or create false expectations.

Mob Programming

Mob programming, also known as mobbing, is now seen with interest in the development of software focused on businesses. Zuill (2015) is one of the pioneers who made known details on this approach, which he refers to as "a step beyond pair programming". Zuill, based on his own experience, proposes principles for Mob programming to work successfully, such as treating others with kindness, consideration and respect; applying driver-navigator roles, where the driver used the keyboard and the navigator expresses his/her idea and guides him/her to implement it in the program; rotating driver-navigator roles every fifteen minutes; using the telephone and email as if this was a single person, which implies having a single email account and to answer calls on the speaker in a single telephone extension. Similarly, he recommends the organization of retrospective meetings to reflect on what has been done well and on what could be improved.

Pyhäjärvi & Falco (2018), creators of the first material in a book format completely dedicated to Mob programming, explain that the person who is using the keyboard ought to not think, he/she should only let ideas, thoughts and reflections of the rest of the team to flow to capture and implement them in the task which is executed. Thus, people who are not using the keyboard also assume an active position.

Currently, people who have used Mob programming in business environments say that, although they went through adaptation stages, they have obtained benefits for working as teams of more than three persons; however, it is still not clear what they are, how they are obtained and reached, for the contributions mainly come from experiential narratives that are difficult to quantify.

Schartman (2014) says that his software development team adopted the Mob mode after being encouraged by a chat where it was said that he could be up to ten times more productive with this work approach. Notwithstanding, they were barely equally productive as they were with other approaches that were used before. Schartman makes a reflection on the importance of having specific measures on Mob programming so that expectations are germane to reality.

In the other hand, Lilienthal (2017) emphasizes on the lack of scientific experiments aimed to know learn about the usefulness of Mob programming. Balijepally *et al.* (2017) agree that, currently, there barely is any initial evidence from anyone who has started using Mob programming, and that this requires an empirical validation both from software engineers and academicians. On the other hand, they identify, through the work of Zuill (2015) the following benefits of Mob programming: it reduces administrative processes, it eliminates communication barriers of participants, it diminishes the number of decisions to be made for future situations; in addition, there is less waste, less quick batches in the code, less external interruptions, less policies, less extensive meetings, more continuous learning, more satisfaction of members of the team and better software quality. The following are possible risks that may affect the manner in which the team works: organizational culture, scarce acquaintance with the agile development, dominant personality, as well as physical and mental fatigue of the members.

Buchan & Pearl (2018) describe their experience in a software development team who worked with Mob programming for one month to create a product for the financial services sector. They found the following benefits in this mode: tasks were completed faster, the team had a better sense of belonging towards the code, the style of the design and codification was more consistent, using tools was more productive, people got to know the system they worked on more widely, trust in the code increased, the new members incorporated faster, and the estimated time required for the tasks was more accurate.

Buchan & Pearl (2018) reported the following risks and challenges of Mob programming: people might be reluctant to accept this work approach, they may also be encouraged at the beginning and then lose interest, the code is slowly generated when people begin programming in Mob, interpersonal relations are increased and team capacities may be altered to complete the job, some people become isolated and they find it difficult to communicate with their teammates, getting physical facilities and the necessary equipment might pose a problem.

Wilson (2015) reports the experience of his team when working with Mob programming and explains that the approach may be adapted to project needs. They, for example, evolved from a purely Mob manner of work to a hybrid one involving pair programming; they used the Mob approach to amend the code of critical importance, they found that not all the tasks demanded the attention of the whole team and realized that, to them, this approach was more useful in settings where the solution is unknown than in setting where the solution is known, but demands more time to implement it.

Kerney (2015) describes how, in spite of his strong personality, he was able to interact and work with his partners in Mob programming and, in doing so on a daily basis, some features of their character changed in a positive manner; for example, their listening capacity improved and their consideration for each other increased.

Arsenovski (2016) speaks of his experience in the development of collective software in an inherited project with many defects. He calls his particular work approach 'swarm', a term which refers to several programmers who cooperate in a common project. Arsenovski says that they used several patterns such as 'branch out', where a participant withdraws from the team, with the purpose of resolving a problem that affects the whole team and then he returns. On the other hand, Lilienthal (2017) makes a difference between Mob programming and Mob architecture. The latter refers to the creation of and improvement of the architecture of software systems; this is done by the whole team by using automated tools and an external navigator who inspects the source code, models architecture visits, requests refactoring and establishes priorities, always in the company of the development team in the role as observant.

Bohekhout (2016) used the Mob approach with good results for programming and designing flows of processes. He recognizes the benefits of this mode of work, he explains that it took him long to find the optimal configurations of time, space, resources and strategies; as he attained this, he successfully combined individual modes and Mob. Table 1 includes the most important findings identified in the literature we saw on pair programming and Mob programming.

Table 1. Summary of the main findings reported in the analyzed literature on pair programming and Mob programming

Pair programming	Mob programming
• Pair programming in introductory courses helps students get better grades in more advanced courses (Smith *et al.*, 2018) • Individual preparation along with pair programming is beneficial to avoid poor academic performance of students (Lee *et al.*, 2016)	• More scientific experiments toward knowing more about Mob programming are necessary (Lilienthal, 2017) • Empirical validation of Mob programming is required. (Balijepally *et al.*, 2017)
• Pair programming is a useful tool that enhances communication between students, comprehension of subjects and, moreover, triggers new ideas. (Du *et al.*, 2015) • Pair programming improves the levels of communication and reduces the time in which the code is written. (Saltz & Shamshurin, 2017) • The development of software does not have to be made between two people all the time. Single-person and pair activities can be alternated. (Swamidurai & Umphress, 2015) • Pair programming can be used occasionally, especially for the most complex parts of the software. (Meyer, 2018)	• Mob programming reduces administrative processes and long meetings; it promotes continuous learning, satisfaction in teammates and quality of the software. (Zuill, 2015) • With Mon programming people know more thoroughly the system they are working in. At the beginning, however, they move slowly and can show unwillingness, lose interest and become isolated. (Buchan & Pearl, 2018) • With Mob programming, the ability to listen and the consideration for others can be improved. (Kerney, 2015) • It may take a while to find the optimal configurations of time, space, resources and strategies to work with Mob programming. (Bohekhout, 2016)

METHODOLOGY

Purpose and research questions

The purpose of our research was to compare students' perceptions on the approaches of individual, by pairs and Mob programming, aimed to have empirically validated elements that enable valuing the feasibility of implementing any of these three approaches. Thus, it will be possible to establish future actions and strategies to facilitate that they be adopted in college courses. This is important because, currently, measures to determine comparisons are scarce in this context, especially, regarding Mob programming.

Guiding questions of the survey were: which of these three approaches (individually, by pairs, Mob) is mostly appropriate in the context of college teaching? What is the perception of students about them?; and, what are the differences between these modes?

Research design

For this research we used a sequentially explanatory design (Hernandez, 2014), which consists in applying a quantitative approach, to analyze data and obtain conclusions so that a qualitative approach may be later employed aimed to deepen on these results and achieve the interpretation of the whole analysis (see figure 1).

We also applied a questionnaire and a statistical data analysis for the quantitative approach, in addition to deep interviews for the qualitative approach.

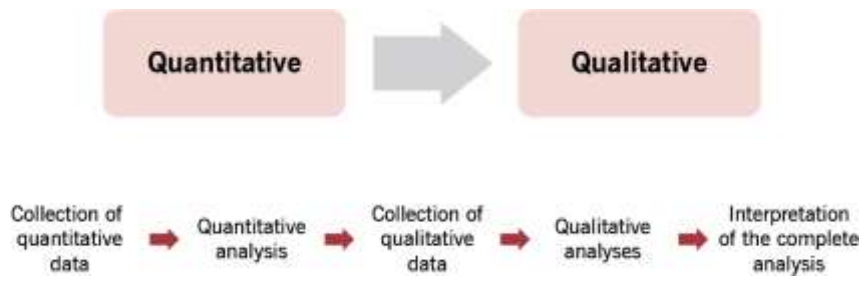
Quantitative
Qualitative
Collection of quantitative data
Quantitative analysis
Collection of qualitative data
Qualitative analyses
Interpretation of the complete analysis

Figure 1. Sequential explanatory design.

Participants

There were 24 students participating who were taking the subject of Intermediate Programming in the third semester of the undergraduate program in Information Technologies. The age range was between 19 and 25 years, with a mean of 20.70 and a standard deviation of 2.01. Of these, 20 were males and four females. During the development of this work, the students attended regular lesson sessions and were not aware that his research was in progress; therefore, we did not give them any incentive or remuneration for their answers. All the participants had previously taken and passed the college courses: Fundamentals of computing and programming methodology and Basic programming. The former one was centered on the necessary logic to realize computer algorithms and the latter, in solving problems of basic programming structures using Visual Basic.Net language.

Setting

This work was done at the computer lab students usually go for their programming lessons. At this place, there is access to 30 computers with an Intel i5 processor, 21-inch monitors and a 1 tb hard drive. The Windows 8 operation system is installed on these machines as well as the Visual Studio integrated development environment using Visual Basic.NET language.

Procedure

Every participant had individual work experience, as this was the usual way of doing their practices until then; however, they were not aware of the pair programming and Mob programming approaches. For purposes of this research, first off, at a class session, we asked participants to do individual programming. In a different session, we asked to do pair work and, in a third session, with Mob programming. Finally, we invited them to answer the questionnaire only based on the experience of these three work instances.

Each session was two hours long. Time was distributed as follows: for the first fifteen minutes we waited for all the students to arrive at the lab; then, we called the roll and explained the rules of working with each methodology employed; then, we presented the program they were to work on and randomly assigned students to the computers in the lab. In the case of pair and Mob modes, the teams were also created randomly. All of these activities were done un twenty minutes. The students had 85 minutes to develop the proposed program. During the research, none of the students was to make contact with anyone outside his/her team. Neither were they allowed to copy the work of another person; however, they could consult information using Internet.

Individual programming session

In the individual mode, each student was assigned to a computer. The program requested used a software-oriented architecture for three-layer objects and implemented a list of registries stored in a database using a table.

Pair programming session

Each pair of students used a single computer, where each participant had to use the keyboard for five minutes and then assign it to his partner. The researchers measured the time and told them when to change use of the keyboard. The program requested used a three-layer object-oriented architecture and implemented several selective lists of stored registries in a database using several tables.

Mob programming session

The students were randomly assigned to four teams of six persons who were located in different workspaces, however, within the same practice lab area they attend class. Each team was assigned a single computer with the features described in the "setting" section of this article. In addition, six chairs, a projector, a projection surface, and a workspace were assigned. Each team was self-organized to work. They were free to do this, provided, however, that the team worked together, in a respectful manner and consistent with the principles of Mob programming. The program requested was a graphic interface to carry out diverse registry consultations stored in a database and used a three-layer software-oriented architecture.

Quantitative approach method

We designed a questionnaire with the questions shown on table 2 and we gave it to the students, we asked them to evaluate each aspect for each work mode by using a one-to-ten scale, where one is the lowest score, which represents "bad" or "scarce", and ten, the highest, which means "excellent" or "a lot". We decided to use this scale because students find it simple to use it when they do evaluations, as they related it to school grades they get at the university. Furthermore, in a previous research conducted by our team, some of the students found it difficult to evaluate and express their perceptions using a one-to-five scale.

Table 2. Evaluations requested from participants of individual, by pairs and Mob work

Question	Aspect evaluated
1	Overall grade of the methodology
2	Ability to understand the methodology
3	Ability to implement the methodology
4	Ability to adapt the methodology
5	Ability to work with the methodology
6	Experience of working with the methodology
7	Detection of errors in the program
8	Quickness to finish the program
9	Quality of the development process
10	Ability to communicate
11	Motivation to work on the project
12	Organization for work
13	Level of confidence in the success of the project
14	Level of satisfaction with the work done

Every answer was captured by using the SPSS package, where a review and data clearing process was performed. We did not find any lost or atypical values. Later on, the answers were changed as specified in table 3. These values in the new one-to-five scale were used for all the procedures and analyses done in this research.

Table 3. Answers provided by students and categorized values used in the quantitative analysis

Former values (answers from the students)	New values (new values transformed)
1,2	1
3,4	2
5,6	3
7,8	4
9,10	5

Afterwards, a factor analysis was done taking questions 2 to 14 into consideration. We found three dimensions which we labeled with representative names. Then, we calculated Cronbach's alpha for each of them. In this process we also had the participation of each question in its respective dimension. Later, we calculated the weighted value for each dimension; for that purpose, we multiplied each categorized answer of the students by the percentage corresponding to the participation of each question in said dimension.

Finally, we conducted Friedman's tests to find significant statistical differences between the values of each of the dimensions for the three methodological approaches in the study. In addition, we sought differences between the global score assigned by students to each of the three methodological approaches by using Friedman's test. Figure 2 shows a global scheme of the steps included in the analysis of quantitative data.

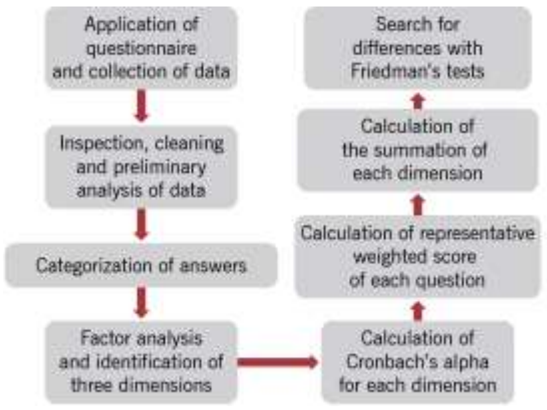
Application of questionnaire and collection of data
Inspection, cleaning and preliminary analysis of data
Categorization of answers
Factor analysis and identification of three dimensions
Calculation of Cronbach's alpha for each dimension
Calculation of representative weighted score of each question
Calculation of the summation of each dimension
Search for differences with Friedman's tests

Figure 2. Quantitative analysis schemes done in this research.

Method for the qualitative focus

For the qualitative component in this research we took the quantitative results obtained as reference: pair programming had the highest scores, whereas Mob programming and the individual mode got the lowest scores. For this reason, we did interviews with open questions oriented to learn more about the reasons of these findings. We formulated the following questions: What are the advantaged you perceive of pair programming over Mob programming and individual programming? What are the main problems you found when working with Mob programming? What are the main problems you found when working with individual programming? What is your opinion on the possibility of implementing pair programming or Mob programing as a daily practice in college courses?

Of the 24 students who partook in the quantitative phase, five were available and willing to participate in the qualitative phase interviews. Although only two provided extensive and rich in content interviews, whereas the other three provided ideas in short phrases which were contained in the answers of the two students present this article.

In this sense, Hernandez (2014) says that in the case of qualitative approaches, the sample need not be statistically representative of the population in the study, and that the size thereof is determined by three factors: the operational capacity and data recollection, accessibility of study units and saturation of categories, which implies the number of cases that allow answering research questions. Thus, in qualitative studies it is also possible to reformulate the sample. This means that unforeseen cases at the beginning may be added in the process, or rather, others may be excluded that were foreseen.

RESULTS

Quantitative results

In order to find the three dimensions shown in table 4, we performed an exploratory factor analysis by using the maximum plausibility extraction method based on eigenvalues greater than 1. The rotation method was Promax with Kappa = 4. We obtained a KMO mean of 0.704; Bartlett's sphericity test showed the values of Chi^2 = 364.60. gl = 78 and sig.= 0. Total explained variance with three factors was 80.32%. Confirmatory factor analysis to exactly find three factors had the same results. Cronbach's alpha calculations and the contribution percentages of each variable to its dimension are included in tables 5, 6 and 7.

Table 4. Dimensions found in the questionnaire

Dimension	Question	Aspect to evaluate
Aspects of process and team management	2	Ability to understand the methodology
	4	Ability to adapt the methodology
	9	Quality of the development process
	10	Ability to communicate
	12	Organization for work
	13	Level of confidence in the success of the project
Aspects of implementation	3	Ability to implement the methodology

	5	Ability to work with the methodology
	6	Experience of working with the methodology
	11	Motivation to work on the project
	14	Level of satisfaction with the work done
Aspects of debugging	7	Detection of errors in the program
	8	Quickness to finish the program

Table 5. Participation percentage for each question of dimension 1: process and team management aspects (Cronbach's alpha = 0.94)

Id	Aspect to evaluate	Overall corrected correlation of the elements	Participation of each question in this dimension
2	Ability to understand the methodology	0.846	0.170
4	Ability to adapt the methodology	0.866	0.174
9	Quality of the development process	0.795	0.160
10	Ability to communicate	0.872	0.175
12	Organization for work	0.850	0.171
13	Level of confidence in the success of the project	0.732	0.147

Table 6. Participation percentage for each question of dimension 2: implementation aspects (Cronbach's alpha = 0.925)

Id	Aspect to evaluate	Overall corrected correlation of the elements	Participation of each question in this dimension
3	Ability to implement the methodology	0.799	0.199
5	Ability to work with the methodology	0.792	0.197
6	Experience of working with the methodology	0.831	0.207
11	Motivation to work on the project	0.775	0.193
14	Level of satisfaction with the work done	0.823	0.205

Table 7. Participation percentage for each question of dimension 3: debugging aspects (Cronbach's alpha = 0.939)

Id	Aspect to evaluate	Overall corrected correlation of the elements	Participation of each question in this dimension
7	Detection of errors in the program	0.889	0.500
8	Quickness to finish the program	0.889	0.500

Friedman's test applied to establish differences in the first dimension (process and team management aspects) was significant (PValue= 0.005, Chi-square = 10.564, n = 24, gl = 2); the highest scores were for pair programming (average range = 2.46) and the lowest, for Mob programming (average range = 1.63). Although we found no significant statistical differences in Friedman's test, in the second dimension (implementation aspects) (PValue = 0.108, Chi-square = 4.447, n = 24, gl = 2), we noticed asymmetric tendencies of pair programming to obtain the highest scores (average range = 2.31) and Mob programming for the lowest scores (average range = 1.83).

In the third dimension (bugging aspects) we did recognize significant differences with Friedman's test (PValue = 0.031, Chi-square = 6.969, n = 24, gl = 2); the highest scores were for pair programming (average range = 2.23) and the lowest, for individual programming (average range = 1-65). For the global score which students gave to methodologies we noticed no significant differences with a level of confidence of 95% as reference, but it may be considered significant with a level of confidence of 90% as reference, as this is an initial research. The results from this comparison with Friedman's test were: PValue = 0.062, Chi-square = 5.548, gl = 2. The highest scores were for pair programming (average range = 2.29) and the lowest, for Mob programming (average range = 1.75).

Qualitative results

The interview for the qualitative section disclosed that students like to do pair programming because they see it as a middle point between not programming with anyone and doing it with a large group. Individual programming may give rise to mistakes, tension, mental block, and uneasiness because of the responsibility of the work; on the other hand, programming with many persons at the same time may give rise to disruption, stress, distraction and imbalance among the participants' work. Table 8 contains a summary of the answers provided by participants.

Table 8. Summary of interviews performed

Question	Summary of answers of participant 1 (male, 20 years old, third semester student)	Summary of answers of participant 2 (male, 21 years old, third semester student)
• What advantages do you see about pair programming compared to Mob and individual programming?	• Having a partner to solve problems • Work and communication are under more control.	• Shared responsibility • Pair programming is a middle point between being completely alone and being surrounded by a lot of

	• Work is better organized. • Support from another person during the whole process	people. Like this, the disadvantages of working alone and in Mob are overcome • More ability to reach agreements for work
• What are the main problems you encountered working with Mob programming?	• Working with so many people at the time generates more stress • Noise affects concentration • A more private workspace is necessary • The heavier work is done by a few people • Not everyone explains their work to the rest	• Lack of organization for working and decision making • Not everyone collaborates with the same interest • Different levels of skills and knowledge • Immature attitudes from some participants.
• What are the main problems you encountered working individually?	• Sometimes there is tension or a working block, especially if you don't know how to solve the problems that come up. You must ask for help from another classmate or the teacher.	• You feel a complete responsibility of the project. • More mistakes are made and it takes longer to find them and correct them
• What is your opinion on the possibility of implementing pair programming or Mob programming as an everyday practice in university courses?	• Implementing pair programming would be alright • It would be nice if students could pick a work partner • Mob programming could be implemented in more advanced courses with students of higher semesters who know each other better and are more focused in their work	• Implementing Mob modality wouldn't be good if there aren't more private workspaces and enough projectors for each team. Maybe students would need training to work in teams • Pair programming would be alright but there would be a need to make sure each couple is compatible to work in the same project.

DISCUSSION

The results of this work show that pair programming could be the most appropriate approach as a didactic strategy, as it is well accepted by students and, furthermore, implementing it does not require of additional equipment or adaptations to the workspace. Students could organize in teams of two persons to use the computers as they are installed and distributed in the computer labs.

Students' perception favors pair programming over individual and Mob programming, as they consider that it is good to share responsibilities, but not with so many people at the same time, because things could easily go out of control. In this sense, in Mob programming, the attitude and behavior of some students affect aspects such as communication, understanding of the problem and organizing tasks. On the other hand, regarding individual programming, students feel that they make more mistakes and that it takes them longer to find them and to correct them.

The position of the authors of this work agree with those of Sadath *et al.* (2018), who state that teaching software engineering in college classrooms is outdated from the reality currently experienced by companies. For this reason, it is important to align thematic contents of courses and, at the same time, to promote significant learning by means of dynamic ways to develop software that are directly applicable to companies. Therefore, it is necessary to perform research on the approaches used in the classroom, for students to learn how to develop software. In the specific case of Mob programing, we agree with Sharman (2014), Lilenthal (2017) and Balijepally *et al.* (2017) on the need to have more guiding measures and evaluations.

Our results are similar to those of Du *et al.* (2015) and Saltz & Shamshurin (2017). Just like them, we found that pair programming improves communication among students. They are also analogous to those of Gomez & Aguileta (2018), because the participants believed it was easier to find and correct mistakes with pair programming, aided by a comprehensive development environment. We also agree with Swamidurai & Umphress (2015) and Meyer (2018) in the fact that students do not have to do pair programming all the time. An evolutionary hybrid approach, which is the product of continuous adaptation of needs, as mentioned by Wilson (2015), could be beneficial for the academic achievement of students.

For the correct interpretation of our results the reader ought to consider that this study was done with a small sample of a college course at a beginner-intermediate level. Further research is required to test these findings and to deepen on them.

CONCLUSIONS AND RECOMMENDATIONS

The students who experienced the three work modes studied in this article showed their preference for pair programming. Using Mob programming was not reported to be favorable in these aspects because of the multiple interaction of participants and of the lack of consensus in the organization and of making decisions. Individual programming, on the contrary, restricts continuous interaction with other people and this may be a barrier if participants are not sure how to solve the problems raised. We recommend that pair work be promoted for programming college courses. This practice may be easily implemented at facilities existing in computer laboratories at the universities.

It is necessary to continue doing research to learn what the most proper manner is to teach programming. In this sense, application of collaborative programming ought to be done in greater depth regarding its diverse modes and beginner, intermediate and advanced students are to be involved.

REFERENCES

Aottiwerch, N. y Kokaew, U. (2018). The Analysis of Matching Learners in Pair Programming Using K-Means, in *5th International Conference on Industrial Engineering and Applications* (362-366). https://doi.org/10.1109/IEA.2018.8387125

Arsenovski, D. (2016). *Swarm: Beyond pair, beyond Scrum*. Agile Alliance.

Balijepally, V.; Chaudhry, S. y Sridhar , N. (2017). Mob Programming – A Promising Innovation in the Agile Toolkit, in *Twenty-third Americas Conference on Information Systems*, Boston, 2017 - Systems Analysis and Design (SIGSAND), 1-9. Boston, Estados Unidos: AIS Electronic Library.

Bohekhout, K. (2016). Mob Programming: Find Fun Faster. Lecture Notes in Business Information *Processing*, 185-192. https://doi.org/10.1007/978-3-319-33515-5_15

Buchan, J. y Pearl, M. (2018). Leveraging the Mob Mentality: An Experience Report on Mob Programming. E*ASE 2018 - Evaluation and Assessment in Software Engineering* (pp. 1-6). Christchurch, Nueva Zelanda: Editorial de la Universidad de Canterbury. https://doi.org/10.1145/3210459.3210482

Du, W.; Ozeki, M.; Nomiya, H.; Murata, K.; y Araki, M. (2015). Pair programming for enhancing communication in the fundamental C language exercise, in *2015 IEEE 39th Annual International Computers, Software & Applications Conference* (664-665). IEEE Computer Society. https://doi.org/10.1109/COMPSAC.2015.67

Gómez, O. y Aguileta, A. (2018). Influence on the use of an IDE as tool support in the pair programming: A controlled experiment, in *IEEE Latin America Transactions*, *16*(3), 948-956. https://doi.org/10.1109/TLA.2018.8358678

Hernández Sampieri, R. (2014). *Metodología de la investigación*. Ciudad de México: McGrawHill.

Kerney, R. (2015). *Mob Programming - My first team*. Estados Unidos: Agile Alliance.

Lee, L.-K.; Au, O.; So, R. y Nga-Inn, W. (2016). Being Well-Prepared for Regular Pair-Programming Helps At-Risk Students, in *2016 International Symposium on Educational Technology* (65-68). https://doi.org/10.1109/ISET.2016.22

Lilienthal, C. (2017). From Pair Programming to Mob Programming to Mob Architecting, in *International Conference on Software Quality - SWQD2017 - Software, Quality, Complexity and Challenges of Software* (3-12). Vienna, Austria: Springer. https://doi.org/10.1007/978-3-319-49421-0_1

Meyer, B. (2018). Making Sense of Agile Methods. *IEEE Software*, *35*(2), 91-94. https://doi.org/10.1109/MS.2018.1661325

Poonam, R. y Yasser, C. (2018). An Experimental Study to Investigate Personality Traits on Pair Programming Efficiency in Extreme Programming, in *2018 5th International Conference on Industrial Engineering and Applications* (95-99). https://doi.org/10.1109/IEA.2018.8387077

Pyhäjärvi, M. y Falco, L. (2018). *The Mob Programming Guidebook*. n/d: LeanPub.

Sadath, L.; Karim, K. y Gill, S. (2018). Extreme Programming Implementation in Academia for Software Engineering Sustainability, in *2018 Advances in Science and Engineering Technology International Conferences (ASET)* (1-6). Abu Dhabi. https://doi.org/10.1109/ICASET.2018.8376925

Saltz, J. y Shamshurin, I. (2017). Does Pair Programming work in a Data Science Context? An Initial Case Study, in *2017 IEEE International Conference on Big Data (BIGDATA)* (2348-2354). https://doi.org/10.1109/BigData.2017.8258189

Schartman, M. (2014). *My Experience with Mob Programming*. Appfolio Engineering.

Smith, M.; Giugliano, A. y DeOrio, A. (2018). Long Term Effects of Pair Programming. *IEEE Transactions on Education*, *61*(3), 1-8. https://doi.org/10.1109/TE.2017.2773024

Swamidurai, R. y Umphress, D. (2015). Inverted Pair Programming, in *Proceedings of the IEEE SoutheastCon 2015* (1-6). Fort Lauderdale, Florida: IEEE. https://doi.org/10.1109/SECON.2015.7133010

Wilson, A. (2015). *Mob Programming - What works, what doesn´t*. Helsinki, Finlandia: Springer Link. https://doi.org/10.1007/978-3-319-18612-2 33

Zuill, W. (2015). *Mob Programming - A Whole Team Approach*. https://www.agilealliance.org/wp-content/uploads/2015/12/ExperienceReport.2014.Zuill_.pdf

HOW TO CITE

Roque Hernández, Ramón Ventura; Guerra Moya, Sergio Armando & López Mendoza, Adán. (2020). Solo, pair or Mob programming: Which should be used in university? *Apertura*, *12*(1), pp. 39-55. http://dx.doi.org/10.32870/Ap.v12n1.1791