Proposal for the Formation of Experimental Pair Programmers

Mauricio Dávila[1,*], Marisa Panizzi[1], Darío Rodríguez[2,3]

[1] Master's Degree Program in Information Systems. Universidad Tecnológica Nacional. Facultad Regional Buenos Aires, Castro Barros 91, (C1178AAA), C.A.B.A, Argentina.
[2] Engineering Group for Virtual Workstations and Information Systems Research Group. Department of Technological Productive Development. Universidad Nacional de Lanús, 29 de Septiembre 3901, (B1826GLC), Lanús, Buenos Aires, Argentina.
[3] Commission for Scientific Research - CIC. Calle 526 e/10 y 11, (B1906APP), La Plata, Buenos Aires, Argentina.
davilamr.80@gmail.com, marisapanizzi@outlook.com, dariorodriguez1977@gmail.com

Abstract. This work proposes a protocol aimed to form homogeneous experimental pairs of programmers to ensure that two individuals with the same characteristics are undistinguishable in terms of their abilities as programmers. This protocol enables the measurement and evaluation of the characteristics of programming languages independently of the programmers' skills or the lack of them. A test case is presented so that it validates the protocol. To this end, the protocol will be applied to a group of C language programmers to show that the members of the experimental pairs formed do not show significant differences, in terms of quality and time, in the coding of a specification.

Keywords: experimentation in software engineering, protocol, experimental pair programmers, programming

1 Introduction

At present, there is a great diversity of programming languages and this often makes it difficult to select the language that better adapts to the needs of a given development. The selection of a programming language to find a solution entails multiple factors of analysis, many of which are subject to the generated source code and the time employed to generate it.

Using the systematic reviews method [1], a documentary research has been conducted on studies which discuss the metrics that can be established on the generated code with a given programming language [2][3][4] and [5]. When metrics are used to determine whether a language is the best choice compared to another one, it must not be overlooked that such metrics only focus on the resulting code and do not take into consideration the characteristics of the programmer who built the code. This may result in the sub classification or over classification of a language as a result

* Corresponding autor: Mauricio Dávila

of the skills or the lack of them of the programmers using it. This poses a problem when it comes to designing an experiment to determine the language to be used, since the group of programmers who use language A may have a higher level of knowledge than the group of programmers who use language B or vice versa, which would impact on the results of the experiment. One possible solution to this problem would be to have a group of programmers who can solve the same task in the languages that are being evaluated.

Beyond the difficulty in gathering this group of individuals with the necessary knowledge in each language to be evaluated, this solution poses some additional difficulties which arise when designing experiments.

Both Juristo and Moreno [6] and Wohlin [7] argue that the experiments performed in the field of software engineering are strongly influenced by the characteristics of the individuals, like in other sciences, generally known as social sciences. Juristo and Moreno [6] enumerate some issues related to social factors and the specific characteristics of the software development that must be taken into consideration when designing experiments.

- Learning effect: If an individual should solve the same issue applying different programming languages, it is highly probable that they will learn more and more about the issue and that the final result will be better than the first one, simply due to the fact that the individual knows more about the issue rather than due to the fact that the programming language is better.
- Boredom effect: the individuals get bored or tired of the experiment and put less effort and interest as time passes by.
- Enthusiasm effect: It may happen that the individuals who use an old programming language are not motivated to do a good job while those who use a new programming language are.
- Experience effect: when performing an experiment that involves programmers, it is to be expected that there will be different levels of both knowledge and skill about the programming language used.
- Unconscious formalization: it happens when the same individual uses two or more programming languages with different levels of definition or formality.
- Setting effect: the emotional state of participating individuals is closely related to their performance.

A set of actions to consider so as to control the abovementioned effects is described below:

- Learning effect: do not use the same group of individuals to work on a development using more than one programming language.
- Boredom effect: motivate the individuals who perform the experiment in the same way regardless of the group they belong to.
- Enthusiasm effect: do not inform the individuals about the hypotheses or objectives of the experiment.
- Experience effect: to control this effect, experimental pair programmers who are undistinguishable in terms of their knowledge and skills regarding the programming language will be formed.

- Unconscious formalization: aspects regarding the level of knowledge of each individual should be considered when it comes to the programming language employed when forming the experimental pair.
- Setting effect: it must be taken into consideration that all the steps of the experiment should be carried out under the same conditions.

In the field of software engineering, it is common to face a need to perform experiments with a reduced group of people who apply different treatments on the objects of study. Taking into account that comparing experimental units within homogeneous pairs not only increases the accuracy of the analysis but also allows us to control most of the undesired effects [6] when performing an experiment, this work is based on the hypothesis that it is possible to form homogeneous experimental pairs of programmers and that, therefore, two subjects with the same characteristics are undistinguishable in terms of their abilities as programmers. This hypothesis leads to the following research questions: is it possible to form undistinguishable experimental pairs in terms of abilities and skills as programmers? If so, do they require the same amount of time to solve the same task with the same programming language?

In (Section 2), a protocol for the formation of experimental pairs programmers is proposed. In (Section 3), the validation of the protocol is performed through a pilot test; and in (Section 4), conclusions and future lines of research are presented.

2 Proposal for the Formation of Experimental Pair Programmers

When it comes to deciding how to form homogeneous pairs of programmers, the authors adhere to Campbell [8], who argues that many factors may indirectly affect the performance of an individual, but only three are direct determinants of performance: *knowledge, skill* and *motivation.* For this reason, a protocol will be designed with the aim to form experimental pairs of programmers who are homogeneous in terms of both level of knowledge and skills, and it will be assumed that the participants to be characterized do not show significant differences regarding motivation.

The first step consists in identifying the methods used to categorize programmers in other experimental research studies (Section 2.1). Then, the guidelines considered for the design of the categorization instruments used in the experiment are described (Section 2.2). Lastly, (Section 2.3) presents a mechanism to form homogeneous experimental pairs of programmers based on the characterization made.

2.1 Programmers' experience

Like in most human activities, individual performance in software development varies considerably from one person to another and mechanisms should be articulated so that such variations do not affect the results of the study. Feigenspan et al. [9] conducted a documentary work which analyzed 161 publications and found nine ways used by researchers to determine a programmer's experience. These authors define experience as the amount of knowledge acquired regarding the development of programs.

- Years: in forty-seven works, the number of years a programmer had been programming in general or in a company or in a certain language was used to determine their experience in the programming field.
- Education: In nineteen of the articles reviewed, participants' education was used to indicate their experience, which included information about the level of education obtained (pre-university, undergraduate, graduate, etc.) or the grades obtained in their course of studies.
- Self-estimation: In twelve works, participants were asked to estimate their own experience.
- Specific survey: In nine works, the authors applied a survey to evaluate programming experience.
- Size: the size of the programs written by the participants was used as an indicator in six articles.
- Exam: In three works, an exam on programming was administered to evaluate the experience of the participants.
- Supervisor: In two works, in which professional programmers acted as participants, a supervisor was in charge of estimating their experience.
- Not specified: authors often argue that programming experience was estimated but they did not specify how. That was the case in thirty-nine works.
- Not controlled: programming experience was not mentioned at all in forty-five works, which compromises the validity of the corresponding experiments.

2.2 Characterization of programmers

It is necessary to develop a characterization method that is not based on the perception that each individual has on their own skill as a programmer since less competent people tend to overestimate their skills because they do not have enough knowledge to recognize their own limitations and it is also common for more prepared people to tend to underestimate their achievements and competences [12]. This characterization is aimed at establishing a set of the programmer's abilities in order to find programming pairs that may be considered homogeneous. The purpose of the characterization is to ensure that two individuals with the same characteristics are undistinguishable in terms of their abilities as programmers. To this purpose, it was decided that a broad set of skills of the programmer would be analyzed. The authors agree on the idea that the elaboration of a characterization based only on few criteria may result in serious errors. Some characteristics of the programmers are not related to the programming language and others are dependent upon it. For this reason, guidelines to develop two characterization instruments will be set, one disregarding the programming language (in Spanish, CILP) and another taking into account the programming language (in Spanish, CDLP). There exists a large number of measures to capture attributes of software processes and products which have traditionally been performed by relying on the experts' proficiency, and this situation has frequently led to a certain degree of inaccuracy in the definitions, properties and assumptions of the measurements, making the use of measurements difficult, their interpretation dangerous and the results of many validation studies contradictory [10]. For the development of characterization tools, the general procedure for the design of a measurement instrument proposed by Sampiere [11] and the method for the

definition of valid measurements proposed by Genero, Cruz-Lemus and Piattini [10] were taken into account, adapting such procedures to the needs of this work.

Due to issues related to the synthesis demanded by this publication, it is not possible to detail each of the steps followed to define either the instruments or the content of each of their dimensions.

Table 1 presents the content domains of the variable (dimensions), the indicators for each dimension and the nomenclature proposed for the dimensions of the instrument that will be used for the independent characterization of the programming language.

Table 2 shows the content domains of the variable (dimensions), the indicators for each dimension and the nomenclature proposed for the dimensions of the instrument that will be used to measure the dependent characterization of the programming language.

Regarding the decision on the type and format of the instrument and the context of its administration, the mixed procedure for data collection will be used, consisting of two questionnaires (CILP and CDLP) and an interview.

In order to minimize characterization errors, once the participant has answered the questionnaire, an individual interview will be conducted in which the interviewer will ask some questions so that the participant can justify their answers. If the participant provides a correct justification, the interviewer will consider it valid.

The context of administration will be a room with one computer for each programmer since the first phase, the one related to the questionnaire, is self-administered. Therefore, this can be done individually or simultaneously with a group of people. Then, the interview is conducted individually.

Table 1. Variable, dimensions, nomenclature for each dimension and their indicators.

Variable to be measured	Dimension	Nomenclature	Indicator
Characteristics which are independent of the programming language	*Education level*	***CILP1***	It refers to the education level attained by the participant, whether formally or informally.
	Experience	***CILP2***	It refers to the years of experience as a programmer and to the number of languages the participant declares to know.
	Comprehension of a specification	***CILP3***	Ability of the programmer to understand a simple specification and the time spent on it.
	Comprehension of a pseudocode	***CILP4***	Ability of the programmer to interpret the operation of pseudocode blocks and the time spent on it.
	Algorithmic ability	***CILP5***	Ability of the programmer to develop a solution with pseudocode and the time spent on developing such solution.

Table 2. Variable, dimensions, nomenclature for each dimension and their indicators.

Variable to be measured	Dimension	Nomenclature	Indicator
Characteristics which are dependent on the programming language	*Language chosen*	***CDLP1***	It refers to the programming language that the participant declares to handle more fluently and to the years of experience using it.
	Theoretical knowledge	***CDLP2***	It refers to the level of knowledge that the participant has on theoretical aspects of the programming language.
	Comprehension of the source code	***CDLP3***	Ability of the programmer to interpret the operation of pseudocode blocks and the time spent on it.

2.3 Mechanism to form experimental pairs

It is important to design a mechanism to ensure that the experimental pairs are formed by homogeneous subjects, which means that, according to their characterization, they should be undistinguishable or have negligible differences.

Criterion for the use of variables. Multiple variables emerge from the characterization procedure, some of them related to characteristics which are independent of the programming language and others related to characteristics dependent on the programmer's performance in a certain programming language.

Normalization. The normalization process consists in converting the values of the independent variables so that they are expressed in the range [0-10], regardless of their original scale. This step will ensure that none of the variables included in the distance calculation is weighted more heavily than the others.

Penalty for time spent. Each of the variables measured is accompanied by the time spent by the participant to complete the exercises related to such variable. In order to form experimental pairs which are undistinguishable not only in terms of knowledge but also in terms of time required to solve a task, a score penalty will be applied according to the time spent on the it. The score obtained will be reduced by 10% every 5 minutes.

Distance calculation. In a scenario where multiple variables need to be evaluated, all of them quantitative and whose values belong to the interval [0,10] after the normalization process, the criterion used to determine the distance between two

subjects must be defined. Since an n-dimensional space is being considered, the Euclidean distance calculation will be applied.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

Figure 1. Euclidean distance [15]

The calculation of the distance between participants will be performed, where the value of the module of the difference between variables of the same type does not exceed a threshold. This restriction will make it possible to establish the maximum distance tolerated, which shall not entail a significant difference in a single variable.

Algorithm for the selection of experimental pairs. The algorithm follows a sequential process in which it makes a decision at each step. It must select the minimum distance among the options available and then in the next step the algorithm has an identical problem, but with fewer options than in the previous step, and applies the same selection function to make the following decision [14].

3 Case for the Validation of the Protocol

A series of actions were taken to demonstrate the level of initial reliability and validity of the measurement instruments. The characteristics included in the formation of experimental pairs are: Comprehension of a Specification (CILP3), Comprehension of a Pseudocode (CILP4), Algorithmic Ability (CILP5), Theoretical Knowledge (CDLP2) and Comprehension of the Source Code (CDLP3).

To perform the initial pilot test, we recruited people of legal age who declared to know C programming language, and formed a group of 14 programmers. After each participant was characterized, they were asked to solve a task of medium complexity. Then, the protocol for the formation of experimental pairs of programmers was applied in order to determine whether the members of each experimental pair showed any significant differences in solving the task. If the experimental pairs formed by applying the protocol do not present significant differences when solving the same task using the same language, both the instruments and the protocol for the formation of experimental pairs can be deemed to have reached a stable version.

The results obtained from the characterization process are shown in Table 3. Table 4 presents the corresponding distance matrix. The intersections painted in black represent the subjects who should be ruled out since they have shown a distance over fifty percent in at least one of their dimensions. The experimental pairs obtained after applying the algorithm are highlighted in gray. Finally, the time differences between the programmers in each experimental pair are presented Table 5.

Table 3. Normalized results of the characterization.

Subject	CDLP2	CDLP3	CILP3	CILP4	CILP5
1	2.40	0.00	10.00	0.00	1.64
2	8.10	6.75	9.00	5.00	6.36
3	6.40	4.00	7.50	5.00	2.73
4	9.00	9.00	4.50	5.00	6.36
5	6.30	2.25	6.75	7.00	2.18
6	7.20	6.00	4.50	9.00	0.67
7	8.10	0.00	7.50	4.50	1.48
8	5.60	2.25	10.00	4.50	4.43
9	8.00	4.50	6.75	9.00	2.96
10	6.30	4.50	4.50	4.50	7.64
11	7.20	2.50	6.75	5.00	1.86
12	9.00	6.75	9.00	9.00	5.45
13	5.60	0.00	7.50	5.00	7.00
14	6.40	0.00	9.00	4.50	3.87

Table 4. Distance matrix

ID	2	4	5	7	9	11	12	14
2	-	5.11	7.07		6.14	6.65	4.20	
4	5.11	-			7.34		6.49	
5	7.07		-	3.95	3.54	2.23	6.88	4.38
7			3.95	-	6.58	2.83		9.20
9	6.14	7.34	3.54	6.58	-	4.67	4.17	7.00
11	6.65		2.23	2.83	4.67	-	7.44	4.03
12	4.20	6.49	6.88		4.17	7.44	-	
14			4.38	9.20	7.00	4.03		-

Table 5. Differences in time spent by each member of the experimental pair

Subjects		Difference	
		Time	%
2	4	6	7.06%
5	11	3	2.48%
7	14	5	4.76%
9	12	5	4.35%

With regard to the time spent to perform the characterization, the average time for the characterization that was independent of the programming language was 28 minutes; for the characterization dependent on the programming language, the average was 19 minutes; and for the interview, an average of 6 minutes was used for each participant. In Table 5, it can be observed that there are no significant

differences between the members of the experimental pairs of programmers in relation to the time spent on solving the same task.

4 Conclusions and Future Work

With the aim of proposing a protocol for the formation of experimental pairs of programmers, a document analysis was conducted on the benefits brought by this type of experiment to the software engineering sector. Two instruments were designed to characterize programmers. Using the data obtained from these characterization instruments, a procedure for the formation of experimental pairs was defined.

Finally, a validation case was implemented to verify whether the members of the experimental pairs obtained showed any differences in solving the same programming task.

It can be concluded that in terms of the formation of experimental pairs of programmers, the protocol worked satisfactorily and it is thus considered to have an acceptable level of reliability, validity and objectivity since it was consistent in the results provided.

The future lines of work identified are the need to: (1) apply the protocol to other programming languages and (2) use the protocol to form experimental pairs of subjects using different programming languages in order to determine whether a given programming language affects computing productivity.

References

1. Argimón, J. Métodos de Investigación Clínica y Epidemiológica. Elsevier España. 84-8174-709-2. (2004).
2. Halstead, M.. Elements of Software Science. Elsevier Science Inc., New York, NY, USA.(1977).
3. McCabe, T. J. A complexity measure. IEEE Transactions on software Engineering , 4, 308-320. 1976.
4. Riaz, M., Mendes, E., & Tempero, E. A systematic review of software maintainability prediction and metrics. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 367-377). IEEE Computer Society. 2009.
5. Rilling, J., y Klemola, T. Identifying Comprehension Bottlenecks Using Program Slicing and Cognitive Complexity Metrics. 10th IEEE Working Conference on Reverse Engineering. 10, pp. 115-125. Oregon, USA: IEEE. 2003.
6. Juristo, N., & Moreno, A. M. Basics of software engineering experimentation. Springer Science & Business Media. 2013
7. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. Experimentation in software engineering. Springer Science & Business Media. 2012.
8. Campbell, J. P., McCloy, R. A., Oppler, S. H., & Sager, C. E. A theory of performance. In Schmitt, N., Bormann, W.C. et al. (Eds.), Personnel selection in organizations, 35-70, San Francisco, Jossey-Bass. 1993.
9. Feigenspan, J., Kästner, C., Liebig, J., Apel, S., & Hanenberg, S. Measuring programming experience. In Program Comprehension (ICPC), IEEE 20th International Conference on (pp. 73-82). IEEE. 2012.

10. Genero, M., Cruz-Lemus, J. A., Piattini, M.: Métodos de Investigación en Ingeniería del Software. RaMa 2014.
11. Sampieri, R. H., Collado, C. F., & Lucio, P. B. Metodología de la investigación. México: McGraw-Hill. 2010.
12. Kruger, J., & Dunning, D. Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. Journal of personality and social psychology, 77(6), 1121. 1999.
13. Wilking, D., Schilli, D., & Kowalewski, S. Measuring the human factor with the rasch model. In Balancing agility and formalism in software engineering (pp. 157-168). Springer Berlin Heidelberg. 2008.
14. Soriano, M. A. Algoritmos Voraces. Facultat d'Informàtica, U.P.C. Consultado el 3 de Enero de 2017, disponible en http://www.cs.upc.edu/~mabad/ADA/curso0708/GREEDY.pdf. 2007.
15. Deza, Elena; Deza, Michel Marie. Encyclopedia of Distances. Springer. p. 94. 2009.