ISSN 2007-9737

Exploring the Influence of Fault Type, Fault Position and Gender on the Testing Process Using Code Review Technique

Raúl A. Aguilar[1,*], Julio C. Diaz[1], Antonio A. Aguileta[1],
Omar S. Gómez[2], Brenda L. Flores[3]

[1] Universidad Autónoma de Yucatán,
Facultad de Matemáticas,
Mexico

[2] Escuela Superior Politécnica de Chimborazo,
Facultad de Informática y Electrónica,
Ecuador

[3] Universidad Autónoma de Baja California,
Instituto de Ingeniería,
Mexico

avera@correo.uady.mx

Abstract. In the software process, one of the oldest tasks is coding. Despite the body of knowledge developed, there are still variables such as the type and position of the fault, which require a more in-depth and specific analysis. This study reports a controlled experiment, in which four exploratory analyses are developed, considering the factors: fault type, fault position and gender of the reviewer. The experiment was developed in an academic setting with 54 students from a Mexican institution. From the first two analyses it was concluded that types of faults influence the detection process, since the tester detects more faults by commission than by omission, but also detects more computational faults compared to interface, data and cosmetic. Regarding their position, faults in the first half of the code are more visible than those in the second half. Finally, regarding tester gender, no significant differences were observed in case of effectiveness.

Keywords. Code review technique, controlled experiment, fault position, fault type.

1 Introduction

Software Engineering (SE), over more than half a century, has managed to reach a consensus on a body of knowledge emanating from the interaction between theory and practice [1], maintaining a dynamic of constant improvement in terms of quality. In this context, Software evaluation has tried to answer two questions: how do we know if we have built the right software? and how do we know if we have built the software correctly? For this reason, validation and verification processes are constantly evolving [2], however, the human factor cannot be separated from either of these two processes, so research in this context requires empirical evidence for progress.

One of the evaluation processes that has been of special interest in the software process community, even before Empirical Software Engineering [3] existed as a research area in the context of SE, is code verification, that is, the process that ensures that the generated code has been written correctly according to the required functionality [4]. It should be noted that in this process the human component also has an influence.

The study here presented takes as reference the previous works of [4-7]; in these controlled experiments authors have used experimental designs considering as factors: the testing technique, the type of fault, as well as the type of program to be reviewed.

In previous works, the authors have explored factors such as: Belbin role [8], role type [9,10] and [11], fault position [9,10], as well as gender [9];

however, in the three cases cited, a homogeneous variety of fault types has been injected into the code, without performing an analysis of the efficiency in their discovery.

In this research we have run a controlled experiment reducing the testing technique and the type of program to parameters, thus isolating the factor: type of fault in a first pair of experiments and introduces the position of the fault as a factor in a third experiment.

2 Theoretical Backgrounds

The purpose of verification process is to ensure that the task performed has been carried out correctly. In the case of code review process, the intention is to obtain functional code. In this sense, it is convenient to make a distinction in a set of terms that are often used interchangeably. According to [2], it is worth to define the following terms:

– Error: human action that generates a fault or produces a failure.
– Fault: something that does not work well in a product (model, code, document, etc.).
– Failure: manifestation of a fault.
– Defect: general term referring either to error, fault or failure.

The software process considers the software testing process as one of its phases. In this process, depending on the level of testing (unit, integration, system and acceptance), it usually uses some pertinent strategy. In case of unit tests, it is possible to identify two testing strategies: static ones, which allow faults to be identified, and dynamic ones, which allow defects to be identified.

The code review process is carried out by stepwise abstraction. The software engineer, in his role as tester, identifies the main subprograms of the software, he determines their functions by comparing each one with the specifications provided [12]. This abstraction process involves reviewing the syntax of the language used and the semantics of the written code. In the code review, faults are identified in the code, so the correction of the code is carried out immediately, whereas in a dynamic process, failures are detected, and then an inference process is required to try to identify the reason for the failure and thus the possible fault associated to it.

Since failures are manifestations of faults, and ultimately the reviewing process is intended to identify faults, it is interesting to analyze this process based on the type of fault made by the programmer. Among the best-known classifications are the two proposed in [4]; the first classification (Classification A, CA) is characterized by the absence of necessary code (omission) or the presence of incorrect code (commission); the second classification (Classification B, CB) characterizes the type of fault (initialization, computation, control, interface, data and cosmetic).

3 Method for the Study

One of the research methodologies that allows generating empirical evidence commonly used in the context of Software Engineering is experimentation, in particular, experimentation in controlled environments [17]. This methodology allows comparing alternatives based on prior knowledge between relationships of independent and dependent variables around the phenomenon of the software process. Among the characteristic elements of controlled studies found in the literature, the use of groups of students as experimental subjects stands out; as stated in [18], a sample composed by students allows the researcher to obtain preliminary empirical evidence to confirm or refute a hypothesis, which could be later contrasted in industrial contexts.

3.1 Experimental Object

An Experimental Object (EO) is the piece or element used in the task to generate a value that is representative of the result of the experiment.

In the present study, the experimental object is the code reviewed by the subjects during the experiment. Based on the case study presented in [15], the code was modified and its functionality is to calculate the mean, median, mode and variance from a set of data included in an array of one hundred integers, each of these calculations are encoded in a function respectively; for the

Table 1. Faults injected into the experimental object

# Fault	CA	CB	# LOC
F1	Omission	Initialization	19
F2	Omission	Cosmetic	46
F3	Omission	Control	69
F4	Commission	Data	81
F5	Commission	Computation	83
F6	Commission	Interface	93
F7	Commission	Computation	116
F8	Commission	Initialization	138
F9	Omission	Cosmetic	152
F10	Omission	Control	158
F11	Commission	Interface	178
F12	Omission	Data	207

purposes of our study, twelve faults were injected, using as a reference the CA and CB classifications proposed in [4]. Faults were distributed along the 214 lines of code (LOC) as presented in Table 1. In order to collect information from the code review task, an instrument was designed in which the experimental subject would record each detected fault, its classification and justification, according to the two classifications used (CA and CB) as well as the number of the line of code where it was found.

4 Experimental Analysis I

The first controlled experiment described in this paper aims to explore the influence of the two alternatives in the first classification of faults (CA) on the task of software testing (unitary) by using the static technique called "code review".

4.1 Planning

The independent variable or factor considered in the study is the type of fault detected in the unit testing process; according to the first classification selected (CA), there are two fault alternatives: Omission and Commission; the independent variable is on a nominal scale.

The effectiveness of the tester was considered as a dependent variable, and for this purpose the number of faults correctly detected by the subject was used as a metric; although the dependent variable is on a numerical scale in which zero makes sense, the effort to detect two faults, even of the same type, is not exactly equal, therefore, the scale must be considered ordinal.

Therefore, the statistical hypotheses for the ctontrolled experiment are as follows:

– H_{10}: The median of the number of commission faults detected with the code review technique is equal to the median of the number of omission faults detected.

– H_{1a}: The median of the number of commission faults detected with the code review technique differs significantly from the median of the number of omission faults detected.

4.2 Execution

The convenience sample used for the experiment consisted of 54 experimental subjects who participated voluntarily. The experimental subjects were students enrolled in the course "Experimentation in Software Engineering" —a mandatory subject located in the eighth semester of the Software Engineering curricular program offered by the Autonomous University of Yucatan.

The controlled experiment was carried out in two sessions:

– S1. A teaching session (90 minutes) was used in which the researcher—in the role of teacher—presented content related to the verification process using the code review technique; in this session, the two most common fault types in the context of Software Engineering were described and exercises were carried out to identify these faults using the technique.

– S2. A double teaching session (180 minutes) was used to carry out the controlled experiment.

At the beginning of S2, the researcher gave instructions regarding the process of the controlled session, questions from the experimental subjects were answered, and then each subject was given: (a) a document with the description of the requirements that gave rise to the generated code, (b) the code to be verified, (c) a document with the

Table 2. Statistical summary of faults according to CA

Faults	#	Mean	Median	α
Commission	54	3.2037	3	1.5586
Omission	54	2.2222	2	1.0400

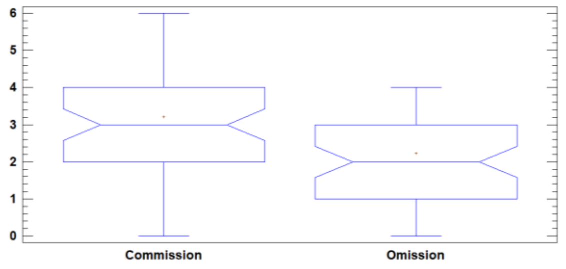

Fig. 1. Boxplot for the efficacy variable with CA

description of the two fault types, (d) Format for reporting the verification task.

4.3 Analysis

To analyze the collected data, a descriptive statistical analysis was first performed, and to analytically evaluate the defined statistical hypotheses, an inferential analysis was performed; both analyses were performed with the support of the statistical software Statgraphics.

According to the statistical summary in Table 2 it is possible to identify that the mean and median of the number of faults by commission detected is greater (by one unit) than the faults by omission, however, the variability of the faults by omission is less than those of commission.

Figure 1 shows that the 2nd quartile of commission faults coincides with the 3rd quartile of omission faults, i.e., 50% of the number of commission faults detected is greater than 75% of the number of omission faults detected; this gap of 25% of the data visually illustrates the differences between the two alternatives compared.

Because the two measurements are obtained from the same subject, they are paired (dependent) samples. Based on the characteristics of the scale and the samples, for the inferential analysis, the nonparametric Wilcoxon signed rank test will be used [16]. Since the P value for this test is less than 0.05 (0.000373948), it is not possible to accept the null hypothesis, therefore, with 95.0% confidence it is possible to affirm that the difference between the median of the effectiveness in detecting faults by commission and by omission is significant.

4.4 Results

According to the inferential analysis, the results show that there are significant differences between the effectiveness of the code reviewing process, between the detection of faults by commission and the detection of faults by omission; likewise, based on the descriptive analysis, we can complement the results by stating that with the code review technique, software engineers detect more faults of commission than of omission.

5 Experimental Analysis II

The second controlled experiment described in this article aims to explore the influence of the six alternatives in the second classification of faults (CB) on the task of software testing (unitary) by using the static technique called “code review”.

5.1 Planning

As in the first analysis, the independent variable or factor considered in the study is the type of fault detected in the unit testing process; according to the second classification selected (CB), there are six fault alternatives: initialization, computation, control, interface, data and cosmetic; the independent variable is on a nominal scale. Similarly, the dependent variable considered is the effectiveness of the tester, using as a metric the number of faults correctly detected by the subject. Therefore, the statistical hypotheses for the controlled experiment are as follows:

– H_{20}: The median number of faults detected with the code review technique is the same, regardless of the type of fault.

– H_{2a}: There are at least two types of faults whose median number of faults detected differs significantly.

Table 3. Statistical summary of faults according to CB

Faults	#	Mean	Median	α
Initialization	54	0.7407	1	0.6496
Computation	54	1.5925	2	0.6873
Control	54	1.2407	1	0.7507
Interface	54	0.9259	1	0.8655
Data	54	0.8518	1	0.7868
Cosmetic	54	1.0740	1	0.7590

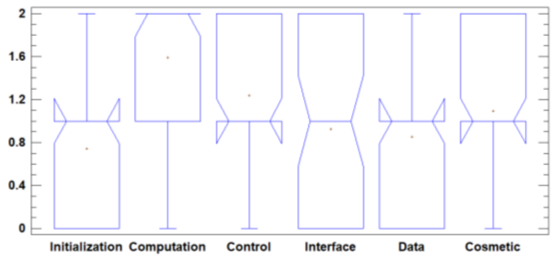

Fig. 2. Boxplot for the efficacy variable with CB

5.2 Execution

Since each experimental subject performed the fault detection task using the code review technique, and for each fault identified made a report justifying the fault based on the two classifications proposed in [4] the description of the execution for the second analysis is the one described in the first.

5.3 Analysis

To analyze the collected data, a descriptive statistical analysis was first performed. Table 3 presents a statistical summary of the second analysis in which it is possible to identify that the mean of the faults due to initialization and due to data are the lowest of the 6 types of faults, with the computation fault type having the highest mean, even presenting the highest median. In relation to the variability of the data, the standard deviation of the type due to initialization is the one with the lowest fluctuation, and that of interface is the one with the greatest variability.

On the other hand, the box diagram in Figure 2 illustrates four compartments differentiated in terms of the types of faults; initialization and data faults present similarities, control and cosmetic faults present a second type of similarity, computation faults, which present a bias towards higher values in one of the boxes, and interface faults, which present the greatest variability, so the boxes are distributed homogeneously.

The analytical evaluation through a hypothesis test used the non-parametric Friedman test, which is recommended when comparing more than two dependent samples [16]. Based on the empirical data collected in the experiment, the p value obtained is less than 0.05 (1.19004E-8), which indicates that at least two types of faults are significantly different; likewise, with the Bonferroni correction, 4 of the comparisons are statistically significant at the 95.0% confidence level.

5.4 Results

According to the descriptive and inferential analysis, the results show that there are significant differences between the effectiveness of the code review process, in particular, computational faults differ from interface, data and cosmetic faults.

6 Experimental Analysis III

The third experimental analysis aims to explore the influence of the position of the fault on the number of faults correctly detected by using the static technique called “code review”.

6.1 Planning

The factor for the third analysis is the position of the fault, and to do this, the alternatives are:

– Faults detected in the first half (FDFH).
– Faults detected in the second half (FDSH).

The dependent variable considered is the effectiveness of the tester, using as a metric the number of faults correctly detected by the subject. Therefore, the statistical hypotheses for the controlled experiment are as follows:

– H_{30}: The median number of FDFH with the code review technique is the same as the median number of FDSH.

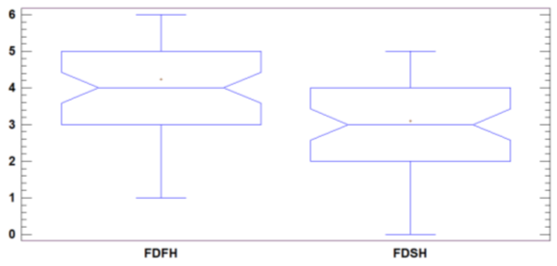

Fig. 3. Boxplot for the variable position of the fault

– H_{3a}: The median number of FDFH with the code review technique differs from the median number of FDSH.

6.2 Execution

The execution for the third analysis is the one described in the first, however, prior to the analysis of the data, it was necessary to count and record the faults correctly detected in the first 107 LOC, as well as those in the second half of the code.

6.3 Analysis

As analysis of the two previously reported studies, descriptive statistical analysis was first performed, and inferential analysis was performed to analytically evaluate the statistical hypotheses.

According to the statistical summary in Table 4 it is possible to identify that the mean and median of the FDFH is greater than the FDSH. Likewise, the standard deviation of the FDFH is less than that of the FDSH.

Figure 3 shows that the 2nd quartile of DHFH coincides with the 3rd quartile of FDSH, i.e., 50% of the number of faults detected in the first half is greater than 75% of the number of faults detected in the second half; this gap of 25% of the data visually illustrates the differences between the two alternatives compared.

Because the two measurements are obtained from the same subject, they are paired samples. Based on the characteristics of the scale for the inferential analysis, the nonparametric Wilcoxon signed rank test will be used [16].

Since the P value for this test is less than 0.05 (0.0000116705), it is not possible to accept the null hypothesis, therefore, with 95.0% confidence it is possible to affirm that the difference between the median of the effectiveness in detecting faults by commission and by omission is significant.

6.4 Results

Based on the statistical analysis carried out, it is possible to conclude, with 95% confidence, that the faults found in the first half of the code are more visible than those located in the second half, result that coincides with the results obtained in [9] and [10].

7 Experimental Analysis IV

The fourth experimental analysis aims to explore whether the gender of the software engineer has an influence on the efficiency in detecting faults using the “code review” technique.

7.1 Planning

The independent variable (factor) for the fourth analysis is the gender of the software engineer, the alternatives being: Male and Female.

The dependent variable, as in the previous analyses, is the effectiveness of the tester, using as a metric the number of faults correctly detected by the subject. The resulting statistical hypotheses are:

– H_{40}: The difference between the median number of faults correctly detected by male and female software engineers is zero.

– H_{4a}: The median number of faults correctly detected by male software engineers differs from the median number of faults correctly detected by female software engineers.

7.2 Execution

For the fourth analysis, it was necessary to identify, with the list of participating students, the gender of each of the 54 participants. Being a curricular program in the engineering area, naturally there were many fewer women than men, however, being an exploratory study, it was decided to proceed with the analysis.

Table 5. Statistical summary of faults according to fault position factor

Position	#	Mean	Median	α
Female	10	6.8000	8	2.0962
Male	44	7.4545	8	2.27683

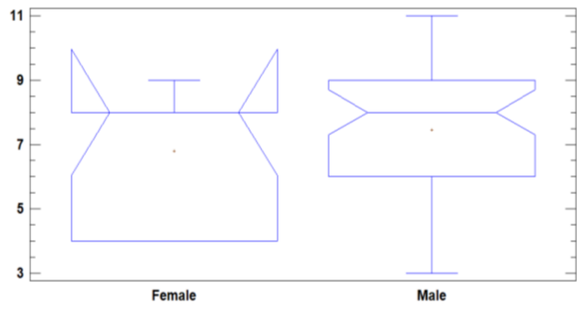

Fig. 4. Boxplot for the variable gender

7.3 Analysis

According to the information in Table 5, it is possible to identify that the average of the faults detected by men is slightly higher than that detected by women, and in the case of the median, this is the same; in terms of variability, that of women presents a slightly higher standard deviation.

The boxplot in figure 4 illustrates a more symmetrical behavior of the sample collected from male experimental subjects than that of women; in the sample of female subjects, a positive asymmetry is observed.

In the case of inferential analysis, unlike the three previous analyses, the samples are independent, so the Mann-Whitney (Wilcoxon) W test to compare medians is appropriate [16]. Based on this test, a P-value = 0.379415 is obtained, so the null hypothesis is accepted with an alpha of 0.05.

7.4 Results

Based on the statistical analysis performed, it is possible to conclude, with 95% confidence, that the effectiveness of the fault detection process using the code review technique is not influenced by the gender of the software engineer, result that coincides with the results obtained in [9].

8 Threats to the Validity of the Study

According to [17] the validity of a study refers to the reliability of its results and to what extent the results are true and not biased by the researchers' point of view. Campbell and Stanley in [18] indicate that threats to the validity of experimental study can be classified as: internal or external.

8.1 Internal Validity

Internal validity is a function of the cause-effect relationships between the factor and the dependent variable, in the context of the phenomenon studied [3]; that is, the variability of the response variable should be caused only by the alternatives of the factor and not by a third variable.

In the case of our study, aspects such as the use of a single object of study for all subjects, a homogeneous # of types of faults (CA and CB) in both halves of the code, and the use of experimental subjects with homogeneous expertise, being students, were considered.

Likewise, in the four statistical analyses, the only variable element was the alternatives of each factor: two fault types according to classification A, six fault types according to classification B, two positions in the code, and the gender of the software engineer.

On the other hand, the analysis of the type of variable, which even though it is numerical, was chosen to be considered as ordinal type, the identification of the paired samples in the first three analyses and independent in the fourth, are elements that ensure the use of hypothesis tests relevant to the statistical conditions of the design.

Threats to validity could be considered that the complexity of the faults injected into the code was not in accordance with the subjects' expertise and that this prevented the activity from being like real industry conditions. In the case of the fourth study, the imbalance in the number of subjects of both genders may not be desirable for a statistical analysis.

8.2 External Validity

This aspect of validity deals with the extent to which the findings can be generalized and the extent to which the findings are of interest to other people outside the case under investigation [3].

As a first element, experimental subjects were selected in the process of training in the task that consisted of the analyzed phenomenon and were even trained in the elements of the study (types of faults in the code), that is, creating the most realistic conditions possible to those that could occur in the industry.

As a second element, a controlled experiment was designed with a task (fault detection) using a technique appropriate to the task, in conditions like reality (reviewing a code) with no further support than the subject's knowledge.

As a third element, the activities carried out were planned and documented, in such a way that the study could be replicated both by the research group and by external researchers, under similar conditions.

However, the use of a non-probabilistic sample, selected at convenience, could also be a limiting aspect for the study.

9 Conclusions

The purpose of software verification is to ensure that the generated software has been correctly developed. In the case of the coding phase, the generated artifact is the code. In this case, a criterion for evaluating its quality is to identify the number of faults it contains.

Although there are previous works that analyze various variables that influence the static verification process, these usually include multiple factors in their analysis. The study here reported was based on a controlled experiment that properly isolated the factors; fault type, fault position and gender of the tester; and despite the threats to its validity, the researchers consider that the empirical evidence generated generates contributions to the body of knowledge of the discipline.

The first two analyses used the two most well-known classifications of fault types. From these analyses we can conclude that fault types influence their detection process, since the tester is more effective in detecting faults by omission than by omission. On the other hand, from the analysis we obtained empirical evidence that there is greater effectiveness in detecting computational faults compared to interface, data and cosmetic. Regarding its position, the third analysis generated evidence that confirms what was found in a couple of previous studies: the errors are more visible in the first half of the code than in the second; one possible explanation is tester fatigue. Finally, the fourth analysis shows that the differences in effectiveness based on the tester's gender are not significant, a conclusion that confirms what was found in previous studies.

With the empirical evidence generated, it is possible to continue research in the context of Software Engineering, using multifactorial designs that allow analyzing the variability of a response variable based on two factors of interest.

References

1. **Washizaki, H. (2024).** Guide to the Software Engineering Body of Knowledge (SWEBOK V4.0). IEEE Computer Society.
2. **Juristo, N., Moreno, A., Vegas, S. (2006).** Software Evaluation Techniques. Course Notes. Universidad Politécnica de Madrid, pp. 8.
3. **Malhotra, R. (2016).** Empirical Research in Software Engineering: Concepts, Analysis, and Applications. Chapman and Hall/CRC.
4. **Basili, V., Selby, T. (1987).** Comparing the Effectiveness of Software Testing Techniques. IEEE Transactions on Software Engineering, Vol. 13, No. 12, pp. 1278–1296.
5. **Kamsties, E., Lott, C.M. (1995).** An Empirical Evaluation of Three Defect-Detection Techniques. European Software Engineering Conference, pp. 362–383. Doi:10.1007/3-540-60406-5_25.
6. **Roper, M., Wood, M., Miller, J. (1997).** An Empirical Evaluation of Defect Detection Techniques. Information and Software Technology, Vol. 39, No. 1, pp. 763–775. Doi: 10.1016/S0950-5849(97)00028-1.
7. **Juristo, N., Vegas, S. (2003).** Functional Testing, Structural Testing and Code Reading:

What Fault Type Do They Each Detect?. Empirical methods and studies in software engineering: Experiences from ESERNET, pp. 208–232.

8. **Aguileta, A., Ucán, J., Aguilar, R. (2017).** Explorando la influencia de los roles de Belbin en la calidad del código generado por estudiantes en un curso de ingeniería de software. Revista Educación en Ingeniería, Vol. 12, No. 23, pp. 93–100. Doi:10.26507/rei.v12n23.742.
9. **Ucán, J., Aguilar, R., Mendoza, J. (2023).** Software testing Usinf the Code Review Technique: An Exploratory Study. Revista electrónica de Computación, Informática, Biomédica y Electrónica, Vol. 12, No. 2, pp. 1–16. Doi:10.32870/recibe. v12i2.312.
10. **Aguilar, R., Diaz, J., Aguileta, A. (2024).** Influence of Team Role Type and Fault Position in the Code Review Process: An Exploratory. Abstraction And Application, Vol. 46, pp. 109–122.
11. **Ucán, J., Aguilar, R., Aguileta, A. (2025).** Influence of Team Role Type and Fault Position in the Code Review Process: An Exploratory. Revista del Centro de Graduados e Investigación del Instituto Tecnológico de Mérida, Vol. 40, No. 109, pp. 107–110.
12. **Linger, R., Mills, H., Witt, B. (1979).** Structured Programming: Theory and Practice. Reading, MA: Addison-Wesley.
13. **Basili, V., Selby, R., Hutchens, D. (1996).** Experimentation in Software Engineering. IEEE Trans. Software Eng., Vol. 12, No. 7, pp. 733–743. Doi:10.1109/TSE.1986.6312975.
14. **Genero, M., Cruz-Lemus, J., Piattini, M. (2014).** Métodos de investigación en ingeniería de software, España: Ra-Ma.
15. **Deitel, P., Deitel, H. (2016).** C++ How to Program. Pearson.
16. **Siegel, S., Castellan, N.J. (1995).** Estadística no paramétrica, aplicada a las ciencias de la conducta. Editorial Trillas. Doi: 10.26439/ persona1998.n001. 1715.
17. **Wohlin, C., Host, M., Runeson, P. (2012).** Experimentation in Software Engineering. Springer Science & Business Media.
18. **Campbell, D.T., Stanley, J.C. (1963).** Experimental and Quasi-experimental Designs for Research. Houghton Mifflin Company Boston.

Article received on 03/06/2025; accepted on 08/10/2025.
**Corresponding author is Raúl A. Aguilar.*