UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO DE SISTEMAS

CARRERA:

INGENIERÍA DE SISTEMAS

TEMA:
"ANÁLISIS DESCRIPTIVO DE LA INCIDENCIA DEL ENCAPSULAMIENTO, DENTRO DEL PARADIGMA ORIENTADO A OBJETOS (POO), EN EL RENDIMIENTO DE UN SOFTWARE CIENTÍFICO"

AUTOR:

Félix Xavier Barahona Cevallos

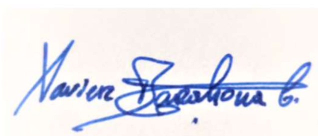
TUTOR:

Msg. Guillermo Pizarro Vásquez

Abril 2021
GUAYAQUIL-ECUADOR

DECLARATORIA DE RESPONSABILIDAD

Yo, **Felix Xavier Barahona Cevallos**, declaro que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad del/los autor/es.



Firma
Felix Barahona

Ing. Guillermo Pizarro, Mg.
C.I.: 0920565066

ANÁLISIS DESCRIPTIVO DE LA INCIDENCIA DEL ENCAPSULAMIENTO, DENTRO DEL PARADIGMA ORIENTADO A OBJETOS (POO), EN EL RENDIMIENTO DE UN SOFTWARE CIENTÍFICO

Abstract. Performance is considered more important feature rather than the application of programming techniques for better software design in most scientific software developers. Here a problem arises if software is written without considering a specific paradigm or some programming technique when performing maintenance, the tasks related to this activity are complicated, since almost no one would understand the source code. The objective of this research is to verify the performance of the software with or without encapsulation component. An ex-post facto experimental methodology has been implemented, carrying out a descriptive analysis of the data and then concluding by verifying the hypothesis by means of a robust test. This work was carried out by running algorithms written in programming language Java, in three groups of data with different conditions to analyze their behavior. The conclusion is that the application of the encapsulation component of the object-oriented paradigm does affect the execution of the scientific software performance.

Keywords: data science, software engineering, scientific software, encapsulation, performance, design software.

1 Introduction

1.1 Object-Oriented Paradigm

Object-Oriented Paradigm (OOP) has been the design principle of many programming languages. The idea behind OOP was derived mainly from the representation of knowledge in the human brain according to the real world. According to this paradigm, everything can be modeled as an object; which is composed of: identity, state and behavior. This allows us to make software design more accessible by information systems developers and architects.

Unhelkar [10] presents six fundamentals of Software Engineering, which are also those of OOP; such as: classification, polymorphism, abstraction, inheritance, association and encapsulation. *Classification* refers to the grouping of identified entities or potential objects; *polymorphism*, as the runtime feature with respect to an instantiated object when understanding a message sent by another object; *abstraction*, understood as the classification of objects that are identified as classes; *inheritance*, which results from classes that have been generalized; the *association*, as the characteristic that allows to relate classes; and *encapsulation*, a feature that is taken into account in this research work, such as the one that locates data and prevents it from being directly exposed to the rest of the system, improving quality and reuse because the data is accessed through calls to operations (methods or functions) of a class and shows the set of "data and code" depending on its visibility (*public*, *private* or *protected*).

1.2 Software Scientific Development

Scientific software development refers to the analysis, design, implementation, testing, and deployment of software applications for scientific research purposes; for example in the field of physics, biology, medical analysis, data science, among others. The need for continuous experimentation and validation of techniques (eg simulations) prior to the publication of scientific results has led to the emergence of the field of scientific software development as an important method for researchers to be successful in multiple fields [5].

According to [1], most of the code implemented for scientific software does not follow a guideline with respect to some paradigm that allows considering some non-functional requirement; such as, coupling, scaling, modularity, among others; since the efficiency of the execution of an algorithm prevails over design techniques.

Hypothesis. The hypothesis of this research work is the following: encapsulation component within the Object Oriented Paradigm (OOP) impacts on the performance of a scientific software.

The structure of this article is as follows: in the first section, a brief state of art about performance of Scientific Software and the paradigms that have been taken into design and implementation are mentioned; in the second section, the source of the data and runtime environment are mentioned; in the third section, an exploratory analysis of the data and its respective robust hypothesis test; in the fourth section, the conclusions and recommendations regarding this research work in general.

2 Materials and Methods

2.1 Experimental Design

This research is based on code of Pizarro's Master's Thesis [9], not yet published; in which, a randomized experimental study was carried out, with simulated data for the generation of batches with orders (instances from [4]) and their respective collection in a rectangular warehouse with one cross aisles (see Fig. 1).

The order grouping algorithms were: *Random* (batches are formed randomly), *First Come First Served (FCFS)* batches are formed according to the orders come up to the capacity of the cart, *Strict Order Picking (SOP)* a batch is formed with a single order, *Greedy 1 (G01)* are ordered from highest to lowest number of items of each order and batches are created, *Greedy 2* (G02) are ordered from lowest to highest and batches are created and *Greedy 3* (G03) are grouped according to the closest orders and batches are created.

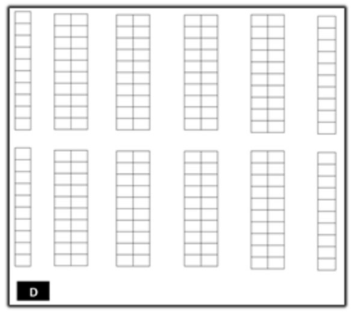


Fig. 1. Rectangular warehouse with one cross aisles.

There is a heuristic that was applied to two groups of experimental data after having a set of solutions obtained with the algorithms explained in the previous paragraph, this heuristic is called **Local Search (LS)** with four variants: *1x0* two batches are taken randomly and a single random order is taken out from each batch then exchanged if is into cart capacity; *1x1* two random batches are taken and a random order is taken from each batch then exchanged; *2x1* two random batches are taken and two random orders are taken from one batch and one random order from another batch is then exchanged; and, *2x2* two random batches are taken and two random orders are taken from one batch and another two random orders are taken from another batch, then exchanged.

It should be noted that exchanges are made if the verification of the cart's capacity is fulfilled; otherwise, the exchange is not made, another batch is sought until the exchange can be made.

Routing algorithms were: ***S-Shape*** (the route through the warehouse is like a letter S) and ***Largest Gap*** (the products are collected first at the top and then the products at the bottom, in general).

2.2 Simulations

Three simulations were run running the order grouping algorithms together with the routing algorithms in three groups:

Group 1. Constructive Algorithms (Random, SOP, FCFS, Greedy 01 – G01, Greedy 02 - G02 and Greedy 03 - G03) with S-Shape and Largest Gap.

Group 2. Random constructive algorithm (Random) and an heuristic of Local Search (1x0, 1x1, 2x1, 2x2) with S-Shape and Largest Gap;

Group 3. Greedy constructive algorithms (G01, G02 and G03) and the heuristic Local Search (1x0, 1x1, 2x1, 2x2) with S-Shape and Largest Gap.

The specification of the detailed experimentation in the previous paragraphs can be seen in Fig. 2:

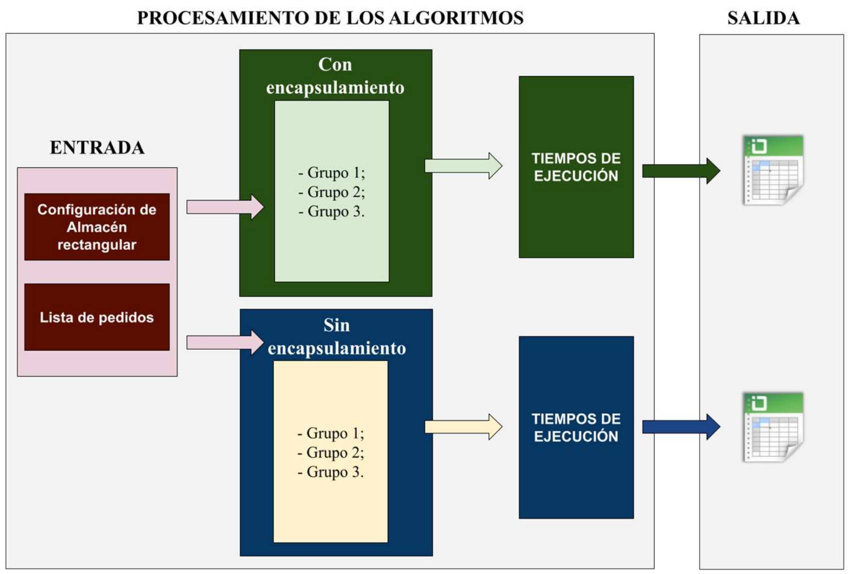


Fig. 2. Scientific research design.

The source code of the warehouse configuration with several cross aisles for this research is implemented in Java [8] based on a Perl code from the research works of [12, 13, 11].

For data analysis, the statistical programming language named R was used and RStudio [7] was used as IDE.

3 Results

3.1 Exploration Data Analysis

Before applying a hypothesis testing to the experimental data, it is necessary to verify them in a descriptive way and thereby check the statistical results of the hypothesis test.

In all boxplot plots (Fig. 3, 4 and 5), they have been applied the logarithm of base 10 with respect to the axis of the execution times (in nanoseconds), so that they can be displayed in an adequate way, as shown in the graphs, since previously they could not be appreciated in a better way due to the amount of aberrant data obtained in experiments.

In these three groups, it can be observed data and its execution times with encapsulation is slightly higher than execution times without encapsulation.

The following table shows the values of means, trimmed means to 10% and variances of execution times (in nanoseconds) of each group that were previously visualized.

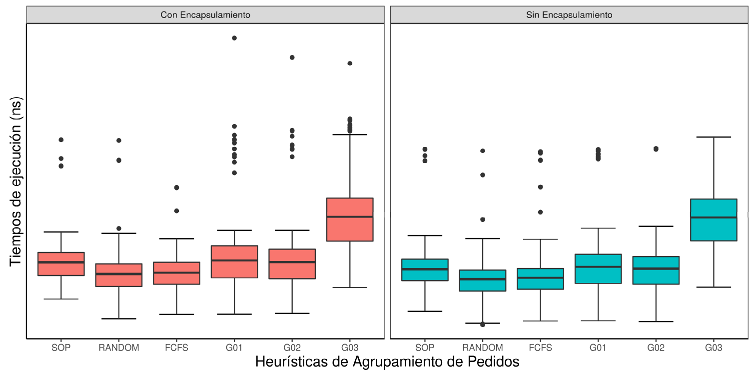


Fig. 3. Box Plot of Group 1 experimental data.

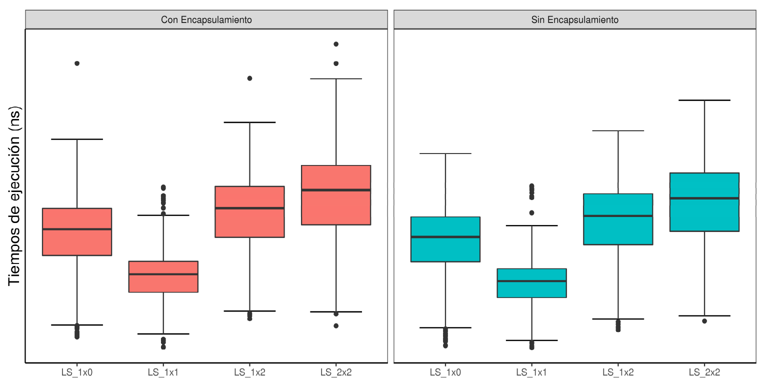


Fig. 4. Box Plot of Group 2 experimental data.

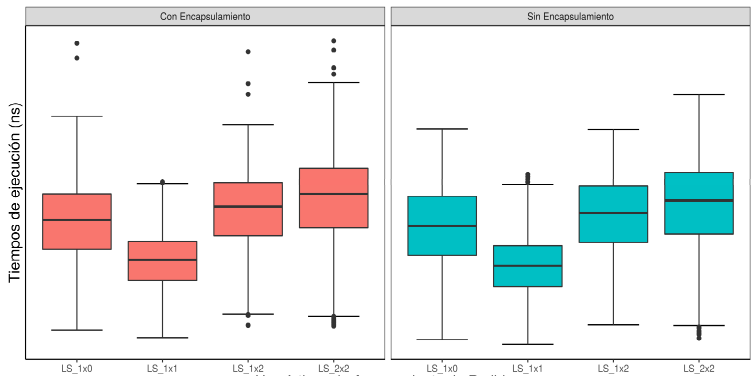


Fig. 5. Box Plot of Group 3 experimental data.

In Table 1, 2 and 3, it can be seen how the experimental data where encapsulation was applied in the source code is greater than the experimental data in which encapsulation was not applied.

Table 1. Descriptive experimental data from **Group 1**.

	With Encapsulation Component			**Without Encapsulation Component**		
	Media	**Trimmed mean**	**Variance**	**Media**	**Trimmed mean**	**Variance**
SOP	1024637.6	984356.9	1.07E+12	862458.2	804322.5	1.52E+12
Random	718507.1	686736.8	8.47E+11	599995.9	576146.4	4.37E+11
FCFS	738722.7	722968.7	1.28E+11	647044.9	605959.2	8.28E+11
G01	1555259.4	1087353.7	3.56E+14	952754.1	862217.4	2.40E+12
G02	1320236.1	1017183.3	1.09E+14	848957.1	814291.2	8.64E+11
G03	5504069.7	4530229.5	1.20E+14	4935173.2	4415207.6	2.52E+13

Table 2. Descriptive experimental data from **Group 2**.

	With Encapsulation Component			**Without Encapsulation Component**		
	Media	**Trimmed mean**	**Variance**	**Media**	**Trimmed mean**	**Variance**
LS 1x0	9557378	8472821	2.69E+14	7483487	6780018	4.16E+13
LS 1x1	2154638	2064382	1.98E+12	1801292	1707401	2.00E+12
LS 2x1	18266779	15728281	4.20E+14	14671111	12794689	2.18E+14
LS 2x2	38733059	29950634	3.52E+15	30546922	24013919	1.71E+15

Table 3. Descriptive experimental data from **Group 3**.

	With Encapsulation Component			**Without Encapsulation Component**		
	Media	**Trimmed**	**Variance**	**Media**	**Trimmed**	**Variance**

	mean			mean	
LS 1x0	19684438	15891130	1.25E+15	16845381	13864973	4.58E+14
LS 1x1	4103574	3677788	1.32E+13	3971539	3346910	1.89E+13
LS 2x1	23209315	20085973	8.91E+14	20067883	17582944	4.41E+14
LS 2x2	40754879	31027295	4.21E+15	34820637	26843721	2.46E+15

In the figures of the density of execution times (Fig. 6, 7 and 8), both with encapsulation and without encapsulation, it's evident that they don't have a normal distribution, which allows us to deduce that an alternative other than a parametric hypothesis testing.

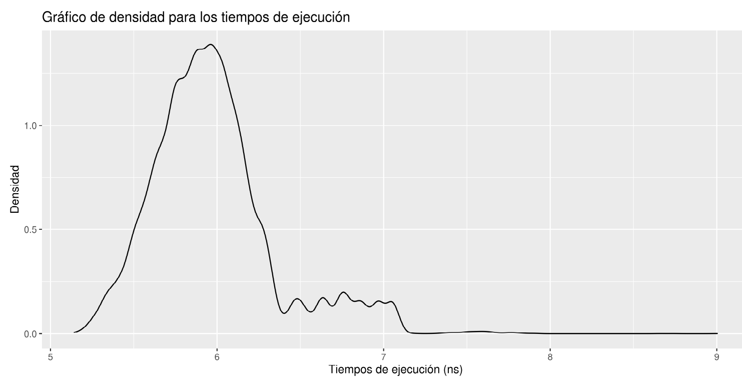


Fig. 6. Density diagram of the experimental data from Group 1.

Fig. 7. Density diagram of the experimental data from Group 2.

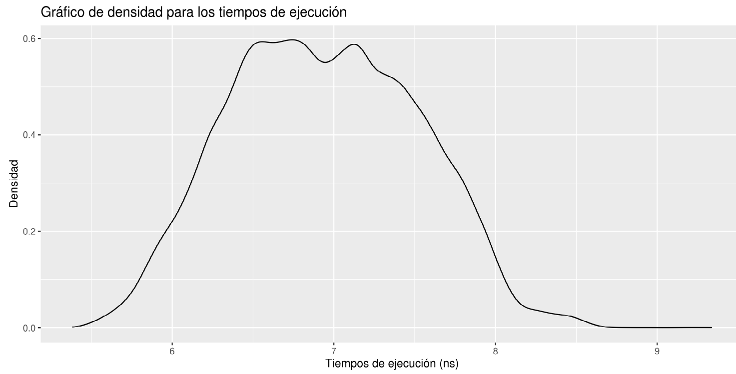


Fig. 8. Density diagram of the experimental data from Group 3.

3.2 Hypothesis Testing

Graphically, these three groups of experimental data don't have a normal distribution; which was verified in the three groups of experimental data, using Lilliefors normality test (Kolmogorov-Smirnov) [3], the null hypothesis being that *"the data have a normal distribution"*:

In Groups 1, 2 and 3, the following conclusion is reached: "*with a value of significance close to zero (<2.2e-16), it can be concluded that the null hypothesis is rejected; therefore, the data do not have a normal distribution"*.

Since these three groups of data do not have a normal distribution; now the requirement to apply the non-parametric hypothesis test must be proved, verifying the null hypothesis: *"homogeneity of the variance in the data"*, using the Bartlett homoscedasticity test [2]:

In Groups 1, 2 and 3, the following conclusion is reached: "*with a value of significance close to zero (<2.2e-16), it can be concluded that the null hypothesis is rejected; therefore, the variance is not homogeneous "*.

Since it does not meet the two previous requirements: neither the assumption of normality nor homogeneity in the variance, now it will be statistically verified if the mean of the execution times with the code implemented with encapsulation is greater than the mean of the execution times with the source code without encapsulation, using Yuend [6] robust hypothesis test of two dependent groups, with null hypothesis: *"the mean of the execution times of each group is equal":*

In Groups 1, 2 and 3, the following conclusion is reached: "*with a significance value close to zero and a difference of means bounded with a positive value, it can be concluded that the null hypothesis is rejected; therefore, the bounded mean of each group is different and that the mean of the execution times with encapsulation is greater than the mean of the execution times without encapsulation"*.

4 Conclusions

The impact on how the source code is implemented, considering encapsulation or not, on the execution times of scientific software; It has been shown statistically that the execution times of the source code with encapsulation are greater than the execution times of the source code without encapsulation.

According to a study [1] where it has not been considered as a matter of interest by the scientific software development community to consider programming techniques; In this research work, it is shown that if considered, it would affect the performance of the software, specifically if encapsulation is included in all defined classes.

In scientific software one of the most important feature is performance; therefore, a paradigm more in line with this type of computer solution must be sought; as the procedural paradigm. From this, it is recommended to carry out future research implementing a solution following the object-oriented paradigm and the same solution to implement it considering the procedural paradigm.

5 Acknowledgment

Thanks to Universidad Politécnica Salesiana, due to the financial contribution to this research work, which is within the framework of the research project: "ACISoft - Computational Analysis in Software Engineering" carried out in the Research Group of Software Engineering and Knowledge Engineering (GIISIC).

References

1. Arvanitou, E.M., Ampatzoglou, A., Chatzigeorgiou, A., Carver, J.C.: Software engineering practices for scientific software development: A systematic mapping study. Journal of Systems and Software 172, 110848 (2021). https://doi.org/https://doi.org/10.1016/j.jss.2020.110848.
2. Bartlett, M.S., Fowler, R.H.: Properties of sufficiency and statistical tests. Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences 160(901), 268–282 (1937). https://doi.org/10.1098/rspa.1937.0109, https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1937.0109
3. Dallal, G.E., Wilkinson, L.: An analytic approximation to the distribution of lilliefors's test statistic for normality. The American Statistician 40(4), 294–296 (1986), http://www.jstor.org/stable/2684607
4. Henn, S., Koch, S., Doerner, K.F., Strauss, C., Wäscher, G.: Metaheuristics for the Order Batching Problem in Manual Order Picking Systems. Business Research 3(1), 82–105 (2010). https://doi.org/10.1007/BF03342717 LNCS Homepage, http://www.springer.com/lncs, last accessed 2016/11/21.
5. Hourani, H., Wasmi, H., Alrawashdeh, T.: A Code Complexity Model of Object Oriented Programming (OOP). In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT). pp. 560–564. IEEE (2019). https://doi.org/10.1109/JEEIT.2019.8717448

6. Mair, P., Wilcox, R.: Robust statistical methods in r using the wrs2 package. Behavior Research Methods 52(2), 464–488 (2019). https://doi.org/10.3758/s13428-019-01246-w
7. Pizarro-Vasquez, G.O.: Statistical analysis source code repository in R. https://github.com/omarjcm/poo-research, [Online; accedido 25-Febrero-2021]
8. Pizarro-Vasquez, G.O.: Repository of source code of the TFM research project in Java. https://github.com/omarjcm/warehouse, [Online; accedido 25-Febrero-2021]
9. Pizarro-Vasquez, G.O., Pardo, E.G.: Resolution of the problem of optimization of the batching and routing of orders in warehouses with multiple transversal aisles (unpublished), unpublished.
10. Unhelkar, B.: Software Engineering with UML. CRC Press (2017)
11. Valle, C.A., Beasley, J.E.: Order batching for picker routing using a distance approximation. arXiv preprint arXiv:1808.00499 (2018)
12. Valle, C.A., Beasley, J.E., da Cunha, A.S.: Modelling and Solving the Joint Order Batching and Picker Routing Problem in Inventories, pp. 81–97. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-45587-7
13. Valle, C.A., Beasley, J.E., da Cunha, A.S.: Optimally solving the joint order batching and picker routing problem. European Journal of Operational Research 262(3), 817–834 (2017). https://doi.org/10.1016/j.ejor.2017.03.069