Journal of Information Systems Education

Volume 35
Issue 3
Summer 2024

Self-Explanation Effect of Cognitive Load Theory in Teaching Basic Programming

Carlos Sandoval-Medina, Carlos Argelio Arévalo-Mercado, Estela Lizbeth Muñoz-Andrade, and Jaime Muñoz-Arteaga

Recommended Citation: Sandoval-Medina, C., Arévalo-Mercado, C. A., Muñoz-Andrade, E. L., & Muñoz-Arteaga, J. (2024). Self-Explanation Effect of Cognitive Load Theory in Teaching Basic Programming. *Journal of Information Systems Education*, 35(3), 303-312. https://doi.org/10.62273/GMIV1698

Article Link: https://jise.org/Volume35/n3/JISE2024v35n3pp303-312.html

Received:	July 5, 2023
First Decision:	September 25, 2023
Accepted:	December 12, 2023
Published:	September 15, 2024

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 35(3), 303-312, Summer 2024
https://doi.org/10.62273/GMIV1698

Self-Explanation Effect of Cognitive Load Theory in Teaching Basic Programming

Carlos Sandoval-Medina
Carlos Argelio Arévalo-Mercado
Department of Information Systems
Autonomous University of Aguascalientes
Aguascalientes, México
al285667@edu.uaa.mx, carlos.arevalo@edu.uaa.mx

Estela Lizbeth Muñoz-Andrade
Department of Electronic Systems
Autonomous University of Aguascalientes
Aguascalientes, México
lizbeth.munoz@edu.uaa.mx

Jaime Muñoz-Arteaga
Department of Information Systems
Autonomous University of Aguascalientes
Aguascalientes, México
jaime.munoz@edu.uaa.mx

ABSTRACT

Learning basic programming concepts in computer science-related fields poses a challenge for students, to the extent that it becomes an academic-social problem, resulting in high failure and dropout rates. Proposed solutions to the problem can be found in the literature, such as the development of new programming languages and environments, the inclusion of virtual and augmented reality, gamification, automatic grading tools, and intelligent tutoring systems, among others. However, most of these solutions do not explicitly describe the application of some learning theory, instead, they focus on new technologies. Cognitive Load Theory (CLT) is an instructional design theory that aligns the design of instructional materials with human cognitive architecture using 17 design guidelines to optimize learning. The goal of this research is to design, develop, and test instructional materials to support the teaching and learning of basic programming, measuring their effectiveness compared to traditional materials, based on the self-explanation effect of CLT. To compare the instructional materials, a quasi-experimental design with homogeneous groups was used, involving students from the Autonomous University of Aguascalientes. The results indicate a positive impact of the use of CLT-based instructional materials, either through the application of a single effect or the combination of two effects such as worked example and self-explanation.

Keywords: Cognitive load theory, Self-explanation, Introductory programming, Computing education, Computing skills

1. INTRODUCTION

The software industry plays a crucial role in business processes and is experiencing a growth trend. One of its core processes is programming, which implies an increasing demand for programmers. According to Voichick et al. (2019), software developers are reported to be one of the professions with the highest growth projection for the year 2030, with an increase of over 30%.

Skills acquired in programming are not only important in education but also in the advancement of technology and communication (Rahman et al., 2020), industry 4.0, as well as in data science and artificial intelligence (Nakagawa et al., 2021). However, learning programming has been widely documented as highly challenging for beginners in computer science-related fields, often resulting in failure rates of around 34% (Bennedsen & Caspersen, 2019; Simon et al., 2019; Watson & Li, 2014).

The application of Cognitive Load Theory (CLT) in the teaching and learning of programming mostly reports positive empirical results (Aureliano et al., 2016; Beege et al., 2021; Price et al., 2020; Sands, 2019; Vieira et al., 2017; Yen &

Wang, 2017). As a result, CLT is one of the most cited instructional design theories in many learning areas, with citations ranging from 10,000 to 20,000 occurrences (Sweller, 2016).

In this paper, the design, development, and application of instructional materials are described. Materials were developed for students at the Autonomous University of Aguascalientes, as support for teaching and learning basic concepts of the C++ programming language in the structured paradigm. The instructional materials were designed and developed based on the self-explanation effect of CLT. A quasi-experimental pre-post study compared the effectiveness of the instructional materials based on the self-explanation effect compared to traditional instructional material in improving learning outcomes.

2. PROBLEM OUTLINE

Learning programming is a challenging task for both students and teachers due to its complex nature. Concepts with a high degree of complexity impose a greater cognitive load on students. However, the brain has a limited capacity for information processing (Sweller et al., 2019), requiring significant effort, dedication, high levels of motivation, time, and ample practice and trial-and-error (Silva-Maceda et al., 2016). Regarding the factors that affect programming learning, on the teaching side, there are factors such as the complexity of the topics, instructional materials, teaching methods, and the learning environment (Insuasti, 2016; Silva-Maceda et al., 2016). On the learning side, student-related factors include gender, mathematical knowledge (Silva-Maceda et al., 2016), complex cognitive skills (Insuasti, 2016), prior knowledge (Anfurrutia et al., 2017), attitude, perception of self-efficacy, discipline, self-forecasting of success or failure, and motivation (Gurer & Tokumaci, 2020; Sharma & Shen, 2018).

We find numerous investigations in the search for alternatives to help in the teaching-learning process, which date back practically from the creation or beginning of programming as such, to the present day, such as the development of new programming languages and environments, the inclusion of virtual and augmented reality, gamification, automatic grading tools, and intelligent tutoring systems, among others. However, most of these alternatives do not explicitly describe the application of some learning theory (Becker & Quille, 2019; Kim et al., 2019; Luxton-Reilly et al., 2018; Oberhauser & Lecon, 2017), and the teaching-learning process of programming still represents a challenge for both teachers and students.

This challenge can be confirmed by the high failure and dropout rates of introductory programming courses, which are around 34% (Bennedsen & Caspersen, 2019; Simon et al., 2019; Watson & Li, 2014). Although the global trend shows a slight decrease in failure rates, this does not apply to Latin America. Studies such as Beltrán et al. (2015) from the Central University of Ecuador report a failure rate of 47%, Juárez Viveros et al. (2016) from the Technological Institute of Mexicali report a 49.7% failure rate, Justo-López et al. (2021) from the Autonomous University of Baja California report a 38.4% failure rate, Cerón Garnica et al. (2021) from the Benemérita Autonomous University of Puebla report a 46% failure rate, and the Autonomous University of Aguascalientes, through its statistics department, presents varying percentages with most values above 30%.

The problem of learning programming languages in the teaching-learning process has become an academic-social problem, not only directly affecting students but the development of fundamental and instrumental competencies in the knowledge society (Tejera-Martínez et al., 2020).

3. BASIC CONCEPTS

3.1 Cognitive Load Theory (CLT)

The Cognitive Load Theory (Sweller et al., 1998, 2019) is an instructional theory based on evolutionary biology (Geary, 2012), human cognitive architecture (Atkinson & Shiffrin, 1968), and Bartlett's schema theory (Carbon & Albrecht, 2012). In the human brain, information processing occurs through two subsystems: working memory, which is highly limited; and long-term memory, which is unlimited.

When information is processed in working memory, it is transferred to long-term memory, where it is stored and organized into collections called schemas. These schemas can last a lifetime and can eventually become automated, requiring minimal cognitive resources (through practice). According to the CLT, learning involves the construction of schemas (Berssanette & De Francisco, 2022; Sweller et al., 2019). Therefore, instructional design should aim to reduce the cognitive load imposed on working memory (Sweller, 2016).

The term cognitive load refers to the number of resources that working memory requires for a learning process (Berssanette & De Francisco, 2022). CLT identifies three types of loads that are involved in learning processes. Intrinsic cognitive load is associated with the complexity of the information or knowledge to be learned and the student's prior knowledge. Extraneous cognitive load is related to how the information is presented to the student, influenced by instructional methods, and it can be modified. Finally, germane cognitive load is the load necessary for learning, which refers to the resources allocated by working memory during the learning process (Sweller et al., 2019).

From research on CLT, 17 "effects" have been identified that predict learning outcomes under specific instructional conditions that modify extraneous cognitive load and promote germane cognitive load (Sweller et al., 2019). These effects have mostly shown positive learning results. Among them, the most cited effects are the worked-example effect and the completion problem effect (Andersen et al., 2016; Beege et al., 2021; Sands, 2019; Zhi et al., 2019). However, limitations have also been reported regarding their application and effectiveness, where their efficiency may decrease (Moreno, 2006; Rittle-Johnson & Loehr, 2017) or even be counterproductive (Kalyuga, 2007).

The most studied effect in CLT and commonly used instructional strategy for studying or review purposes is worked examples (Sweller et al., 2019). However, learning from worked examples has limitations when the learner has not acquired sufficient knowledge, and they tend to study them superficially. However, when learners are asked for verbal explanations of the problem (Kalyuga, 2007; Rittle-Johnson & Loehr, 2017), it promotes the search for and reinforcement of schemas in long-term memory.

3.2 The Self-Explanation Effect

The self-explanation effect is a mental process in which the student, while studying a worked example, establishes the relationship(s) between it and the previously acquired knowledge through self-explanation. This process promotes relevant cognitive load by creating connections with existing prior knowledge in long-term memory and generating cognitive schemas.

For this reason, the application of the self-explanation effect is recommended for students with prior knowledge, as students without adequate prior knowledge may generate incorrect explanations and develop erroneous cognitive concepts (Sweller et al., 2019). Self-explanation can be described as an active learning construction activity that engages students with the content, allowing them to monitor their level of understanding (Vihavainen et al., 2015). At times, it may be necessary for the instructor to provide an explanation when the student is unable to generate a correct or complete explanation (Bisra et al., 2018).

This research aims to evaluate the effectiveness of CLT's self-explanation and worked example effects, through the design of instructional materials to support the teaching of basic C++ programming. Using quasi-experimental methods, the study compares these CLT-based instructional materials with the use of traditional classroom examples at the Autonomous University of Aguascalientes.

4. RELATED WORK

The literature reports benefits of utilizing the self-explanation effect that are applicable to most educational domains (Bisra et al., 2018). However, it also highlights the limitations that need to be addressed to optimize its effectiveness (Rittle-Johnson & Loehr, 2017).

In the field of programming, Vihavainen et al. (2015) employed the inclusion of two types of self-explanation questions, with and without help options, through an online programming exercise book and self-explanation tasks. They reported positive results reflected in the grades of the students.

Yen and Wang (2017) developed a programming environment based on self-explanation that provides feedback using an ontology created by expert programmers for C++ programming, with a focus on correcting programming syntax. They reported that the tool successfully provides appropriate learning materials and extended examples for programmers to correct their potential misconceptions, thereby helping them develop solid concepts.

Price et al. (2020) present a prototype called "Instructor's Solution" as support for online programming tasks. The prototype is based on comparison, self-explanation, and incomplete explanations (explain prompts). Their results indicate that students can learn more with explain prompts, but it is also reflected in increased time investment, which can lead to a loss of interest for many students.

5. METHOD

In this research, instructional materials were designed based on self-explanation problems, based on the self-explanation effect in CLT. They were developed to support the teaching and learning of basic concepts in the C++ programming language, and their effectiveness was evaluated in terms of learning objectives compared to traditional instructional materials. Traditional instructional materials consist of exercises or notes used in a setting where the teacher explains topics and solve examples step-by-step to the group. A second set of materials was created by combining these self-explanation problems with videos of worked examples following the principles of CLT.

This research was done in two phases. The first involved the design of instructional materials, which included the creation of problems and videos based on the CLT effects. The second consisted of the application of the materials and the evaluation of its effectiveness in learning through a quasi-experimental design.

5.1 Design of Instructional Material

Here are some examples of traditional problems commonly used in class on the topics of arrays and strings, in which the teacher demonstrates the solution:

- "Create a program to fill an array with *n* numbers (defined by the user) and order the contents of the array from smallest to largest."
- "Create a program that, given a string of characters, inverts the content of said string, saves it in an additional variable and outputs both strings (Note: without using the strcmp, strcpy, or strlen functions)."
- "Create a program that generates *n* (data entered by the user) next numbers of the Fibonacci series. Remember that the series begins with 0 and 1 where the next number is the sum of the previous 2."

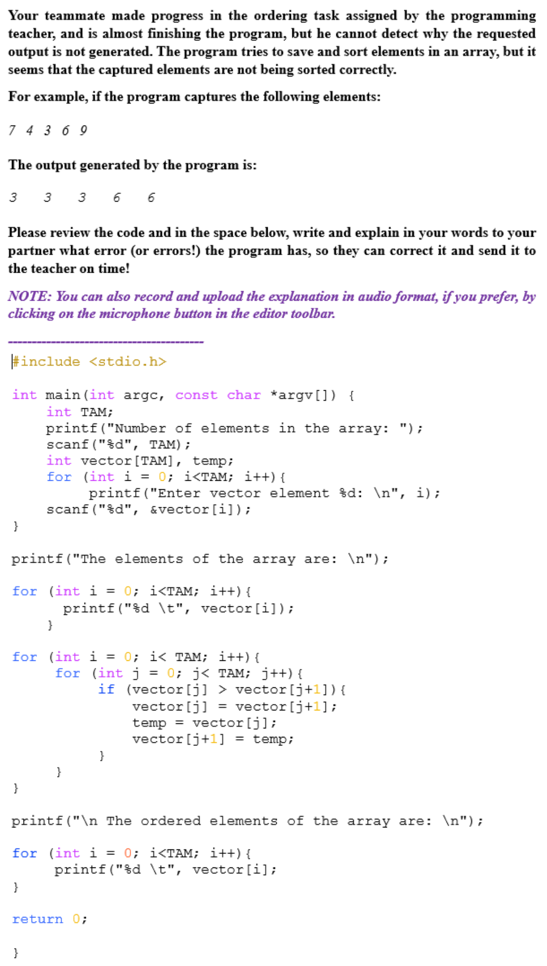
Similar problems were designed in topics related to arrays and strings, but these problems were transformed into a self-explanation format. In this format, a "real-life situation" is presented where the student needs to identify the error(s) in code (like the ones seen in class) "developed by a classmate." The student then needs to explain to their "classmate" the reasons for the errors and help them correct the program to produce the correct output. These exercises consist of three parts: problem description, program output, and code with introduced errors. An example of such a problem is shown in Figure 1. The design structure of these problems is described below:

1) *The problem description:* It describes what the student needs to do to solve the exercise.
2) *The program output*: It shows an example of the output generated by the program for the given exercise.
3) *Source code:* The source code of the program for the exercise is included, which the student must analyze to identify the errors.

In addition to the problems in self-explanation format, videos of solved examples were created, like the presented problems, to combine the self-explanation effect with the worked example effect in an audiovisual format and apply it to a second experimental group. The development of instructional materials was carried out following the principles of CLT aimed at reducing cognitive load through the five phases described in Figure 2.

5.2 Experimental Design

A quasi-experimental PRE-POST study was conducted with three groups: Experimental Group 1 (Exp1) n = 50, Experimental Group 2 (Exp2) n = 49, and Control Group (Cont) n = 45, with the same instructor, consisting of first-year students majoring in Computer Systems Engineering, in a distance education context for all groups (due to the COVID-19 pandemic). In this study, the treatment for Exp1 involved the application of self-explanation problems, for Exp2 it included self-explanation problems + videos of solved examples, and the Control group received exercises covered in class. The topics covered included array sorting, nested loops, matrix addition, string length, string reversal, and string comparison.



Your teammate made progress in the ordering task assigned by the programming teacher, and is almost finishing the program, but he cannot detect why the requested output is not generated. The program tries to save and sort elements in an array, but it seems that the captured elements are not being sorted correctly.

For example, if the program captures the following elements:

7 4 3 6 9

The output generated by the program is:

3 3 3 6 6

Please review the code and in the space below, write and explain in your words to your partner what error (or errors!) the program has, so they can correct it and send it to the teacher on time!

NOTE: You can also record and upload the explanation in audio format, if you prefer, by clicking on the microphone button in the editor toolbar.

```
#include <stdio.h>

int main(int argc, const char *argv[]) {
    int TAM;
    printf("Number of elements in the array: ");
    scanf("%d", TAM);
    int vector[TAM], temp;
    for (int i = 0; i<TAM; i++){
         printf("Enter vector element %d: \n", i);
    scanf("%d", &vector[i]);
}

printf("The elements of the array are: \n");

for (int i = 0; i<TAM; i++){
      printf("%d \t", vector[i]);
    }

for (int i = 0; i< TAM; i++){
     for (int j = 0; j< TAM; j++){
          if (vector[j] > vector[j+1]){
              vector[j] = vector[j+1];
              temp = vector[j];
              vector[j+1] = temp;
          }
     }
}

printf("\n The ordered elements of the array are: \n");

for (int i = 0; i<TAM; i++){
     printf("%d \t", vector[i];
}

return 0;

}
```

Figure 1. Problem in Self-Explanation Format

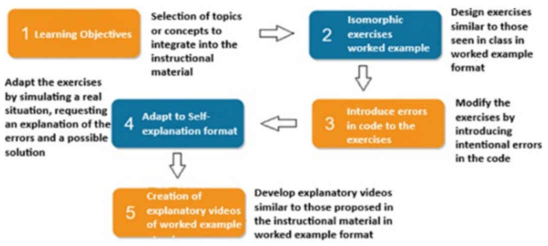

1 Learning Objectives
Selection of topics or concepts to integrate into the instructional material
2 Isomorphic exercises worked example
Design exercises similar to those seen in class in worked example format
3 Introduce errors in code to the exercises
Modify the exercises by introducing intentional errors in the code
4 Adapt to Self-explanation format
Adapt the exercises by simulating a real situation, requesting an explanation of the errors and a possible solution
5 Creation of explanatory videos of worked example
Develop explanatory videos similar to those proposed in the instructional material in worked example format

Figure 2. Proposed Phases for the Development of Instructional Material

These problems were implemented on an educational platform based on Moodle, using open-ended essay-type questions. The exercises given in the experimental treatment were assigned as tasks to be graded, to encourage student participation. They were presented in a format where participants were given a context simulating a real-life situation ("Your teammate has written the assigned sorting program but cannot find the error"), followed by the program output with errors and the corresponding source code to analyze. Participants were then asked to explain in their own words, either in written text or audio format, the reason for the error and provide a solution while explaining their proposed solution.

An automatic feedback system was designed, which provided students with an explanation from the instructor about errors in the exercises, along with suggested solutions and explanations for them. This feedback was presented to students once they completed the exercises on the platform, allowing for immediate feedback. The Cont studied in a traditional manner, reviewing their exercises in class.

After the treatment, learning outcomes were evaluated using standardized and calibrated exams administered by the programming department of the Electrical Systems Department of the Autonomous University of Aguascalientes. These exams were given to all groups and focused only on the topics of arrays, vectors, and strings used in the design of the instructional material. The exams were conducted during the second (PRE) and third (POST) grading periods. To make the PRE and POST results comparable, isomorphic problems were used. For example, typical PRE exam problems were:

a) Briefly explain what the following code does:

```
#include <stdio.h>
int main(){
   int cal=0;
   do{
      printf("Type the school grade: ");
      scanf("%d", cal);
   }while (cal<0||cal>10);
}
```

b) In the city of Aguascalientes, we have 5 hospitals. As we know, a hospital has nurses, doctors, and interns. A staff of 30 people is being assigned to each hospital. Run a simulation using random numbers, where 1 identifies nurses, 2 doctors, 3 interns. You are asked to write a program that uses functions that get:

1) How many doctors belong to each hospital.

2) The number of doctors to be hired by the 5 hospitals.

You are asked to make use of nested loops.

The equivalent isomorphic POST problems were:

a) Briefly explain what the following code does:

```
#include <stdio.h>
int main(){
   int x, month, days=0, aa=2020;
   printf("What month do you want?");
   scanf(%d, &month);
   if(month==1||month==3||month==5||month==7
   ||month==8||month==10||month==12){
   days=31;
   }
   for(x=1, x<=days; x++){
      printf("%d\t", x);
      if(x%7==0){
         printf("\n");
      }
```

}
}

b) There are 10 teams in the Mexican Pacific Baseball League, each team has a team of 30 players. Each team in the league prepares a list where the following data appears for each player: weight and age (which on this occasion are not captured by keyboard, they will be generated randomly)
Weight: range from 45 to 80
Age: range from 18 to 40
You are asked to write a program that does the following using functions:
1) Average weight of the players
2) How many players per team are between 20 and 30 years old?
It is requested to use nested loops.

The provided isomorphic problems used in PRE and POST are comparable, given that they have a similar structure (use of loops and use of functions) with only slight variations of context.

6. RESULTS

For data analysis, the dependent variable was the grades obtained in the second (PRE) and third (POST) partial exams, specifically for the items related to the topics of array sorting, nested loops, matrix summation, string length, string reversal, and string comparison from the developed instructional materials. The independent variable depended on the instructional material used, with the experimental treatment groups (self-explanation problems and worked example videos) and the traditional instructional material group (practice example problems). Descriptive statistics of the participating groups are shown in Table 1. For Exp1 (only self-explanation problems), the pre-treatment (PRE) and post-treatment (POST) grades had an average of 7.76 and 9.46, respectively, indicating an increase of 1.7 points in the average grades. In the case of Exp2 (self-explanation problems + worked example videos), the average grades were 5.97 PRE and 8.40 POST, reflecting an increase of 2.4 points in the average grades. The results for the control group (Cont) show an average of 8.10 for the PRE grades and 8.73 for the POST grades, indicating a difference of 0.6 points higher in the POST grades.

Group	Grades	N	Min.	Max.	Mean	Typ. Dev.
EXP1	PRE	50	3.60	10.00	7.76	1.92
	POST	50	8.00	10.00	9.46	0.74
EXP2	PRE	49	2.60	10.00	5.97	2.16
	POST	49	2.60	10.00	8.40	1.76
Cont	PRE	45	0.80	10.00	8.10	2.55
	POST	45	3.80	10.00	8.73	1.84

Table 1. Descriptive Statistics Groups Participating in the Study

The frequency distribution of Exp1 is illustrated in Figure 3, which shows that most observations in the PRE phase are in the range of 4 to 8. In contrast, the observations in the POST phase are concentrated in the range of 8 to 10. This indicates that there was a shift towards higher grades after the treatment, with a greater number of students achieving higher scores.

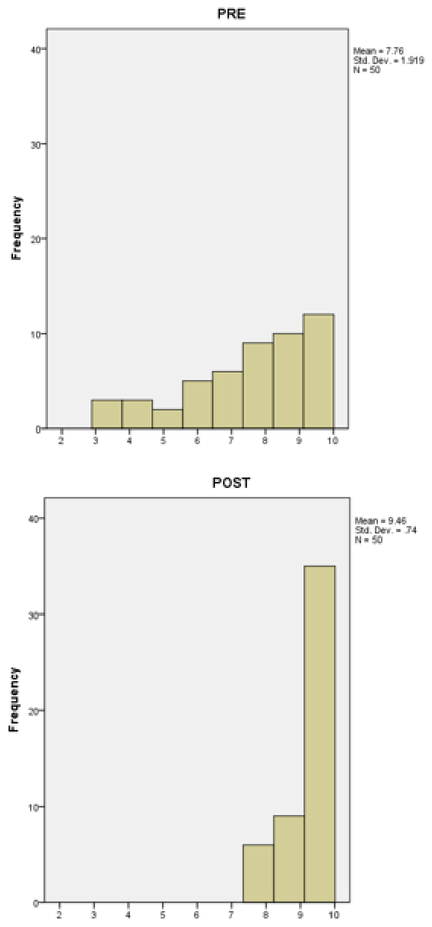


Figure 3. Histograms of PRE and POST Observations of the Experimental Group 1

For Exp2, the observations in the PRE phase indicate that most scores are in the range of 2 to 6. However, in the POST phase, most scores are in the range of 7 to 10, as shown in Figure 4.

For the Cont group, most observations in the PRE phase are in the range of 9 to 10. The same pattern is observed in the POST phase, where most scores are still in the range of 9 to 10. Additionally, the incidence of observations with low values is very similar in both the PRE and POST phases, as shown in Figure 5.

Table 2 presents the results of the t-test for the PRE and POST scores of each group. For both the Exp1 (self-explanation) and Exp2 (self-explanation + worked example videos) groups, the p-value is 0.000, indicating a statistically significant difference between the PRE and POST scores. Similarly, for the Cont, the t-test yields a p-value of 0.017, indicating a statistically significant difference between the PRE and POST scores.

A one-way ANOVA test was conducted for the three groups, both for the PRE and POST scores. The analysis reveals that the groups are different in terms of the average scores, both in the PRE and POST assessments, with a p-value of 0.000 for the PRE scores and a p-value of 0.002 for the POST scores, as shown in Table 3.

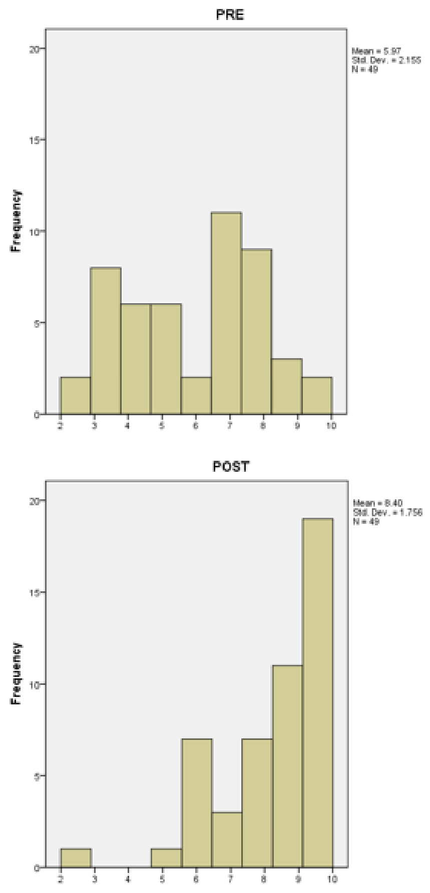


Figure 4. Histograms of PRE and POST Observations of the Experimental Group 2

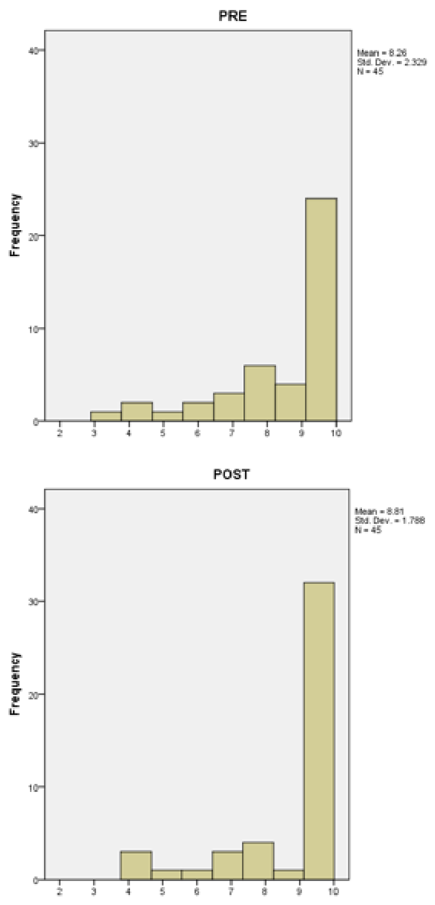


Figure 5. Histograms of PRE and POST Observations of the Control Group

	Related differences					t	gl	Sig. (Bilateral)
	Mean	Typ. Dev.	Typ. Error	95% Confidence interval				
				Lower	Upper			
Exp1POST-Exp1PRE	1.7	2.07	0.29	1.11	2.29	5.80	50	0.000
Exp2POST-Exp2PRE	2.4	2.53	0.36	1.70	3.15	6.72	49	0.000
ContPOST-ContPRE	0.6	1.49	0.22	0.10	0.99	2.49	45	0.017

Table 2. T-Test of Related Samples PRE-POST Participating Groups

Grade		Sum of Squares	gl	Square Mean	F	Sig.
PRE	Inter-groups	126.042	2	63.021	12.849	0.000
	Intra-groups	696.486	142	4.905		
POST	Inter-groups	29.051	2	14.525	6.296	0.002
	Intra-groups	327.603	142	2.307		

Table 3. One-Way ANOVA for PRE and POST Scores

Dependent Variable	(I) Group	(J) Group	Difference of Means (I-J)	Typical Error	Sig	Confidence Interval 95%	
						Lower Limit	Upper Limit
Grade PRE	Exp1	Cont	-0.34	0.45	0.734	-1.41	0.73
		Exp2	1.79*	0.46	0.000	0.73	2.84
	Exp2	Exp1	-1.79*	0.46	0.000	-2.84	-0.73
		Cont	-2.13*	0.45	0.000	-3.20	-1.05
	Cont	Exp1	0.34	0.45	0.734	-0.73	1.41
		Exp2	2.13*	0.45	0.000	1.05	3.20
Grade POST	Exp1	Cont	0.72	0.31	0.055	-0.01	1.46
		Exp2	1.06*	0.31	0.002	0.34	1.78
	Exp2	Exp1	-1.06*	0.31	0.002	-1.78	-0.34
		Cont	-0.34	0.31	0.528	-1.08	0.40
	Cont	Exp1	-0.72	0.31	0.055	-1.46	0.01
		Exp2	0.34	0.31	0.528	-0.40	1.08

*The mean differences are significant at the level 0.05.

Table 4. Post Hoc Tukey Multiple Comparisons

Based on the ANOVA analysis, a post-hoc analysis using the Tukey test was conducted to identify which groups differ in terms of the average scores, both in the PRE and POST assessments. The results for the PRE scores, presented in Table 4, reveal that there are no differences between Exp1 and Cont with a p-value of 0.734. However, there is a significant difference between Exp2 and Cont with a mean difference of 2.13 and a p value of 0.000. Likewise, there is a significant difference between Exp2 and Exp1 with a mean difference of 1.79 and p value of 0.000. For the POST scores, the results of the Tukey post-hoc test reported in Table 4 indicate that there are no differences between Exp1 and Cont (p = 0.055), although this value is at the significance threshold of 0.05. There are also no differences between Cont and Exp2 (p = 0.528). However, there is a significant difference between the two experimental groups, i.e., the self-explication group and the worked example + self-explication group, with a mean difference of 1.06 and p value of 0.002. These results indicate that there is a significant difference between the two experimental groups, but no significant difference between Exp1 and Cont or between Cont and Exp2. In summary, for the PRE averages the group that differs from the other two is Exp2, but for the POST averages the group that differs is the Exp1 group.

7. DISCUSSION

7.1 Interesting Insights

The results suggest that the use of instructional material designed under the format of the self-explanation effect of cognitive load theory benefited student learning in the topics of arrays and strings. The results also suggest that the combination of two or more effects of the cognitive load theory, such as the worked example and self-explanation, can also produce positive results in terms of learning, as shown in Tables 1 and 2, as well as in the PRE and POST histograms of the distribution of the scores of the experimental groups. Furthermore, the results of the Tukey post hoc test show that the exp1 group has a greater difference in terms of POST average over the other two groups, and the exp2 group in PRE average differed negatively even more than the other two groups, reducing that difference in the average POST as shown in the results of Table 4, showing a significant increase in average from PRE to POST.

In accord with our results and based on international studies related to the application of the self-explanation effect (Price et al., 2020; Vihavainen et al., 2015; Yen & Wang, 2017) and the worked-example effect (Beege et al., 2021; Sands, 2019; Zhi et al., 2019) in programming education and various other fields of study, most of these studies have reported positive findings regarding their application. These empirical results demonstrate the effectiveness of the cognitive load theory (CLT) effects in terms of instructional design and implementation. Therefore, we argue that the increase in average grades observed in the experimental groups can be attributed to the use of instructional materials based on CLT principles.

7.2 Limitations of the Study

In the context of distance education resulting from the COVID-19 pandemic, due to the lack of control over the application of the experimental treatment, participants may have used other sources or study tools in addition to the provided instructional material.

Furthermore, a component of the self-explanation effect is the use of prompts during problem explanation, which was not possible to implement in this study. It is possible that this circumstance can be partially overcome with the use of multiple-choice questions, as considered in (Vihavainen et al., 2015), but we believe that this limits the spontaneity and experience that a face-to-face instructor can apply to the task and to evaluation of the learner's questions and explanations. Instead, textual or audio explanations were provided asynchronously.

Also, although care was taken to check that the students' responses were not copied, some copied responses were eliminated. As a control measure, it was emphasized to the participants that the benefit of the exercise is more in the process and mental effort than in the correctness of the responses, since the automatic feedback covered that aspect.

Finally, the results cannot be generalized due to the quasi-experimental design of the study (that is, groups were formed using the prearranged university cohorts), but we believe that, in this sense, they have value as a field study, given its application in a real programming study context, and it shows signs of a positive effect on the learning of basic data structures.

8. CONCLUSIONS

The results obtained from the t test for the PRE and POST scores for both experimental groups yield a p value of 0.000, indicating a statistically significant difference between both scores for both groups. This indicates a positive effect translated into a learning gain that it is shown in the average grade after the application of the study. The increase in the average grade for experimental group 1 (self-explanation) was 1.7 points and for experimental group 2 (self-explanation + solved example videos) was 2.4 points.

There are indications that the use of instructional materials that employ the self-explanation effect as a teaching guide improves the learning of basic programming skills. The gain in terms of learning (reflected in the average grades) yielded better results compared to traditional instructional materials. Furthermore, combining the self-explanation effect with the worked example or problem completion effect also improves learning. However, although the exp2 group obtained a greater difference in increase in the average PRE at POST than the exp1 group, we cannot affirm that the exp2 group was better since the PRE averages of the experimental groups differed significantly. We argue that students can derive benefits from self-explanation exercises through the reinforcement of existing schemas acquired during previous stages of learning the programming language.

The use of self-explanation in instructional materials can provide "variability in practice" (Sweller et al., 2019), by stimulating the contrast and comparison of examples, especially when there are limited solved examples available in the instructional materials provided by teachers, as may be the case for instructors in Mexico and Latin America.

Finally, due to the COVID-19 pandemic, in a distance education context for all groups, the implementation of the experimental treatment was carried out in a remote and asynchronous mode, which limited the control over the treatment. Therefore, we argue that the application of the experimental treatment in face-to-face mode with rigorous control in the treatment, including the use of real-time prompts during the exercises, can benefit students, reflecting this benefit in better learning.

9. REFERENCES

Anfurrutia, F. I., Álvarez, A., Larrañaga, M., & López-Gil, J. M. (2017). Visual Programming Environments for Object-Oriented Programming: Acceptance and Effects on Student Motivation. *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje*, 12(3), 124-131.

Andersen, S. A. W., Mikkelsen, P. T., Konge, L., Cayé-Thomasen, P., & Sørensen, M. S. (2016). The Effect of Implementing Cognitive Load Theory-Based Design Principles in Virtual Reality Simulation Training of Surgical Skills: A Randomized Controlled Trial. *Advances in Simulation*, 1(20), 1-8.

Atkinson, R. C., & Shiffrin, R. M. (1968). Human Memory: A Proposed System and Its Control Processes. *Psychology of Learning and Motivation* (vol. 2, pp. 89-195). Academic Press.

Aureliano, V. C. O., Tedesco, P. C. De A.R., & Caspersen, M. E. (2016, June). Learning Programming Through Stepwise Self-Explanations. *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-6). IEEE.

Becker, B. A., & Quille, K. (2019, February). 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 338-344).

Beege, M., Schneider, S., Nebel, S., Zimm, J., Windisch, S., & Rey, G. D. (2021). Learning Programming From Erroneous Worked-Examples. Which Type of Error Is Beneficial for Learning? *Learning and Instruction*, 75, 101497.

Beltrán, J., Sánchez, H., & Rico, M. (2015). Análisis Cuantitativo y Cualitativo del Aprendizaje de Programación I en la Universidad Central del Ecuador. *Revista Tecnológica - ESPOL*, 28(5), 194-210.

Bennedsen, J., & Caspersen, M. E. (2019). Failure Rates in Introductory Programming: 12 Years Later. *ACM Inroads*, 10(2), 30-36.

Berssanette, J. H., & De Francisco, A. C. (2022). Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review. *IEEE Transactions on Education*, 65(3), 440-449.

Bisra, K., Liu, Q., Nesbit, J. C., Salimi, F., & Winne, P. H. (2018). Inducing Self-Explanation: A Meta-Analysis. *Educational Psychology Review*, 30, 703-725.

Carbon, C. C., & Albrecht, S. (2012). Bartlett's Schema Theory: The Unreplicated "Portrait D'homme" Series from 1932. *Quarterly Journal of Experimental Psychology*, 65(11), 2258-2270.

Cerón Garnica, C. C., Sierra, E. A., Márquez, A. C., & Martínez, B. B. (2018). Experiencias de las E-actividades de Evaluación de las Competencias del Nivel Básico del Área de Programación: Experiences of the E-Activities for Evaluation of the Competences of the Basic Level in the Area of Programming. *Tecnología Educativa Revista CONAIC*, 5(3), 65-74.

Geary, D. C. (2012). Evolutionary Educational Psychology. In K. R. Harris, S. Graham, T. Urdan, C. B. McCormick, G. M. Sinatra, & J. Sweller (Eds.), *APA Educational Psychology Handbook* (vol. 1). Theories, Constructs, and Critical Issues (pp. 597-621). American Psychological Association.

Gurer, M. D., & Tokumaci, S. (2020). Factors Affecting Engineering Students' Achievement in Computer Programming. *International Journal of Computer Science Education in Schools*, 3(4), 1-12.

Insuasti, J. (2016). Problemas de Enseñanza y Aprendizaje de los Fundamentos de Programación. *Revista de Educación y Desarrollo Social*, 10(2), 234-246.

Juárez Viveros, J., López Gerardo, M., & Villareal González, Y. (2016). Estrategias Para Reducir el Índice de Reprobación en Fundamentos de Programación de Sistemas Computacionales del I.T. Mexicali. *Revista de Gestión Empresarial Y Sustentabilidad*, 2(1), 25-41.

Justo-López, Araceli C., Aguilar-Salinas, Wendolyn E., De las Fuentes-Lara, Maximiliano, & Astorga-Vargas, María A. (2021). Uso de Videos Educativos en la Materia de Programación Durante la Etapa Básica de Ingeniería. *Formación Universitaria*, 14(6), 51-64.

Kalyuga, S. (2007). Expertise Reversal Effect and Its Implications for Learner-Tailored Instruction. *Educational Psychology Review*, 19(4), 509-539.

Kim, J., Agarwal, S., Marotta, K., Li, S., Leo, J., & Chau, D. H. (2019). Mixed Reality for Learning Programming. *Proceedings of the 18th ACM International Conference on Interaction Design and Children* (2, pp. 574-579).

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory Programming: A Systematic Literature Review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 55-106).

Moreno, R. (2006). When Worked Examples Don't Work: Is Cognitive Load Theory at an Impasse? *Learning and Instruction,* 16(2), 170-181.

Nakagawa, E. Y., Antonino, P. O., Schnicke, F., Kuhn, T., & Liggesmeyer, P. (2021). Continuous Systems and Software Engineering for Industry 4.0: A Disruptive View. *Information and Software Technology*, 135, 106562.

Oberhauser, R., & Lecon, C. (2017). Virtual Reality Flythrough of Program Code Structures. *Proceedings of the Virtual Reality International Conference-Laval Virtual 2017* (pp. 1-4).

Price, T. W., Williams, J. J., Solyst, J., & Marwan, S. (2020). Engaging Students With Instructor Solutions in Online Programming Homework. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (pp. 1-7).

Rahman, M. M., Watanobe, Y., & Nakamura, K. (2020). Evaluation of Source Codes Using Bidirectional LSTM Neural Network. *The 3rd IEEE International Conference on Knowledge Innovation and Invention (ICKII), Kaohsiung, Taiwan* (pp. 140-143).

Rittle-Johnson, B., & Loehr, A. M. (2017). Eliciting Explanations: Constraints on When Self-Explanation Aids Learning. *Psychonomic Bulletin and Review*, 24(5), 1501-1510.

Sands, P. (2019). Addressing Cognitive Load in the Computer Science Classroom. *ACM Inroads*, 10(1), 44-51.

Sharma, R., & Shen, H. (2018). The Interplay of Factors Affecting Learning of Introductory Programming: A Comparative Study of an Australian and an Indian University. *The 13th International Conference on Computer Science and Education* (ICCSE) (pp. 1-6).

Silva-Maceda, G., David Arjona-Villicana, P., & Edgar Castillo-Barrera, F. (2016). More Time or Better Tools? A Large-Scale Retrospective Comparison of Pedagogical Approaches to Teach Programming. *IEEE Transactions on Education*, 59(4), 274-281.

Simon, Luxton-Reilly, A., Ajanovski, V. V., Fouh, E., Gonsalvez, C., Leinonen, J., Parkinson, J., Poole, M., & Thota, N. (2019). Pass Rates in Introductory Programming and in Other STEM Disciplines. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 53-71).

Sweller, J. (2016). Cognitive Load Theory and Computer Science Education. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (p. 1).

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (1998). Cognitive Architecture and Instructional Design. *Educational Psychology Review*, 10(3), 251-295.

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive Architecture and Instructional Design: 20 Years Later. *Educational Psychology Review*, 31(2), 261-292.

Tejera-Martínez, F., Aguilera, D., & Vílchez-González, J. M. (2020). Programming Languages and Development of Key Competences. Systematic Review. *Revista Electrónica de Investigación Educativa*, 22, 1-12.

Vieira, C., Magana, A. J., Falk, M. L., & García, R. E. (2017). Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education. *ACM Transactions on Computing Education*, 17(4), 1-21.

Vihavainen, A., Miller, C. S., & Settle, A. (2015). Benefits of Self-Explanation in Introductory Programming. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (68, pp. 284-289).

Voichick, F., Gao, G., Ichinco, M., & Kelleher, C. (2019). Towards Validation of a Model of API Learning. *2019 IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC) (pp. 267-269). Memphis, TN, USA.

Watson, C., & Li, F. W. B. (2014). Failure Rates in Introductory Programming Revisited. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 39-44).

Yen, C. W., & Wang, T. I. (2017). Using Self-Explanation and Ontology for Providing Proper Feedbacks in a Programming Environment. *The 6th IIAI International Congress on Advanced Applied Informatics* (IIAI-AAI) (pp. 585-590).

Zhi, R., Price, T. W., Marwan, S., Milliken, A., Barnes, T., & Chi, M. (2019). Exploring the Impact of Worked Examples in a Novice Programming Environment. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 98-104).

AUTHOR BIOGRAPHIES

Carlos Sandoval-Medina is a doctoral candidate in the Information Systems Department at the Autonomous University of Aguascalientes, México. He is currently a full-time Ph.D. student, a scholarship recipient of the CONAHCYT postgraduate program. His Ph.D. thesis involves improving learning in introductory programming courses by applying the effects of cognitive load theory through a proposed roadmap model addressing student motivational issues.

Carlos Argelio Arévalo-Mercado received the Ph.D. degree in Exact Sciences and Information Systems from the Autonomous University of Aguascalientes (UAA), Mexico in 2010. He was the Head of the Information Systems Department at UAA, from 2014 to 2020 and is currently a full-time Research Professor. His research interests include the design and development of learning technologies and teaching methods based on learning theories from cognitive sciences, particularly in introductory programming. He is a frequent collaborator with European universities on multinational projects with Erasmus+ funding. He is a member of the National System of Researchers (SNII) of CONAHCYT, Mexico.

Estela Lizbeth Muñoz-Andrade obtained her Ph.D. in Exact Sciences and Information Systems from the Autonomous University of Aguascalientes (UAA) in 2010. She is currently a full-time Research Professor at the Department of Electronic Systems at UAA. She has been teaching for 25 years in the areas of programming fundamentals and data structures. Her area of research is the development of educational applications for learning programming. She is a member of the "EGEL+ D-I COMPU" of the Academic Committee at CENEVAL, México, where she was the coordinator of the Programming Languages Academy from 2004 to 2017. She was the Head of the Electronic Systems Department from 2011 to 2017. Currently, she is Secretary of Undergraduate Teaching of the Basic Sciences Center of UAA.

Jaime Muñoz-Arteaga is a full-time professor at UAA (Universidad Autónoma de Aguascalientes) in Aguascalientes, México. He holds a Ph.D. in Computer Science from the University of Toulouse, France. He oversees the Computer Science area of the Ph.D. in Applied Sciences and Technology. He has experience as leader of several research projects related to the digital divide, human-centered design for interactive systems, and inclusive education. His research topics are in human-computer interaction, e-learning, and artificial intelligence. He has published several books, two on software engineering, two on human-computer interaction, and two on learning object technology.

ISSN: 2574-3872 (Online) 1055-3096 (Print)