Case Report

Algorithm-Aided Design with Python: Analysis of Technological Competence of Subjects

Thiago Bessa Pontes [1],* , **Guilhermina Lobato Miranda [2]** and **Gabriela Caffarena Celani [3]**

[1] Center of Technological Sciences, Federal University of Cariri, Juazeiro do Norte 63048-080, Brazil
[2] Institute of Education, University of Lisbon, 1649-004 Lisbon, Portugal; gmiranda@ie.ulisboa.pt
[3] Department of Architecture and Construction, University of Campinas, Campinas 13083-970, Brazil;
     celani@unicamp.br
*   Correspondence: thiago.bessa@ufca.edu.br

check for updates

Abstract: Difficulties in learning computer programming for novices is a subject of abundant scientific literature. These difficulties seem to be accentuated in students whose academic choice is not computation, like architecture students. However, they need to study programming, since it is part of the new academic curricula. The results presented here are part of a PhD research, which investigates the achievement motivation and the acquisition and transfer of programming knowledge from an online environment designed on the basis of the 4C-ID instructional design model. These results are a sociodemographic analysis, and the technological competence of these subjects. We concluded that most of the students of our sample do not know how to auto assess their ICT expertise level, because they believed that they had sufficient computational knowledge for their needs. However, most of them told that they had difficulties creating codes. However, they recognized the importance of learning to program, thought it was valuable for architectural students, and felt motivated to acquire this new skill.

Keywords: algorithm-aided design; computer programming; python; technological competences

1. Introduction

There is an abundant literature on the teaching and learning of computer programming and the difficulties associated with them, especially in relation to those who are beginners in computer programming. These studies state that learning to program is a difficult task, involving specific knowledge and skills that many students have difficulty assimilating [1,2].

Some teachers in introductory programming courses try to find strategies that facilitate student learning [3] and even rely on languages such as Scratch and Alice that allow students to make fewer syntax errors [4,5]. The authors [6,7] say that the frequent mistakes made by students lead them to not want to continue learning to program, which leads many students to drop out of the courses. These difficulties seem to worsen in students whose academic choice was not computing, but another disciplinary area, and they have to study computer programming, since this component is part of the new academic curricula, as seen in some architecture schools. Well known offices with renowned names in the contemporary world of architecture, such as Zaha Hadid's, Norman Foster's, Santiago Calatrava's, Herzog and De Meuron's, among others, use computer programming in the design process of their projects.

This study is part of an ongoing PhD research that investigates the achievement motivation, acquisition, and knowledge transfer of computer programming by architects, by means of an online instructional environment designed on the basis of the four components instructional design (4C-ID) by the authors of [8]. In our case, we focused on the teaching of computer programming using the Python

language with architecture students, in a Portuguese public university and a Brazilian university. The objective of this study is to measure the technological competence of the subjects that are part of the sample of an experimental research. One of the hypotheses that we will test is that previous knowledge of programming acquired by architecture students influences the learning process.

1.1. Teaching of Computer Programming

There is a worldwide movement in favor of teaching computer programming to students from different fields and a belief that this learning should begin in basic education. Computer science is one of the required competences for the 21st century, and good programmers are required for many professional activities [9].

The challenges posed by the teaching of computer programming go beyond the barriers of time and technological evolution. From its inception, when the mathematician who authored [10] introduced the subject in the book "A discipline of programming", the need to find teaching methods for novices was started. At that time, programmers needed logical–mathematical knowledge to operate large computational machines, using mathematics as a communication language. The study of Reference [11] deals with the need to review the curriculum for teaching beginner programming, with "Learning to program = learning to construct mechanisms and explanations". In the work of the author [12], "Some difficulties of learning to program", the difficulties of learning to program were identified in specific areas. This author points out that students often have great difficulties in understanding all the issues related to running a computer program. Developing the computational thinking component, in which the student must abstract the solutions proposed on paper to the software, transforming it into computer code, is one of the first great challenges for beginners. According to Reference [12], one needs "a long time to learn the relation between a program on the page and the mechanism it describes." [12] (p. 59). He also affirmed that there should be a "notional machine", which simplifies the machine language so that all transformations of the computer program can be visible. However, the author of [13] began to use models focused on human learning difficulties in his work "Programming pedagogy: a psychological overview". This addresses the challenges of programming for beginners, grouped into three questions that reflect the difficulties presented by students: (i) The relationship between the understanding of the real problem and the generation of the computational solution; (ii) the knowledge acquired in relation to praxis; (iii) and the differences between the functional programming paradigm and the imperative and object-oriented paradigms.

1.2. Computer Programming in Architecture

Preparing professionals for the future is the role of any teacher, and the use of technology can be a motivator for this journey. In this way, the author of [14] affirms that: "It is in this field that the training of the architect falls far short of what should be his ideal education. I argue that their advanced education in new technologies, in the areas of multimedia, computing and digital tools in general, are a very special asset" (Original quotation: É neste campo que a formação do arquitecto se encontra muito aquém do que deveria ser a sua formação ideal. Argumento que a sua educação avançada em novas tecnologias, nas áreas de multimédia, computação e ferramentas digitais em geral, constituem uma mais-valia muito especial.) (p. 5). The author of [15] reinforces this idea when she says that "programming can improve logical reasoning and conceptual thinking in design. My conclusions are drawn on the historical development of CAD software, on pedagogical experiences with children and students of architecture and, finally, on some recent applications of programming in architectural projects" (Original quotation: A programação pode melhorar o raciocínio lógico e o pensamento conceitual no design. Minhas conclusões são tiradas sobre o desenvolvimento histórico do software CAD, sobre experiências pedagógicas com crianças e estudantes de arquitetura e, finalmente, sobre algumas aplicações recentes de programação em projetos arquitetônicos.) (p. 2).

In the development of architectural projects, the use of computer programming concepts for parametric modeling can improve and assist the architect by facilitating his or her work, allowing

him or her to create complex models by changing numerical variables, in a quick and efficient way, as asserted by References [16] and [17]. The authors of [18] agree with this idea when they say: "focused on the architectural field, traditional forms of expression and communication such as the printed panel or the physical model are being complemented and even replaced by the use of all kinds of ICT tools: from advanced virtual simulations or visualization through augmented reality of superimposed models with both real and virtual information, up to the already classic photo montages in compositional panels, the multi-format visualization of CAD files (Computer Assisted Design) and more recently its evolution in the BIM (Building Information Modeling) formats" (Original quotation: Centrados en el ámbito arquitectónico, las formas de expresión y comunicación tradicionales como el panel impreso o la maqueta física se están viendo complementadas e incluso sustituidas por el uso de todo tipo de herramientas TIC: desde las avanzadas simulaciones virtuales o la visualización mediante realidad aumentada de modelos superpuestos con la información tanto real como virtual, hasta los ya incluso clásicos montajes fotográficos en paneles compositivos, la visualización multi-formato de ficheros CAD (Computer Assisted Design) y más recientemente su evolución en los formatos BIM (Building Information Modeling)") (p. 2).

When talking about computer programming for architects, we should be aware that it is specially targeted at computer-aided design (CAD). According to Reference [15], in the last 40 years of the development of CAD, its original purposes were almost lost. Few people know, or many have forgotten, but the theoretical bases of the computer-aided design applied to architecture are closely linked to the design methods movement, which started being developed in England and later in the United States in the 1960s [15] (p. 67). The use of computer programming for CAD does not come from new technologies with great processing and rendering capabilities, but from a time when the use of computers was being proposed to help human creativity to solve issues related to design.

The design methods movement values were: Divergence "that sought to create new understanding (scope of the problem) towards better design solutions"; "transformation" redefining specifications of design solutions that can lead to better guidelines for traditional and contemporary design activities; "prototyping" and the study of possible scenarios for the best design solutions; "sustainability" by managing the process of exploring, redefining, and prototyping design solutions continuously over time; and finally, "articulation" with the visual relation between the parts and the whole [19]. We need to develop expertise to handle vector software tools to get the best out of CAD tools, and we also need to develop the ability to understand that this software has limitations. Therefore, the programming apprentice must combine the technical component with that of creativity to develop appropriate and interesting projects. The author of [20] states that architects, during project design, use representations to test possibilities. These representations take forms such as: Sketches, mental maps, models, and volumetric studies, among others. It is unquestionable that these tools are indispensable in the design of architectural projects, giving opportunities for professionals to create, design, and study construction aspects using virtual models.

Researchers such as the author of [21] claim that "Parts of the design process, such as the routing of the hydraulic and electrical systems that were once performed through extensive physical mock-ups are now developed electronically". [21] (p. 61). One of the great precursors of the use of computer science in architecture was Ivan Sutherland, who, in the 1960s, created Sketchpad, shown in Figure 1.

Figure 1. Ivan Sutherland using a "Sketchpad". Source: [22].

According to Reference [23], "in 1960, Ivan Sutherland, a brilliant student supervised by Claude Shannon (inventor of the theory of information that forms the foundation of digital communications), used the combination of the TX-2 and the light pen to create the seminal "Sketchpad" program. Sketchpad let a designer sketch shapes, which the computer would then turn into precise geometrical figures. It was the first computer-aided program and remains one of the most expressive ones." [23] (p. 41). In his seminal paper "The theoretical foundation of computer-aided architectural design", the author of [24] argues that the sophistication of a project depends on the development of a production unit with a computer–graphics interface.

1.3. Computer Programming in CAD Tools

There are two types of programming languages that are used in CAD modeling software: Visual programming (VPL) and textual programming (TPL).

In this study, we approach textual programming, since it gives the user a greater breadth in relation to the possibilities of creation, not being limited to the default tools offered by CAD software packages, but increasing the possibilities through the use of scripts. Currently, most of the CAD and 3D software packages used by architects and designers incorporate a scripting language and were also used as introductory tools to teach coding from scratch. The author of [25] identifies the different uses of computer programming, such as productivity aids and exploratory code, as well as describes his teaching experience using scripting tools embedded in CAD software.

The TPL chosen for this study is Python because, according to Reference [26], there is a strong tendency to use Python as a language incorporated in programming tools. For the author of [27], students who are new to programming languages may feel frustrated, given the complexity to understand the syntax rules of this kind of languages. However, they affirm that textual languages, such as Python studied here, extend the possibilities of implementing new generative strategies with a higher degree of complexity while, by contrast, visual languages are limited.

The authors of [28] reinforce the need for beginner students to learn some programming language, so that they have some kind of autonomy in developing general tasks. They claim that the use of

"ready-made" software is not discarded in future work. However, an active experience in computer programming acquired in the formative years will give them the possibility of a more assertive choice.

1.4. Instructional Design and Technologies

In dealing with instructional design and technology, we should indisputably cite the study of Reference [29], which defines two types of knowledge and two types of learning. The first is the primary biological knowledge that deals with natural learning, one that does not need to be directly taught/instructed. The second is secondary biological knowledge, which deals with learning by teaching/instruction, which is directly related to the object of this study—Python teaching/instruction for use in CAD. For the authors of [30]: "instructional design and technology encompasses the analysis of learning and performance problems, and the design, development, implementation, evaluation and management of instructional and non-instructional processes and resources intended to improve learning and performance in a variety of settings, particularly educational institutions and the workplace." [30] (p. 53). The learning models that were adopted for the instruction of the programmatic contents addressed in this investigation were the Cognitivist Models, and among them the four components instructional design model (4C-ID Model) of Reference [8], which is appropriate to the complex learning that we analyze here.

The 4C-ID instructional design model was developed to teach knowledge, skills, and attitudes in an integrated way, and to train students to be able to transfer these aptitudes to real life problems [31]. The model admits that real life tasks can be used to create learning situations. Thus, this model, when properly used, facilitates the acquisition of knowledge for complex learning. According to Reference [32]: "The term complex should not assign the connotation of complicated or difficult, since these concepts involve performing tasks with difficult resolution, while the concept complex, in the context of learning, refers to the integration of acquired (and new) knowledge, skills and attitudes about a particular area of study (e.g., sciences, information technology, law, etc.)" [32] (p. 1). In this way, we understand that in this study, the "attitude" component treated in the definition of complex learning is school motivation, namely achievement motivation, based on the theory developed by [33].

2. Methodology

To achieve this objective, a methodology of experimental research was chosen. We observed two classes: One from the undergraduate degree in Architecture of the *Instituto Superior Técnico (IST)*, at the University of Lisbon, in the 2017/2018 academic year, and the other from the postgraduate program in Architecture, Technology, and the City, of the Faculty of Civil Engineering, Architecture, and Urbanism, at the State University of Campinas, in a winter course held in July 2017.

For data collection, we used several procedures and instruments. In this research, we will only mention the results of: (i) A demographic characterization of our sample; (ii) and a questionnaire on the technological skills of the participants.

All the students in the two samples were exposed to an online learning environment, based on the 4C-ID model of Reference [8], based on cognitive load theory [34] and the cognitive theory of multimedia learning [35]. Based on the study of Reference [36], the curricular components in which students present the greatest difficulty in learning Programming (TPL) + CAD are, in particular, "Lists" and "Functions". Thus, they were chosen for the experiment.

Each class was divided into two groups, which gave rise to four subgroups: Two in Portugal (G1 and G2) and two in Brazil (G3 and G4). In the first phase of the experiment, the students of the subgroups G1 and G3 learned the curricular component of programming called "Recursive Functions" in Python, based on the instructional model 4C-ID. The other subgroups (G2 and G4) performed the same curricular activities using the conventional teaching method. In the second phase, the students from the G2 and G3 switched to the 4C-ID instructional method and developed the activities of the curricular component "Lists" in Python, to avoid the Hawthorne effect (see Figure 2). The data used in this study were collected in the pretest.
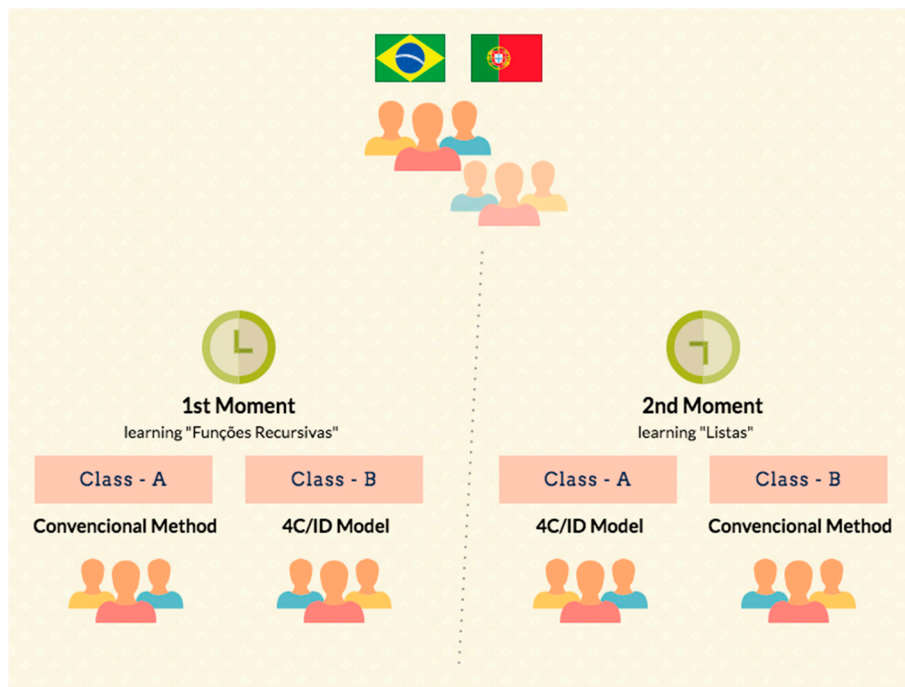
Figure 2. Infographic with the experimental procedures.

The groups that used the online environment were submitted to the experiment in a controlled environment, and all at the same time.

3. Finds

3.1. Sample

The sample covered Brazilian and Portuguese classes, during the academic year of 2017/2018. The Brazilian sample was hosted in the State University of Campinas (UNICAMP), and the Portuguese sample in *Instituto Superior Técnico*. The sample was tested with similar resources. The same online environment was applied, and the same teacher was present in both experiments, Brazilian and Portuguese. Both courses had the same aim: To teach algorithmic aided design for architecture students. In Portugal it was a regular master course, and in Brazil it was a special winter course for Bachelors, Masters, and PhD students.

All the students were exposed to the learning situation with the proposed model of teaching computer programming in Python, supported by the instructional design model 4C-ID. Thus, it is possible to control possible reactive effects, such as the Hawthorne Effect (Hawthorne's studies showed that worker productivity increased during their participation in an experiment, regardless of any experimental changes being introduced. Indeed, the workers altered their behavior because they knew they were being observed [37] (p. 354)), as we mentioned before.

The sample had 64 subjects: 19 in the first study with the Brazilian sample and 45 in the second study with the Portuguese sample. All the subjects were students regularly enrolled in the curricular units. They were all adults of both sexes, as shown in Tables 1 and 2.

Table 1. Presentation of subjects by gender.

	pcs	(%)	pps (%)	pos (%)
Women	41	64.1	64.1	64.1
Men	23	35.9	35.9	100.0
Total	64	100.0		

Table 2. Presentation of subjects by age.

	pcs	(%)	pps (%)	pos (%)
less than 20	5	7.8	7.8	7.8
from 20 to 30	50	78.1	78.1	85.9
from 31 to 40	7	10.9	10.9	96.9
from 41 to 50	2	3.1	3.1	100.0
Total	64	100.0		

It should be emphasized that the sample is a subset of the student population of the Faculties, and that the results and conclusions proposed here apply only to this sample and cannot be extended to the understanding of the universe as a whole. [38,39]

3.2. Characterization of the Sample

As previously reported, 64.1% of the sample (64 subjects) were female and 35.9% male, with a higher incidence of subjects in the 20–30 age group (78.1%), and unmarried individuals with 85.9% (see Table 3).

Table 3. Distribution of subjects by civil status.

	pcs	(%)	pps (%)	pos (%)
Married	3	4.7	4.7	4.7
Not married	55	85.9	85.9	90.6
Union of fact	6	9.4	9.4	100.0
Total	64	100.0		

For a clear understanding of the composition of the sample, we investigated the subjects' technological competence in the use of basic ICTs, and the daily use of computers, and the Internet.

We also found more specific questions, such as participation in online courses, the study of the textual language of computer programming, knowledge, challenges, and motivation.

When asked about computer use on a day-to-day basis, only one subject claimed not to use one, and of the remaining 63%, 85.9% rated use as "frequent use". When it came to the question of daily internet use, all subjects agreed they used it, and the place of use varied between: Home, work, public places, and university, illustrated in Table 4, for several uses such as: Fun, study, and work (Table 5).

Table 4. Internet usage locations.[1]

	pcs	(%)	pps (%)	pos (%)
at home	63	34.2	34.2	98.4
At work	26	14.1	14.1	40.6
In public places	36	19.6	24.8	56.3
In college	59	32.1	19.6	249.2
Total	184	100.0	100.0	287.5

[1] Group.

Table 5. Purposes of using the internet.[1]

	pcs	(%)	pps (%)	pos (%)
Fun	58	36.9	36.9	92.1
Study	60	38.2	38.2	95.2
Job	39	24.8	24.8	61.9
Total	157	100.0	100.0	249.2

[1] Group.

With regards to the question of daily time of internet use, 30% said they used up to 2 h a day, followed by 28% of students who said they used more than 4 h; 27% said they used up to 4 h a day. The other times of use can be seen together in Figure 3.
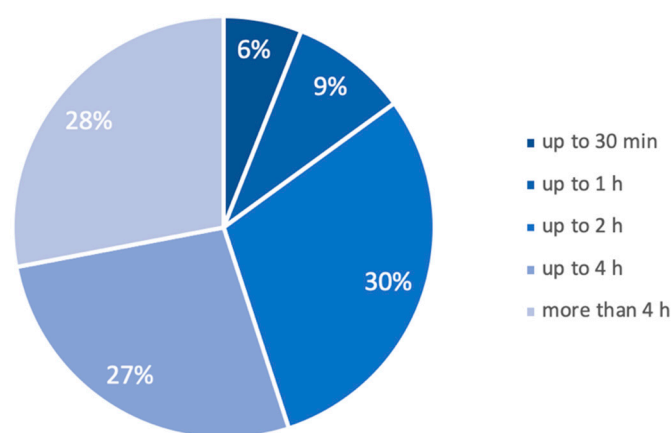
Figure 3. Daily time of internet usage.

Starting with questions that addressed more specific skills in the use of ICTs, we asked the students to make a self-assessment of their computer skills. Forty-seven subjects (73.3%) claimed to have sufficient knowledge for their needs, 9 (14.1%) said they had a lot of knowledge, and 8 (12.5%) reported difficulties in using computer resources, as can be seen in the Table 6.

Table 6. Self-assessment of computer knowledge.

	pcs	(%)	pps (%)	pos (%)
I have difficulties in using computer resources	8	12.5	12.5	12.5
I consider myself a user with sufficient knowledge for my needs	47	73.4	73.4	85.9
I consider myself a user with a lot of knowledge	9	14.1	14.1	100.0
Total	64	100.0	100.0	

This experiment used an instructional environment online. In this way, to understand the relationship of the participants with online learning environments, we asked some questions on this subject. Twenty one of the 64 participants had contact with online instruction; 16 out of 21 stated that the method is productive, and they learned easily, but 4 participants claimed that the experience was not very productive because they felt difficulties in studying through this modality, and only 1 subject said that it is an unproductive and not ideal method to learn.

We also asked the students if they had already had contact with textual programming languages. Fifty-one point six per cent of the students claimed that they had had contact with this type of programming languages, compared to 48.4% who said they had not. Therefore, we asked those students who had had contact with TPL to make a self-assessment of their knowledge. Twelve students (36.4%) claimed to be able to create codes with a lot of effort, 10 (30.3%) created codes with some

effort, 8 of them (24.2%) claimed that they had great difficulties, and only 3 (9.1%) did not have great difficulties in creating their codes, as shown in Table 7.

Table 7. Self-assessment in programming language.

		pcs	(%)	pps (%)	pos (%)
valid	I do not have much trouble creating codes	3	4.7	9.1	9.1
	I often create codes with effort	10	15.6	30.3	39.4
	I often create codes with a lot of effort	12	18.8	36.4	75.8
	I have great difficulty creating codes	8	12.5	24.2	100.0
	Total	33	51.6	100.0	
Missing	0	31	48.5		
Total		64	100.0		

For 45.5% of the subjects, understanding the mathematics involved in problem-solving and the rules of languages was the biggest challenge. For 36.4%, the greatest difficulty was understanding the language rules because mathematics did not pose problems for them. Only 18.2% said that understanding the mathematics involved in problem-solving was the biggest challenge since the rules of languages were understandable (Figure 4).
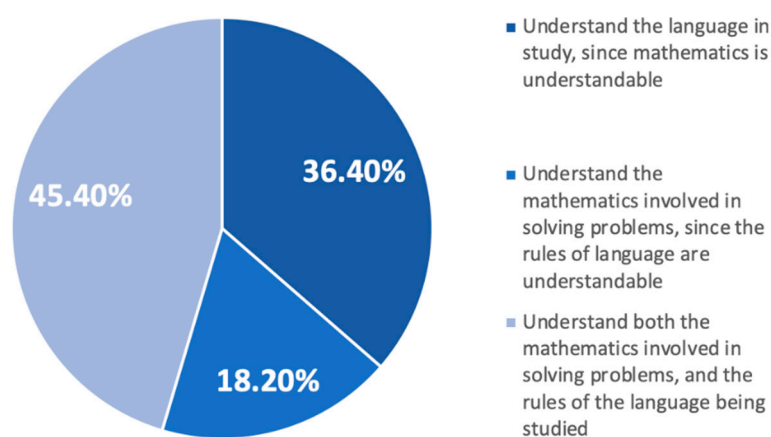
Figure 4. Challenges of learning textual programming language.

Finally, we asked the 64 participants of our study: "Do you feel motivated to study computer programming?", and half of the sample (51.6%) claimed to feel motivated, 31.3% highly motivated, and 17.2% poorly motivated, as can be observed in Table 8.

Table 8. Motivation to study computer programming.

	pcs	(%)	pps (%)	pos (%)
poorly motivated	11	17.2	17.2	17.2
motivated	33	51.6	51.6	68.8
very motivated	20	31.3	31.3	100.0
Total	64	100.0	100.0	

In summary: The vast majority of the subjects in our sample (90.9%) claimed to have difficulty creating codes, and that the challenges of this practice are associated with understanding both the mathematics involved in problem-solving and the rules of the language under study. We also observed that 82.9% of the students are motivated to acquire this competence, which validates the progress of our experiments in the search of specific constructs, like achievement motivation, self-directed learning, and perceived mental effort.

4. Discussion and Conclusions

Promoting learning that brings a clear meaning to the student, especially for new ones in a specific subject area such as computer programming, should be the first goal of all teachers. This study had as its main objective to perceive the technological competence of the subjects participating in an ongoing doctoral research, which intends to measure the achievement motivation, acquisition, and transfer of programming knowledge of architecture students while learning computer programming.

Architects need to develop skills to use digital technologies and computer programming. As students, they are forced to learn these competences, as they are part of the current curricula of most schools all around the world. As professionals, we think that more and more, they will be asked to apply these technological skills when they think and design their projects.

We concluded that most of the students of our sample do not know how to auto assess their programming expertise level, because they believed they had sufficient computational knowledge for their needs. However, most of them told that they had difficulties creating codes, although they recognized the importance of learning to program. They thought it is valuable for architectural students, and they felt motivated to acquire this new skill.

For future work, after collecting the data related to achievement motivation, self-directed learning, perceived mental effort, acquisition, and transfer of programming knowledge, we will perform the statistical inferential analysis of these data in order to determine the validity and reliability of the chosen instruments and test our hypothesis. Lastly, we expect to determine when and how to use the 4C-ID instructional design model to teach the Python programming language to beginner students in computer science for use in CAD tools.

References

1. Jenkins, T. On the Difficulty of Learning to Program. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.596.9994&rep=rep1&type=pdf (accessed on 8 November 2018).
2. Pea, R.D.; Kurland, D.M. *On the Cognitive Prerequisites of Learning Computer Programming*; Technical Report No 18; Bank Street College of Education, Center for Children and Technology: New York, NY, USA, 1984.
3. Qian, Y.; Lehman, J. Students' Misconceptions and Other Difficulties in Introductory Programming. *ACM Trans. Comput. Educ.* **2017**, *18*. [CrossRef]
4. Resnick, M. EdSurge: Learn to Code, Code to Learn. Available online: https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn (accessed on 9 November 2018).
5. Costa, J.M.; Miranda, G.L. Relation between Alice Software and Programming Learning: A systematic review of the literature and meta-analysis. Available online: https://doi.org/10.1111/bjet.12496 (accessed on 8 November 2018).
6. Guzdial, M. Learner-Centered Design of Computing Education: Research on Computing for Everyone. Available online: https://www.morganclaypool.com/doi/abs/10.2200/S00684ED1V01Y201511HCI033 (accessed on 13 November 2018).
7. McCracken, M.; Almstrum, V.; Diaz, D.; Guzdial, M.; Hagan, D.; Kolikant, Y.B.-D.; Laxer, C.; Thomas, L.; Utting, I.; Wilusz, T. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.* **2001**, *33*, 125–180. [CrossRef]
8. Merriënboer, J.J.; Kester, L. *The Four-Component Instructional Design Model: Multimedia Principles in Environments for Complex Learning*; Cambridge University Press: New York, NY, USA, 2005.
9. Tucker, A. *A Model Curriculum for K-12 Computer Science*; CSTA: New York, NY, USA, 2006.
10. Dijkstra, E.W. *A Discipline of Programming*; Prentice-Hall: Eindhoven, The Netherlands, 1976.
11. Soloway, E.; Ehrich, K. Empirical studies of programmer knowledge. *IEEE Trans. Softw. Eng.* **1984**, *SE-10*, 595–609. [CrossRef]

12. Boulay, B.D. Some difficulties of learning to program. *J. Educ. Comput. Res.* **1989**, *2*, 283–299. [CrossRef]

13. Winslow, L.E. Programming pedagogy: A psychological overview. *ACM SIGCSE Bull.* **1996**, *28*, 17–25. [CrossRef]

14. Beirão, J.N. Sobre o Ensino da Arquitectura e o Futuro Profissional do Arquitecto. O papel da Arquitectura nas Sociedades Criativas. Available online: http://www.jornalarquitectos.pt/pt/forum/cronicas/sobre-o-ensino-da-arquitetura-e-o-futuro-profissional-do-arquiteto (accessed on 8 Novemver 2018).

15. Celani, M.G. Teaching CAD programming to architecture students. *Gestão e Tecnologia de Projetos* **2008**, *3*. [CrossRef]

16. Santos, D.M.; Beirão, J.N. Generative Tool to Support Architectural Design Decision of Earthbag Building Domes. Available online: https://doi.org/10.5151/sigradi2017-083 (accessed on 9 November 2018).

17. Beirão, J.N.; De Klerk, R. CIM-S: A Design Grammar for Street Cross Sections. Available online: http://papers.cumincad.org/data/works/att/ecaade2017_155.pdf (accessed on 8 November 2018).

18. Fonseca, D.; Pifarré, M.; Redondo, E. Relación entre calidad percibida y afinidad emocional de imágenes arquitectónicas en función del dispositivo de visualización: Recomendaciones para su uso docente. *RISTI—Revista Ibérica de Sistemas e Tecnologias de Informação* **2013**, *11*, 1–15. [CrossRef]

19. Arruda, A.J.V. *Design & Complexidade*; Blucher: São Paulo, Brazil, 2017.

20. Marques, S.L. *Arquitetura e Cibercultura: O Olhar Telemático e Seus Desdobramentos no Processo Projetual*; USP: São Paulo, Brazil, 1999.

21. Kieran, S.; Timberlake, J. *Refabricating Architecture: How Manufacturing Methodologies Are Poised to Transform Building Construction*; McGraw-Hill: New York, NY, USA, 2004.

22. Dalako, G. History of Computers. Ivan Sutherland Using Sketchpad in 1962. Available online: http://history-computer.com/ModernComputer/Software/Sketchpad.html (accessed on 13 November 2018).

23. Gershenfeld, N. *Fab: The Coming Revolution on Your Desktop—From Personal Computers to Personal Fabrication*; Basic Books: New York, NY, USA, 2005.

24. Mitchell, W.J. The theoretical foundation of computer-aided architectural design. *Environ. Plan. B Plan. Des.* **1975**, *2*, 127–150. [CrossRef]

25. Burry, M. *Scripting Cultures: Architectural Design and Programming*; Wiley: Chichester, UK, 2011.

26. Do Villares, A.B.; de Moreira, D. *Python on the Landscape of Programming Tools for Design and Architectural Education*; Blucher: São Paulo, Brazil, 2017; pp. 207–211.

27. Celani, G.V.; Vaz, C.E.V. CAD Scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view. *Int. J. Archit. Comput.* **2012**, *10*, 121–137. [CrossRef]

28. De Oliveira, C.A.; da Alves, J.B. Ciências da Computação nos Cursos de Engenharia uma Proposta Pedagógica Inovadora. Available online: https://www.researchgate.net/publication/266587758_CIENCIAS_DA_COMPUTACAO_NOS_CURSOS_DE_ENGENHARIA_UMA_PROPOSTA_PEDAGOGICA_INOVADORA/citations (accessed on 12 November 2018).

29. Geary, D.C. An Evolutionary Perspective on Learning Disability in Mathematics. *Dev. Neuropsychol.* **2007**, *32*, 471–519. [CrossRef] [PubMed]

30. Reiser, R.A. A history of instructional design and technology: Part I: A history of instructional media. *Educ. Technol. Res. Dev.* **2001**, *49*, 53–64. [CrossRef]

31. Merriënboer, J.J.; Kirschner, P.A.; Kester, L. Taking the load off a learner's mind: Instructional design for complex learning. *Educ. Psychol.* **2003**, *38*, 5–13. [CrossRef]

32. Melo, M.; Miranda, G. Learning electrical circuits: The effects of the 4C-ID instructional approach in the acquisition and transfer of knowledge. *J. Inf. Technol. Educ. Res.* **2015**, *14*, 313–337.

33. McClelland, D.C.; Atkinson, J.W.; Clark, R.A.; Lowell, E.L. *The Achievement Motive*; Appleton-Century-Crofts: New York, NY, USA, 1953.

34. Sweller, J. Instructional design consequences of an analogy between evolution by natural selection and human cognitive architecture. *Instr. Sci.* **2004**, *32*, 9–31. [CrossRef]

35. Mayer, R.E. *Multimedia Learning*; Cambridge University Press: Santa Barbara, CA, USA, 2009.

36. Pontes, T.B.; Miranda, G.L.; do Santos, D.M. A programação de computadores para alunos de arquitectura: uma análise do uso da linguagem Racket para protótipos 3D. In *Digital Technologies & Future School*; Instituto de Educação da Universidade de Lisboa: Lisbon, Portugal, 2016; pp. 197–208.

37. Tuckman, B.W. *Manual de Investigação em Educação*, 4th ed.; Fundação Calouste Gulbenkian: Lisboa, Portugal, 2012.
38. Fortin, M.F.; Côté, J.; Filion, F. *Fundamentos e Etapas do Processo de Investigação*; Lusodidacta: Loures, Portugal, 2009.
39. Hill, M.M.; Hill, A. *Investigação por Questionário*, 2nd ed.; Edições Sílabo Lda: Lisbon, Portugal, 2009.