Programming for Architecture: The Students' Point of View

Rita Aguiar[1], Afonso Gonçalves[2]
[1,2]INESC-ID, Instituto Superior Técnico, University of Lisbon
[1]rita.aguiar@tecnico.ulisboa.pt [2]afogoncalves@gmail.com

The following paper presents a reflection on computational design education in Architecture schools. For approaching this subject, the specific case of the Programming for Architecture course taught at Instituto Superior Técnico - Universty of Lisbon is presented and analyzed through the students' point of view. The aim of the course is to focus on representation methods through programming, introducing the fundamentals of computational approach to architectural design. We will explain and discuss the subject teaching methods, the structure of the course and the school environment. Also we will express the students' opinion regarding the class organization, the contents of the program and the usefulness of programming, as well as suggestions for an improved strategy for teaching computational methods to Architecture students.

Keywords: *Programming classes, Students opinion, Learning methods, Rosetta, 3D modelling*

INTRODUCTION

The pace at which CAD tools have crept up on the workflow of architects and designers alike in later years has meant that the teaching of such tools are now an integral part in most, if not all, university curricula. However, the road to implementing these new tools in a pedagogical context, with many of them requiring different ways of thinking and problem-solving strategies, has not been without its setbacks and obstacles (Duarte, 2007). Divergent opinions are even greater on the subject of introducing subjects related to the field of Computer Science, leading to a discussion on the importance and to what extent programming courses support architecture students' way of thinking (Wurzer et al 2011).

The most logical way to understand the successes and failures of programs that have tried to implement this new approach to architectural design is to know how students feel about them. Knowing and understanding the way Architecture students are receptive to these relatively new concepts is an important step towards guaranteeing the success and future motivation of young designers in using these tools.

The aim of this study is, thus, to understand, from the students' point of view, the way such concepts are received by students, using for that purpose a specific case study observed at Instituto Superior Técnico (IST). IST is an Engineering institution in Lisbon where every year students from the Integrated Master's Degree in Architecture take a course on Programming for Architecture (PA).

The study tries to focus on structural aspects of teaching programming and computational design,

especially on the difficulties and expectations of the students regarding what they experience every year as part of their school curricula. Many issues reported by the students might often be more conjectural and/or too tied to the institutional nature wherein these subjects are taught so trying to keep the study focused on the structural issues behind their opinions was of major concern for us.

The students were approached with open informal questions during interviews in addition to a formal survey answered by 30 students. Based on their opinions and in the observation of what happened during the semester, we tried to do a synthesis of what we consider essential and more representative of what students think and feel. After this reflection on the case study, we present some suggestions of possible improvements that could contribute to motivate Architecture students to have more interest in programming and to use it in their future activity.

CASE STUDY

Programming Language

The programming language taught to students in PA is Racket, a general purpose, multi-paradigm programming language of the Lisp/Scheme family. The language is used in a variety of contexts such as scripting, general-purpose programming, computer science education, and research [1]. The students use Racket native programming environment - DrRacket, to write their programs with the aid of a specific library of pre-defined functions called Rosetta. With Rosetta, students have access to a multitude of functions that they can embed in Racket code to create and manipulate geometry in a CAD application, such as AutoCAD or Rhinoceros. Rosetta is a programming environment that supports multiple programming languages (known as front-ends) and multiple CAD tools (known as back-ends) (Lopes 2012).

Course subjects

During the theoretical component of this course, students are confronted with several concepts necessary for programming and creating relevant architectural projects. These include recursion, constructive geometry, understanding and using data structures, higher-order functions and parametric representation and processing of surfaces and curves. **Figures 1 and 2** are some examples of what students should be able to accomplish by the end of the semester.

Figure 1
Example of implementation of recursion and randomness behavior

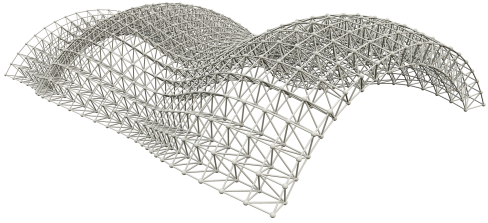


Figure 2
Example of processing data structures

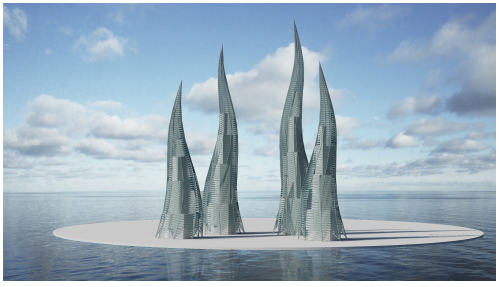


Figure 3
The final project of the 2014/2015 academic year.

Class structure

The PA course is taught weekly in three units of one-hour theoretical lectures, and one unit of two-hour laboratorial lectures.

The theoretical lectures are administered in the traditional method supported by interactive slides

containing demonstrations that result in the direct application of the subjects. The teacher explains the way the computer processes the code exhibited in the slides, helping students understand its execution.

The laboratorial lectures are administered at one of the university's computer labs where students are challenged to solve exercises related to what they learned during the theoretical lessons. A detailed guide with illustrations of the intended results is given to students. The semester ends with a final exercise where students are paired and asked to model a parameterized version of a well known building, previously chosen by the students themselves. **Figure 3** is an example of this exercise

STUDY RESULTS AND FIRST IMPRESSIONS

Raised interest and experienced difficulties

The conducted study showed that, generally, students were impressed with the potential of programming. As observed in **Figure 4**, most students showed great interest on the covered subjects when asked about the "interest that the subject raised" on a scale of 0 to 10.

On **Figure 5** we also notice that students understood and accepted the importance that programming can have on the Architecture curricula.

However, the results of **Figure 4** show a greater dispersion than **Figure 5**. It is possible that this has to do with the fact that students expressed some initial difficulty in using an algorithmic language that is described through textual code with demanding syntactic and semantic rules. These difficulties were well expressed by the interviewees, including the students with higher grades. This is increased by the fact that most of them had no prior contact with such subjects: in 30 students, only 7 had previous contact with programming. This leads them to show some difficulty in developing the ambiguity-free thinking perspective that is required for working with computers.

Figure 4
Level of interest showed by the students

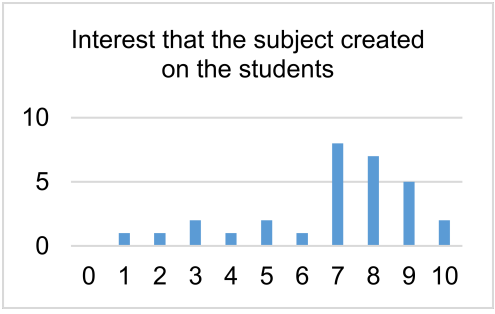


Figure 5
Importance of programming in architectural curricula

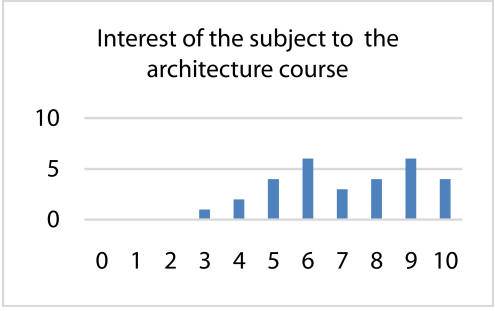


Figure 6
Difficulty in assimilating the concepts

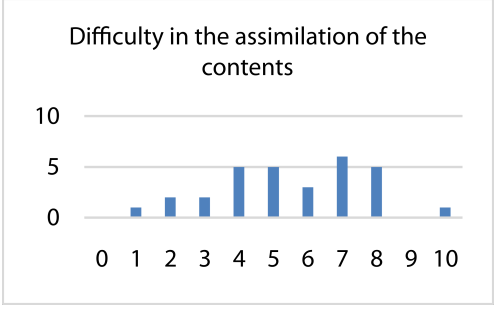
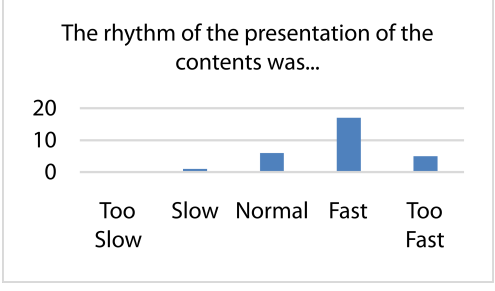


Figure 7
Rhythm of the presentation of the contents



On **Figure 6** we can observe that the answers collected do not seem to demonstrate that there has been a great difficulty assimilating the concepts taught in class but again we notice some dispersion in the answers.

The interpretation of this result might be on what students understand as assimilation. If they are thinking in the context of the theoretical lectures, then the clarity with which the teacher exposes the issues might be an important factor that leads to a more facilitated assimilation of the contents, not necessarily meaning students do not find them difficult.

Understanding what the teacher says does not mean that the knowledge is internally understood and interiorized since this can only occur when the information is organized and processed according to the mental structure of the student.

This is where difficulties can appear. According to what we observed, in the beginning there were not major difficulties although it is a matter that generally students do not consider to be easy. The exercises were simple and the results of the first test were reasonable.

The difficulties started to appear in exercises that involved the use of more elaborate and demanding concepts having a great demand in terms of developing a greater analytic capacity in interpreting what is being asked and also in finding strategies for problem solving. Given the way the question was asked, the possibility of bias in the matter of a psychological issue of personal self-esteem must be also taken into account.

According to **Figure 7**, when we asked if students found the course timeframe appropriate, the results show the general opinion that the program is too extensive for the class time. This, then, requires an accelerated rhythm that, for many students, does not provide enough time for developing and integrating the information received in class. The consequences may lead to students feeling discouraged to keep attending class, especially considering how demanding and time consuming the rest of the Architecture degree is. If for whatever reason a student skips a lesson, it is much more difficult for him to follow the next classes.

Students expressed a great emphasis on the importance of practical classes since they feel the need to interact with the computer in order to verify the effect of the code they elaborate.

One of the obstacles encountered regarding the final project was how to manage time efficiently and apply the knowledge learned in class, as well as knowing how to organize the programmatic content in a systematic way. Starting is where students feel the greater hardships since they have to take what is complex and simplify it to the point that it can be translated into code. There should also be a previously understanding of the geometry and mathematics involved in the project design. The elaboration of a more complex project benefits from the use of functions that are introduced at latter stages of the course.

If the students start thinking in the project early, they do not have the most important means to elaborate it. If they start it very late, then the time they have left is short. This resulted in the teacher giving a good help, presenting in classes the key functions for its elaboration.

When asked their opinion on "Should this subject be extended to another semester?", 12 students answered "yes" while 17 answered "no". Naturally, their position towards this question is dependent on many factors, such as the interest that the subject provoked on them, the difficulties they felt and the perception of the importance that this will have in their future careers. According to the interviews there seems to be a consensus that if the subject was divided in two semesters the course would be much easier to follow, especially in regards to the final project which could be done in a second semester.

Opinions on the impact of programming in architectural design

In general, students were deeply impressed by the subject of programming, to the point where they look at design in a new perspective.

Students realized the potential of programming in architectural design, especially in the extent to which with a few lines of code they can draw shapes that are intangible by either their manual abilities or the restrictions imposed by CAD tools (Leitão et al 2010). The ease with which, through variation of function parameters, students can produce variations of a model is also a very helpful feature when searching for the ideal form to their designs. But here the students' impression is that programming is only useful when designing and drawing monumental, complex and exotic buildings. Students doubt that programming can be useful for the type of work which has to be elaborated during the Architecture degree.

Potential use in other subjects

One of the aspects to which students expressed great interest was the integration of what they learn at PA and the other courses they attend, especially with Architecture Project. In 30 students inquired, 20 answered positively to this integration. Obviously this requires that professors in charge of these courses show acceptance and willingness to having students make use of programming in their classes. But the reality shows that many are used to a more traditional modus operandi for architectural design and remain skeptical and not receptive to such integration. However, if the teaching of computer science subjects is to have any success in the Architecture curricula, then they must be tailored to the needs of the architects. Students learn best when they see a connection between what they are learning and their current or future needs (Duarte 2005).

The use of programming in other disciplines will have to come from students' own initiative regarding their own projects, but there is also a time pressure to consider in the equation. During the programming course, students learn how to elaborate small independent projects but they do not exercise their integration into broader projects. Students who had the subject recently do not yet feel confident enough to risk using methods that are still not commonly used on academic design disciplines. If students had the programming course a long time before, unless they continued their training with programming since then, they fear they will have forgotten much of what they learned, feeling even less secure to integrate this knowledge in their projects.

Use of programming in their future career as architects

Figure 8 synthesizes the answers to a question regarding the students' thoughts on using programming on their future Architecture projects.

The dispersion of the answers may reveal some degree of uncertainty regarding the perspectives while mode 5 may indicate a certain hesitation in the answers. The mean value is 5.2 while the median is 5.

Through informal talks with the students we could conclude that, generally, students feel that the way Architecture is practiced in Portugal does not create high expectations for the use of programming, while in other countries, several studios already seek professionals with programming skills.

We also concluded from the interviews that there were students interested in the possibility of programming for other backends such as Revit and ArchiCAD because sometimes AutoCAD is simply unfeasible. The team behind the development of Rosetta has already understood this importance and is actively working on having this integration.

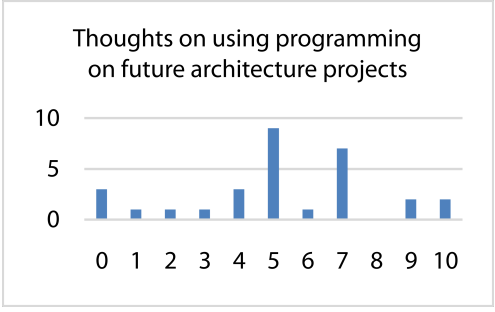


Figure 8
Thoughts on using programming on future Architecture projects

SUGGESTIONS AND REFLECTIONS ON THE RESULTS

The focus of academic education is the student, hence the importance of hearing what students have to say. It is crucial to understand the educational background in order for a teacher to employ the most appropriate methods and communication strategies. With the proper methodology students feel more interest in further extending their knowledge and skills.

What follows now are a set of suggestions that the authors consider to be relevant in addressing the issues presented previously taking into account the profile of Architecture students.

Program and teaching methods

Initial motivation. We consider of great importance that, from the very beginning, students be amazed with the potential of programming. Consequently, in order to motivate students to become interested in programming, the teacher should find ways to captivate the students' interest, probably through images of great visual impact during the very first lesson, and showing how few lines of code can produce surprising results, and give examples of the powerful effect that parameter variation can have in the resulting image and its usefulness in the search of the desired shape.

Type of lectures. Considering that proficiency in programming requires practice it seems more suitable to have a mixed system of theoretical-practical classes, instead of the traditional method of entirely theoretical lessons separated from practice. Students understand programming concepts better if they have the opportunity to test their programs right away.

The suggestion is not to undermine the importance of theory but rather understanding the benefits of interconnection with practice. Switching between more expository stages and more experimental stages will help keep the interest of students and motivate them at later stages.

Time management. It is important to tailor the programmatic content of a programming course to the time students have available. This adjustment can be made in extent and/or in depth.

With respect to extent, without undermining a general knowledge of all the capabilities of programming, there could be a selection of what is more important in Architecture use. In what concerns depth, the best solution would be beginning with very simple examples and deepening with progressively more complex examples till the available time for the theme is reached. Thus, the depth attained would depend on the ease with which students are learning.

Presentation of the various themes in parallel to the conception of a more elaborate project. During classes, students learn through small exercises the various components of the program.

It could be more beneficial if, from the beginning, students were presented with a more extended exercise instead, that also made sense as an architectural whole, especially conceived for this purpose. This exercise could incorporate the various concepts that are to be learned through the subject. This means that some of the smaller exercises executed during the semester could be directly incorporated in this project. This way, students could be taught from the beginning about how to approach a more complex exercise in addition to preparing them for the final project.

Selection of exercises oriented to a future use. The study that we conducted shows that students do not feel very encouraged to use what they learned in their works on other project subjects.

One specific exercise the students are shown during class is how to take an urban site plan (Figure 9), provided by municipal services or topographers, and develop the equivalent 3D version with the aid of programming. This involves cleaning spurious data from the drawing and extracting relevant geometry information from a CAD application to use it in the programming environment.

After a study of what students normally do in these subjects, some of the exercises in practical classes could be conceived so that they could be applied in the project subject.

Figure 9
Exercise of taking a 2D site plan and turning it into a 3D model. Downtown of Lisbon City.

From the visual to the abstract. During the last decade, it has been discussed how far programming courses support the architecture students' way of thinking (Wurzer et al 2011). It has also been reported that Architecture takes on a much more problem-oriented approach to design than the algorithmic thinking behind programming (Lawson 2005). What seems certain is that Architecture students have a very strong visual mindset so focusing on visual communication during the teaching process should be privileged. The visual representation may even be the starting point for understanding more abstract concepts, as illustrated farther in connection with the concepts of recursion and higher-order functions.

Regarding what has been mentioned, we think that for an Architecture student it is preferable to start presenting the concept in a visual form first, using for that purpose a simple example that is more relatable to Architecture rather than presenting complex or purely mathematical examples.

Simple examples have the advantage of capturing the maximum attention of the student on the new concept, saving them the effort to understand other complex aspects of the example. This could eventually constitute a kind of background noise and prevent the student from looking clearly at the new concept.

Sequence of the presentation of the contents. The suggestions we present in this section take into account what we proposed before and are intended for a type of course where students use a front-end programming language where they can embed functions from a library that produces objects in a CAD tool as backend.

After the initial motivation we referred before, students could have their first contact with the computer by means of elaborating a simple program in which they would experiment and use the programming language and functions from the library to generate a simple image in the CAD tool. Then, after the students had elaborated this, the teacher could explain, in a schematic way, the architecture of the software being used.

In connection to this, the teacher could begin teaching the language in a more formal way. When appropriate, students can execute examples and exercises in the computer improving the comprehension and helping maintaining a good mood.

We think there is an advantage of presenting functions and the associated concepts from the library as soon as possible to be used in more visual examples that may help students to understand more formal and abstract concepts of the language.

During the semester, the choice of contents and the way they are presented should take into account that this course is aimed at Architecture students and not Computer Science students. Although many efforts have been made in the past to do so, we consider that this matter could still be improved.

Concepts more difficult to achieve. As previously stated, several powerful concepts that we commonly use in programming, such as recursion and higher-order functions, are more difficult to assimilate, considering that they represent a new way of thinking for most students.

In the case of recursion, the immediate perception of the spectacular effect and usefulness of its use in solving architectural problems would facilitate a quicker apprehension of this concept by the student. The concept of higher-order functions is more difficult to grasp and so more time and observation of new useful examples have to convince the student of its potential.

Recursion. In order to contextualize the matters being discussed, let us look at the concrete example of a simplified drawing of a staircase. The intention is to draw the representation of this staircase with n steps, starting at a point P and with a riser length of r and a tread length of t. The recursive construction can

be well understood according to figure 10 retrieved from the teacher's book.

In reality, the concept of recursion is visually well represented in the figure which translates verbal language describing the process of drawing an n-step staircase the same as drawing one step first and then based on it draw a staircase with n-1 steps.

The representation of the algorithm and its implementation and coding should follow, so that the students feel the effect immediately, in particular the variation of the parameters.

Higher Order Functions. Before presenting it in an abstract way, the concept could be apprehended in an experimental way through the elaboration of a sequence like the following:

The teacher asks the students to define a function that, given a point, creates a cylindrical single column of certain dimensions centered at point P.

Next, the teacher asks the students to define a function that uses the previous one, but not given as argument to this new function, and creates an alignment with a certain orientation of n cylindrical columns at a certain distance from each other.

The execution of the program for 5 columns would produce an effect as can be seen in Figure 11.

Continuing, the teacher would then ask the students to repeat the same steps but for creating prismatic columns. The execution would produce the effect in Figure 12.

Finally, the teacher would present the solution of a single function capable of producing the same result as before but with the particularity that it would receive as argument any of the two previously defined functions to create a single column as opposed to having to use two different functions to produce alignments, one for each type of column.

This way, and simultaneously with the apprehension of the concept, the students would realize the benefits of the use of higher order functions.

Next, a very simple mathematic example could be showed like the following.

Let us define a function named shift1 that receives as argument a real function of real variable and produces as image, another function whose graph is obtained from the graph of the first function by a vertical translation applied by the vector (0,1).

The visual illustration of the function shift1 applied to the function square would be such as in Figure 13.

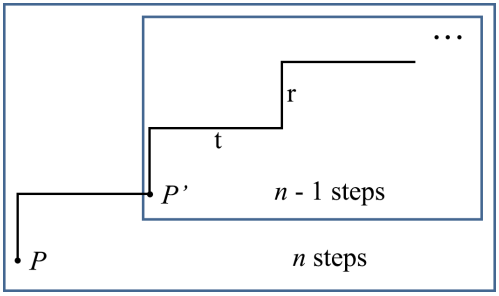


Figure 10
A simplified stair and the recursive process of how it is drawn.

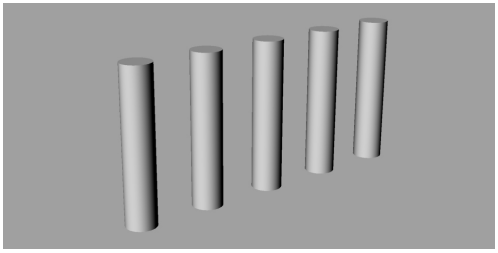


Figure 11
A series of cylindrical columns

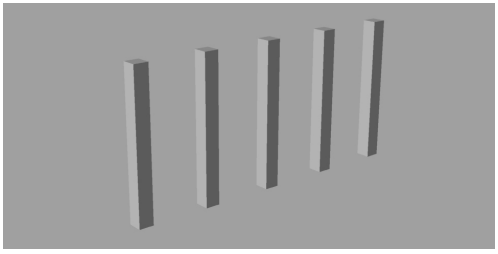


Figure 12
A series of prismatic columns

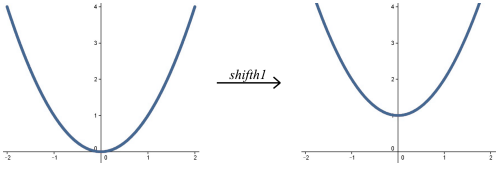


Figure 13
Visual representation of the function shift1 applied to the function (1)

A very suggestive description in mathematical language is:

$$(f \rightarrow (x \rightarrow f(x) + 1)) \quad (1)$$

This type of description could also be used to visually illustrate the semantics of the execution of commands of the given language.

As an example, and to illustrate the semantic behind Racket's code execution, the usage of the function shift1, when given as argument the function square and the result applied to the value 5 could be suggestively presented as in Figure 14.

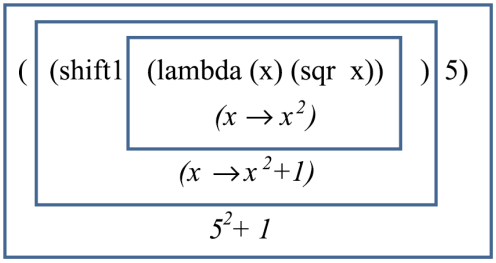


Figure 14
Visual representation of the semantics of the use of shift1 in Racket language

Continuation of the theme in new courses

The students' general opinion is that one semester is not enough time for the extent and level of demand in the course's program, and if it was divided and complemented in two semesters it would be more accessible. Yet, when confronted with the prospect of an additional semester, 12 students expressed a favorable opinion and 17 unfavorable.

This can be explained by the fact that, according to what we have stated throughout this work, students show interest in regards to programming but the difficulties experienced and the unmet expectations do not motivate them to go further. If that is indeed the case, then we can legitimately assume that the reorganization of the course's program into two semesters may allow the first semester to be much less overwhelming, more attractive and could motivate students to continue in a second semester.

Envisaging a scenario of a programming course divided in two semesters, a more introductory but motivating first semester could prepare students for the second semester. Throughout the second semesters, students would then elaborate a project using the proper methodology, possibly even exploring the application of programming in other areas of their curriculum. A second semester could even be optional as not to force students, who are clearly not interested in the theme of programming, to continue.

The question is whether the proposal to add a second semester would, in this context, be welcomed by the body responsible for coordinating the Architecture course.

Integration in other disciplines of learned methods

Finally, it is very important to settle for convincing those responsible for architecture courses and faculty the importance of increasing programming nowadays and especially in the future so that there is an openness and willingness to reinforce learning these methods and to introduce its use in other subjects.

CONCLUSION

A study of a particular case was presented. All the information regarding this study was based on the experience of students that attended the PA course during the 1st semester of the school year of 2014/2015, at Instituto Superior Técnico.

We concluded that, although students expressed their interest in programming and accepted the importance that programming can have on the academic Architecture curricula, they felt it is unlikely they would continue using and integrating what they learned in other subjects or in their own design projects. This is due to many factors among which we consider most relevant the lack of applicability the subjects have on the students' work in other courses, the difficulty in managing the time needed for elaborating the final project, and the lack of encouragement shown in other courses in trying to integrate new digital design tools in their educational

programs.

We hope this study provides useful reflections on this subject and that will contribute with relevant suggestions geared towards an improved strategy for teaching computational methods to Architecture students. For a broader view on teaching methods and other stories of success and failure this study would need to be expanded to other contexts. As for future work, it seems pertinent to approach students from other backgrounds, other educational institutions and possibly even professionals who make daily use of these technologies in their practice. It may also prove useful to compare the results of our studies with other similar analysis that have already been developed in other institutions.

ACKNOWLEDGMENTS

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013, and by the Rosetta project under contract PTDC/ATP-AQI/5224/2012.

We thank our colleagues who contributed to the elaboration of the survey and attended the interviews, although they may not agree with all of the interpretations/conclusions of this paper. We would also like to thank Ana Rita Santos and Sandra Monteiro for their help in the revision phases.

REFERENCES

Celani, G and Vaz, C 2012, 'CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison From a Pedagogical Point of View', *International Journal of Architectural Computing*, 18, pp. 122-137

Duarte, J. 2005 'Towards a New Curricula on New Technologies in Architecture', *Giaconia, P. (ed.), Script: Spot on Schools, Editrice Compositori*, pp. 40-45

Duarte, J. 2007 'Inserting New Technologies in Undergraduate Architectural Curricula', *Predicting the Future 25th eCAADe Conference Proceedings*, Germany, pp. 423-430

Lawson, B 2005, *How designers think: the design process demystified*, Architectural Press, Burlington

Leitão, A. 2013 'Teaching Computer Science for Architecture', *1ST eCAADe Regional International Workshop*

Leitão, A., Cabecinhas, F. and Martins, S. 2010 'Revisiting the Architecture Curriculum: The programming perspective', *FUTURE CITIES [28th eCAADe Conference Proceedings*, Switzerland, pp. 81-88

Leitão, A. and Proença, S. 2014 'On the Expressive Power of Programming Languages for Generative Design - The Case of Higher-Order Functions', *Proceedings of the 32nd eCAADe Conference*, England, pp. 257-266

Lopes, J. 2012, *Modern Programming for Generative Design*, Master's Thesis, Instituto Superior Técnico/University of Lisbon

Wurzer, G., Alaçam, S. and Lorenz, W. 2011 'How to Teach Architects (Computer) Programming: A Case Study', *29th eCAADe Conference Proceedings*, Slovenia, pp. 51-56

[1] http://racket-lang.org/