ICARUS Project - An Open Source Platform for Computer Programming in Architectural Teaching and Research

Odysseas Kontovourkis¹, Constantine C Doumanidis²
¹Department of Architecture, University of Cyprus ²School of Electrical & Computer Engineering, Aristotle University of Thessaloniki
¹kontovourkis.odysseas@ucy.ac.cy ²kdoumani98@gmail.com

This paper, presents an ongoing work entitled ICARUS, an abbreviation for 'Integrating Computerized ARchitecture with USers'. The aim of this work is to develop an open source platform for computer programming implemented in architecture, for teaching and research. In particular, the platform provides the framework for a simplified and user friendly textual programming methodology for the needs of our architectural institution. It consists of several modules like coding, plug-in and repository development, targeting to be publicly available in the future. The platform is created based on the Python programming language, which is run in Grasshopper, a plug-in for Rhino 3D. In the first phase of ICARUS development, several case studies within the framework of a postgraduate course are conducted, aiming at providing an overview of its potentials, limitations and generally, its impact on establishing a useful methodology for algorithmic thinking among students with little or no prior computer programming skills.

Keywords: Computer programming, Open source platform, Parametric design, Plug-in development, Algorithmic thinking

INTRODUCTION

In a rapidly growing community of users and open source plug-in developers of parametric design tools, the knowledge in regard to the visual and textual programming languages (Leitão and Proença 2014), has become a more than important skill among students and professional architects. In order to face this new reality, relevant skills and competences were gradually introduced in curriculums of architectural education (Terzidis 2006; Burry 2011).

Within this framework, parametric design software including Grasshopper, Dynamo, Generative Components, etc. has already been established and used extensively either in architectural schools or in practice. Nevertheless, the majority of students and architects have been mostly involved in the visual programming aspect of the process (Wurzer and Pak 2012) with the use of available components provided by the platforms or by the third party plug-in developers [1]. This, together with the necessity for understanding algorithmic thinking and visual programming principles, have been the areas that mostly dominated such investigation so far. As a consequence, textual programming as well as the development of tools and plug-ins remained at a premature level, mostly conducted by people having a back-
In addition to the above, the introduction of textual or computer programming in architectural education and practice is not something new, as it dates back to the work of architects-educators such as Frazer (1995) and Coates (2010). Also, a number of software has included textual programming languages, for instance MAXScript for 3D Studio Max, AutoLISP for AutoCAD, RhinoScript for Rhino, etc. Moreover, stand-alone applications combining textual programming and graphical interfaces like Processing (Reas and Fry 2007), together with traditional and well-known programming languages such as VB and C++ show the wide range of available algorithmic tools introduced in architecture.

In all the above cases, the involvement of architects has been growing and expanding considerably; however, a full and satisfactory level of engagement has not yet been achieved, due to a number of reasons. To name a few, these might include the fact that students do not have a relevant background; computer programming courses are not offered by most schools of architecture; there is less interest among students and professionals; it is an undesirable subject of research; there are difficulties in understanding and implementation, etc. In order to achieve a more thorough and productive involvement of students in computer programming several projects have been conducted (Wurzer and Pak 2012; Ireland 2017).

Following similar objectives, the ICARUS project was developed in order to overcome obstacles and offer an architectural platform that might allow a more active involvement of students and architects in textual programming. The aim is to reconsider the role of textual programming in architecture according to the needs of students and academic staff of our department. Towards this direction, the project has been visualized as an open source platform based on a bottom-up logic where modules of code are combined to create overall algorithms. The major objectives of the platform are to teach students textual programming for complex design investigation in a parametric environment in an easy and understandable way (Celani and Vaz 2012; Kontovourkis 2012), to collectively gather information from the parametric design community and to encourage and make plug-in development accessible to the wider public.

The following sections starts by briefly describing teaching aspects of textual programming within the framework of a current postgraduate course in our institution, then the organizational structure of the suggested platform is briefly described, and finally a number of case study results are demonstrated and discussed.

TEACHING TEXTUAL PROGRAMMING IN GENERAL
Teaching textual programming to the students of our architectural institution has started since 2012 within the framework of the postgraduate course ARH-522 - Advanced Computer Aided Design Topics. This course is mainly focused on the advanced use of algorithmic design and physical computing as a continuation of the course on parametric design and digital fabrication given to the students during undergraduate studies. The specific postgraduate course offers the opportunity to the students to move beyond simple use of visual programming principles, allowing them to be involved in textual programming. Algorithmic approaches can be implemented in the development of interactive workflows based on users’ interaction with computational design platforms in order to allow investigation of optimised architectural forms and structures (Kontovourkis et al. 2016; Kontovourkis et al. 2018a; Kontovourkis et al. 2018b).

Driven by the need to deepen into the use of computational design tools at postgraduate level, the ICARUS platform aims at individuals who wish to gain an insight on algorithmic design thinking and on how this can contribute towards a more responsive architecture. Thus, by having as background the experience of parametric design and especially visual programming principles, students move a step further to deal with issues related to textual programming and algorithmic logic as well as its structure and
TEXTUAL PROGRAMMING IN A POST-GRADUATE COURSE

Textual programming has been introduced, together with two other modules, which formulate the overall content of the postgraduate course. These extra modules consist of physical computing experimentation and users-driven interactive workflows development, implemented in an integrated manner. Due to the nature of the course, which integrates three modules, leading to a capstone project, the modules are structured in classes, where different skills and competences are introduced to the students alongside with the development of the main project (Figure 1). It is important to note that in these classes prior knowledge of parametric programming skills (for example Grasshopper) is preferred but not required for the students to be enrolled. This makes learning objectives of textual programming even more challenging because students without prior knowledge should acquire multitude of knowledge in this field in a short time period.

Initially, Processing [2] has been introduced and taught as the textual programming language. At the end of the course, students have been able to understand the basic structure of the language and have developed their own programs, always in an interactive relationship, involving in parallel human motion and bodily behavior. Although without previous programming experience, none of the students has failed, while they have shown a particular interest and high success rate.

The knowledge gained by students in textual programming has been tested in the context of a three-hour examination. Within this framework, students have been asked to develop their own programs leading to the computer generation of final animated products that comprise motion behavior activated by the movement of mouse in an interactive way. The nature of the exercise was such that students had to arrive at a specific result with specific functional characteristics. This has allowed their assessment based on particular objective criteria related to the ability of the algorithm to achieve the animation goals of the exercise. Although with little or without prior knowledge on textual programming, the students managed to accomplish the given tasks within the time frame of the three-hour exam. The time of fulfillment ranged between 45 min. to 156 min (Figure 2). Teaching of Processing as a textual programming language has evolved over the first five years of the course (Figure 3).

Although the teaching of textual programming using Processing environment has been done successfully through the first five years, it has been done so more as an independent module within the wider objectives of the course. The need to concentrate different modules under a common parametric platform has triggered us to the exploration of alternative solutions. These had to actively contribute to the successful development of projects by integrating users and their behavior within a common and easily manageable environment for design investigation. Another requirement was to integrate other plug-ins in order to achieve a complete design result. Finally, the
new solution was also affected by the necessity to combine in an easy and flexible manner both visual and textual programming, providing an understanding of their mutual use. This was also demanded by students, who were looking for an environment in which, both, learning a programming language and developing new plug-ins could help them enhancing their design investigation. All the above resulted in the introduction of the Python programming language in the Grasshopper parametric environment (plug-in for Rhino 3D) through ICARUS platform since the academic year 2017-2018.

In addition to teaching pure Python, the textual programming module of the course incorporates other interesting features included within the wider context of ICARUS platform, which are described in detail in the following section of the paper. This platform aspires to bring forth the full benefits of the use of the open source computer programming environment by allowing the development, storage and distribution of new design and fabrication tools that can be made widely available for use by the architectural teaching and research community. An advantage that expects to overcome current obstacles of fragmented information in regard to the deepening on computer programming and developing new plug-ins in a single parametric environment.

**ICARUS PLATFORM DESCRIPTION**

The platform is created based on the Python programming language, which is run in the parametric modelling environment of Grasshopper, a plug-in for Rhino 3D modeling software. In its initial phase, the platform was designed to consist of four separate, but interrelated parts: a. teaching manual that includes modules of code, b. samples of combinatorial coding based on bottom-up composition of modules (.gh files), c. guidelines for Grasshopper plug-in development including samples of tools (.ghpy files) created specifically for the teaching and research needs of our institute, and d. an open source online repository for development and operation in the Github platform (Figure 4).

**Modules of code and bottom-up composition**

In this part of the platform, the basic code syntax of programming language Python is recorded and presented so that even students without prior programming skills are able to understand, edit and modify the suggested modules of code, as well as to write their own tools. All this is done via the GhPython plug-in [3] which is embedded in Grasshopper’s environment.

Within this framework and based on ICARUS teaching outline, students are taught a number of basic functions of the language such as code for ‘Mathematical’ and ‘Logical Operations’. Then, modules of code are provided, which describe the scripting of commands and primitive shapes. Figure 5 shows the example of Curve modules of code provided by the platform based on the organizational principles of the Grasshopper environment.

Subsequently, the way the users of the platform can move on from modules of code understanding to the composition of modules based on bottom-up logic, is examined. Our purpose is to teach students how to develop algorithms consisting of successive geometrically defined shapes. Towards this direc-
Plug-ins and tools development
The Grasshopper plug-in development in .ghpy type of files, is an indispensable part of ICARUS platform. The suggested procedure simplifies the way in which plug-ins are developed as compared to previous methods, where knowledge in regard to programming with VB or C sharp was necessary. In particular, by using the capabilities provided through Rhino 6, users are able to convert their Python’s modules of code compositions into ‘packed’ single components or an aggregation of components and hence, they can develop their own plug-ins [4]. Note that this method is only available in the new version of Rhino 6 [5].
Repository
Finally, collective information from the previous parts are included in Github, a popular application in code management based on the Git protocol, an open version control system (vcs). Students, in parallel with textual programming learning, acquire knowledge in regard to the use of Github, which can make textual program and plug-in development easier throughout the phases of writing, distribution, and improvement. In this way an open source online repository can be developed in the benefit of the students and the architects’ community involved in the ICARUS project. Information regarding ICARUS Codebase can be found in [6] (Figure 6).

CASE STUDIES
As already mentioned, a first attempt to test the feasibility of the platform was conducted, within the framework of a postgraduate course, during the academic year 2017-18 and specifically between September-December of 2017. The participants were seven students with Grasshopper experience ranging from enough to nothing and with textual programming skills ranging from little to nothing. The classes were broken down into nine segments: six for teaching basic functions and modules of code and three, for teaching bottom-up composition of codes. Then, students had to work on their capstone project that involved the development of interactive design platforms using data acquisition devices, sensors and actuators embedded in Grasshopper through different plug-ins, accompanied by skills acquired during Python classes. In parallel, an exam that involved simple algorithmic development took place in the classroom.

ICARUS teaching outline
The textual programming module is run for nine weeks and the content is distributed according to the structure of the tutorial, which is available on the Github platform of ICARUS. Starting from the basics of Python programming language, students learn the basic syntax of the language, its important concepts and its functionality. Then, students are introduced to the Python component and its relation to the available Grasshopper components. Our attempt is to continuously relate the textual programming with the visual programming approach, aiming to provide an easy understanding through observation, comparison and testing. At this point, students study systematically the given modules of code. Modules are structured according to the currently available structure and syntax of Grasshopper components. This part of the learning process, which takes up most of the time, aims to connect the students’ understanding of the basic components that appear in Grasshopper with textual programming syntax, with the prospect of developing other modules of code by the students, following the same principles. Finally, through practical examples, students are introduced to inductive thinking through functioning of modules of code, aiming at their composition based on bottom-up logic in order to develop larger algorithms that can be applied to design examples.

Six classes for teaching basic functions and modules of code. The first class starts by introducing students into the fundamentals of textual versus visual programming languages. Particularly, a discussion is taken place in relation to the Grasshopper platform,
its associative logic, the function of components and their syntax in larger clusters, creating overall visual algorithms. Then, discussion in regard to the available textual programming platforms in Grasshopper is conducted (C sharp, VB and Python) and their association with other standalone programming languages (Processing, etc.). In the second class, an introduction into the Python programming language is given, correlating the available Grasshopper components with the modules of code and their syntax in Python that can produce respective results. This is again a general discussion that aims to teach to the students the fundamental concepts behind textual programming languages.

During the third class, an introduction into the working environment and variables of the GhPython programming language is given, aiming to recap information delivered in previous classes in regard to the use of programming languages in the Grasshopper platform. In addition, the basic syntax of the Python programming language and samples of code are handed out in order to introduce all students into textual programming logic, even those who do not have any programming skills. This enables them to understand, edit, and modify the offered code but also to write their own plug-in tools. Thus, in this class, students familiarize themselves with information regard to Python 3, a high-level programming language in an open source status. Subsequently, fundamentals of the language are covered including ‘Variables’, ‘Mathematical operations’ ‘Strings’, ‘Lists’ and ‘Dictionaries’. Finally, more analytical explanation in regard to ‘Logical operators’, ‘Decision structures’, ‘Looping’, ‘Functions’, and ‘Classes and objects’ are provided.

In the fourth class, modules of code are taught, emphasizing the direct development of geometries in Grasshopper environment. Thus, students are initially taught how to use the modules of code for ‘Curve’ that include ‘Primitives’, ‘Analysis’, ‘Division’, etc. At the same time, they learn how to develop their own shapes and control respective parameters. During the fifth class, students are introduced to the ‘Maths’, ‘Sets’ and ‘Vectors’, enabling them to combine these modules of code with previous knowledge obtained.

Finally, the sixth class introduces the ‘Surface’ modules of code that include ‘Primitives’, ‘Analysis’, ‘Freeform’, etc. In addition, modules of code related to the ‘Transform’ and ‘Intersect’ are also explained. By the end of these classes, students are required to write or combine different modules of code that are associatively related in order to develop their own algorithms.

Three classes for teaching bottom-up composition of codes. Having acquired an initial knowledge in regard to the way modules of code can be basically combined to develop larger algorithms, in this part of textual programming teaching, an attempt to provide a bottom-up logic is provided. The goal is to combine and develop parametrically controlled algorithms that involve a series of modules using existing knowledge. This is achieved by allowing the students to study, manipulate and edit given larger algorithms in order to understand their organizational structure. Then, they are asked to write algorithms that combine, in a bottom-up manner, the modules of code introduced in previous classes.

Teaching Case Study A
Python textual programming exams. The first teaching case study refers to a three-hour examination within the postgraduate course, where students are asked to develop an algorithm using Python textual programming language in Grasshopper. Due to the short time of the exam, the creative design aspect is not included, and instead a ready-made visual programming algorithm is given. Based on this, students are asked to write the same algorithm using textual
programming principles. Seven different visual algorithms representing simple parametric morphologies are given to each one of the students who participated the course.

The results of case study show that although students did not have enough background, either on visual or textual programming, all of them managed to accomplish the given programming tasks at less than half of the total time. Also, the time difference between the student who finished first and the last one was 71 minutes, which can be considered logical given the different background of the students (Figure 7). Still, comparing the difference between completion time with Processing (111 min) and with Python (71 min), it appears that, by using the second language, the gap between expert and non-expert users is smaller, showing that Python is probably a more user-friendly language to learn for people without programming skills. Further studies will compare the time that takes to develop the same algorithm in the visual and textual programming environments to see if in the second case the time needed is reasonable as the students have little or no programming experience. Figure 8 shows two representative examples of the assignment. Overall, the results show that previous experience in parametric design or computer programming is not a prerequisite condition to learn textual programming and be able to parametrically define a 3D model using Python.

Teaching Case Study B
ICARUS platform correlated with the capstone project. In the second case study, students had the opportunity to use their knowledge in textual programming in order to develop their capstone project. This was achieved by incorporating custom Python components in Grasshopper in order to develop an interactive workflow for design optimisation, an approach implemented based on a given scenario of use. Thus, Python textual programming components have been integrated into parametric definition, making textual programming an essential part of the process.

Research Case Study C
Finger joint example. The example described in the third case study attempts to simulate another real-world scenario of use by referring to a parametric tool for calculating and designing the slots needed to join two components together with the ultimate goal of their digital manufacturing and assembly through digital fabrication principles. Originally designed as a simple parametric algorithm that was given to the students to build their prototypes, the development of a textual counterpart algorithm and the further creation of a plug-in tool for digital fabrication has followed the steps outlined in the ICARUS platform. The given code is used to define a notch in one of two complementary pieces of material for the purpose of joining them together. As input variables it receives a curve, a set of variables that have to do with the rebuild curve, a variable that regulates the num-
ber of slots, a Boolean variable that regulates the direction of the slots, and a variable defining the thickness of the material and the size of the notches respectively. As a result, the notch is given as a continuous curve (Figure 9).

This case study shows the ability of the ICARUS platform to lead students and anyone involved in programming to develop plug-ins for design and manufacturing purposes in an easy and comprehensible way, but also through simple steps and without sophisticated programming skills and knowledge.

CONCLUSION

By developing the ICARUS platform, our aim was to reconsider the way a textual programming language, in this case the well-known programming language of Python that is embedded in Grasshopper, can be taught and progressively learned. Our target is to provide guidelines, both in terms of understanding the structure of individual modules of code and their bottom-up composition, in order to enable students and even professional architects to participate in the process of writing an algorithm with little or without prior programming background and knowledge, as well as to facilitate the development of plug-ins.

Preliminary results obtained through a series of case studies related to teaching and research activities have shown that the platform and its integration within a visual programming environment can contribute towards textual programming understanding and learning. Also, the results undertaken demonstrates its great potential to be introduced as a comprehensive tool for textual programming among students and researchers in architecture field. These results encourage further development of the different parts of the platform for it to provide the background and knowledge, for better understanding of the various matters regarding the programming process. Towards this direction, on the Grasshopper front, further work will include the enrichment of the platform with new modules of code, compositions of codes, the quotation of several compositions, and an extensive sample of plug-ins.

Ultimately, the goal of the ICARUS project is to
evolve into a universal codebase that expands in order to cover the greatest possible range of design tools of the Architectural community. In pursuit of this, and given the near conclusion of the fundamental development of the Grasshopper part of ICARUS, we will seek to expand ICARUS’s codebase to other parametric design environments and programming languages. During this process we hope to see support from the open source culture that ICARUS embraces and cultivates. Finally, several other case studies will be conducted and discussed in greater depth aiming to the better understanding of how to further develop the project.

ACKNOWLEDGEMENTS
The authors would like to thank the students of the postgraduate course ARH-522 - Advanced Computer-Aided Design Topics taught in the Department of Architecture at the University of Cyprus during the Fall semesters of academic years 2016-17 and 2017-18. Their engagement in learning and testing of Processing and ICARUS platform was valuable and productive, allowing useful results to be obtained. Also, we would like to thank George Tryfonos, PhD student, for his involvement in the parametric development of finger joint example conducted for the purpose of the parametric course ARH-421 - Advanced Computer-Aided Design.

REFERENCES
Burry, M 2011, Scripting cultures, Wiley, West Sussex
Coates, P 2010, Programming. Architecture, Routledge, Oxon
Wurzer, G and Pak, B 2012 ‘Lawnmower: Designing a web-based visual programming environment that generates code to help students learn textual programming’, Proceedings of the 30th eCAADe 2012, Prague, pp. 655-663