METU JFA 2007/2
(24:2)

AN INNOVATIVE DESIGN EDUCATION APPROACH: COMPUTATIONAL DESIGN TEACHING FOR ARCHITECTURE

Birgül ÇOLAKOĞLU and Tuğrul YAZAR

Received: 20.09.2007

Keywords: computational design thinking; computational design; algorithmic thinking; computational design education.

This paper describes an innovative design education approach developed to explore new intellectual and theoretical directions of design generation. It utilizes algorithms as a tool for introducing the concept of design computing to graduate students of architecture. The graduate course titled "Designing the Design" is thus developed. The aim of the course is to introduce to the students of architecture the computational design thinking and the new emerging language and method of designing. Examples from the course process are given to illustrate explorations in this new auxiliary teaching method of the design.

INTRODUCTION

Computation deals with solving problems on a computational model using an algorithm. Within this definition design computation deals with solving design problems on a computational model. Any design problem that can be described as computational model, in other words as an abstract model in which the problem is represented with a set of variables and a set of logical relationships between them can be solved by using computational techniques.

In the realm of architecture, computational design has emerged as a sub-discipline of architecture which is multidisciplinary in nature and uses advanced computing capabilities to understand and solve complex problems of the architectural design. It provides methods for an architect/designer in harnessing a more deliberate and conscious thought process in the design.

Computational design approach in education was first introduced by Mitchell, Ligget and Kvan in *The Art of Computer Graphics Programming* (1987). This book created a source of inspiration for a new research area that explores computational design teaching models (Yakeley, 2000; Celani, 2002) in architecture. New design courses that aim to investigating and

exploring the process and theories of computational design have been developed (Nagakura, 1998; Terzidis, 2002; Celani, 2004; Duarte, 2007).

The objective of Nagakura's *Formal Design Knowledge and Programmed Constructs* course was stated as "…to provide students practical and theoretical foundations to explore computational issues relevant to representation of architectural forms and design knowledge (Nagakura, 1998). Students were taught basic concepts of programming language in AutoLisp, the scripting language of Autocad. The aim of Terzidis' *Algorithmic Architecture* course was stated as "…to develop algorithms and computational methods that would encapsulate the process that lead to the generation of meaningful architectural form." (Terzidis, 2002) In this course, for the codification of design intention scripting languages available in 3D packages (i.e. Maya embedded scripting language MEL, 3Dmax Script) were used. Celani's experimental course *CADCreativo* aimed to explore the use of logical operation in design and the use of CAD not only as representational tool but rather as an explorative, customizable design aide for the creative process. The CAD software used in this course was AUTOCAD 2000 with included VBA development environment. Duarte's *CAD II: Programming and Digital Fabrication* course introduces "... the theoretical and practical fundamentals for the exploration of computational aspects of architectural form and knowledge" (Duarte, 2007). The basic concepts of programming are introduced by using Autolisp in Autocad.

The course described in this paper is in line with the above mentioned courses. However, it distinguishes from them by putting emphasis on the computational thinking rather than just mere algorithm developments that lead to the generation of a meaningful architectural form. It points out that computational thinking is not only programming but rather conceptualizing that operates in the multiple layers of abstraction simultaneously.

COMPUTATIONAL DESIGN FOR ARCHITECTURE

The concept of computational design thinking is related to algorithmic thinking that architects use in their design process rather than the tools they use. It involves an algorithmic logic that is deterministic, rational, consistent and systematized. Most algorithms are symbolic and are used to automate manual methods by means of formal languages. Computational design thinking is described as being algorithmic. Computational design systems and techniques are derived on the base of computational design thinking.

Practices such as Gehry Associates and Foster Partners are establishing Research and Development teams to look at computational techniques and their possible impact on design capability. Architects that are driven by this are exploring new computational design methodologies/languages that allow them to go beyond the mouse screen interaction, into the logic of formal language (Goulthorpe, 2003; Cache, 2003). They are discovering non-visual or numerically driven methods of computation which assist and enhance their designs. They use design systems to calculate a façade, apply transformation equation to a surface or to measure building performances under various conditions. To do this, architects have to have explicit knowledge of how to use the design system. This is a challenging new task

in educating an architect, as the students of architecture need to be taught the basics of programming and logical approach to a problem solving.

Programming imposes esoteric computational laws on architecturally trained designer therefore expecting an architect to be a programmer would be unrealistic. However, architects that are driven by ideas instead of technology can develop casual programmer skills (Ousterhoud, 1998) that allow them to go beyond object manipulation into a creative use of computer.

The alternative to a programming language can be the scripting language that is relaying on the components of higher level programming language and gluing them together (Ousterhoud, 1998). Scripting languages are used for rapid development of a program, connecting and creating relationship of different parts within the program. The syntax and semantics of a scripting language are simpler to understand and develop for non programmer therefore it is used by designers to customize software to achieve the benefit of new tools and material with computer technology.

DESIGNING THE DESIGN: COMPUTING FOR ARCHITECTURE COURSE

Recent theories of form in architecture focused on computational explorations and expressions of how we teach design. Barts Lootsma in "Hybrid Space" (Zelner, 1999) speaks of the new direction in architecture: "instead of trying to validate conventional architectural thinking in a different realm our strategy today should be to infiltrate architecture with other media and disciplines to produce crossbreed."

Following these concerns the graduate course "Designing the Design" is developed reconsidering relationships between computational design thinking, design computing and digital design. It aims to introduce students with computational design thinking and tools to explore the new methods of designing. The creative use of computer scripting is being used to mechanize the abstraction layers and their relation in computational design thinking.

The course is developed in two modules: the first describes the concept of computational design. It includes an introduction to computational design thinking and formal languages. It discusses new computational methods of formal exploration and expression of design. The second module introduces the students with algorithms and scripting through exercises described below.

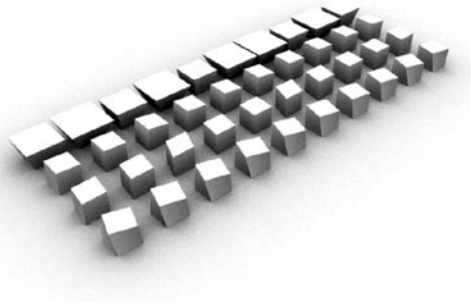
USING ALGORITHMS AND SCRIPTING AS DESIGN TOOL IN ARCHITECTURAL EDUCATION

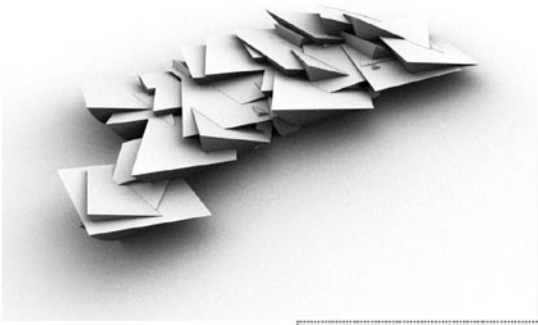
This module involves the codification of design intention through algorithmic scripts built on top of existing CAD systems. It includes class and home exercises and a final project. The exercises start with abstract forms and their computation then focus on various architectural problems. 3dMax scripting environment is used as a design tool for teaching algorithmic logic and scripting.

First, the students are introduced with principles of programming logic through class exercises.

- Syntax of computer programming,
- Program flow, (loops and conditional statements),

```
for i = 1 to 40 do (
	a = box heightsegs:20
		if i>0 and i<11 then (
			a.pos = [(i*50),50,0]
			addModifier a (twist angle:(random 10 20))
			)
		if i>10 and i<21 then (
			a.pos = [(i-10)*50,100,0]
			addModifier a (skew amount:(random 1 10))
			)
		if i>20 and i<31 then (
			a.pos = [(i-20)*50,150,0]
			addModifier a (bend angle:(random 0 10))
			)
		if i>30 and i<41 then (
			a.pos = [(i-30)*50,200,0]
			addModifier a (taper amount:(random 0.1 1.0))
			)
)
```





```
for i = 1 to 40 do (
	a = box height:40 heightsegs:10
	if i<11 and i>0 then (
		a.pos=[(i*50),50,0]
		addModifier a (skew amount:(random 0 20))
	)
	if i<21 and i>11 then (
		a.pos= [(i-10)*50,100,0]
		addModifier a (taper amount:(random 0 5))
	)
	if i<31 and i>21 then (
		a.pos=[(i-20)*50,150,0]
		addModifier a (twist angle:(random 0 10))
	)
	if i<41 and i>31 then (
		a.pos=[(i-30)*50,200,0]
		addModifier a (bend angle:(random 0 30))
))
```

Figure 1. An introductory exercise. The script code and output.

Figure 2. An introductory form-finding exercise. The script code and two randomized outputs.

- Variables, operators and transformations,
- Custom functions and built-in features of the script language.

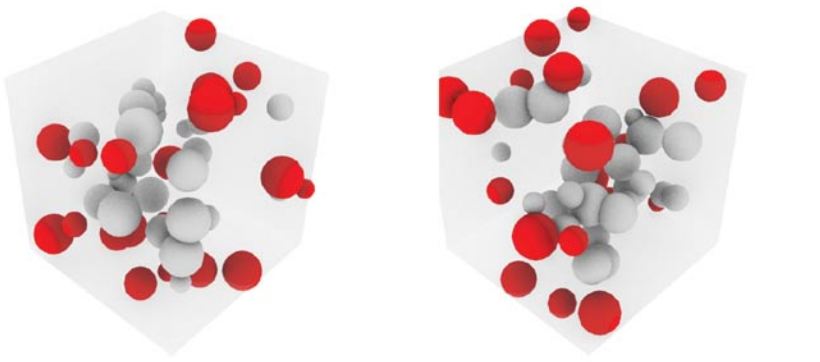
Figure 1 and **Figure 2** illustrate such exercises.

Figure 3 illustrates an exercise that particularly focuses on geometric problem-solving. It asks students to define a function that paints randomly distributed spheres in relation to the boundaries of a parametric box. Spheres that run over the box are painted red, and the spheres that are inside the box are painted white.

Once the students gain experience in scripting through exercises in the next phase, abstract objects are replaced with architectural objects. Here, they are introduced with the parametric structure in CAD tools. Three such exercises are explained below:

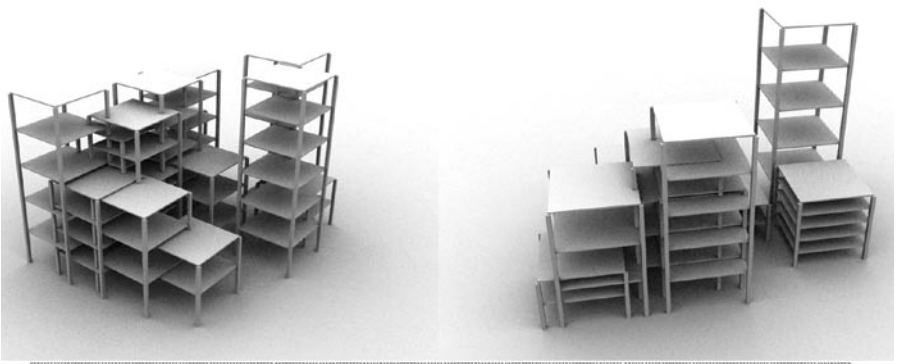
The first exercise aims to teach the logic of relational geometry. In this exercise, students are asked to develop a scripted function that creates frame structures consisting of columns that are related with varying floor dimensions.

As students develop the function, they realize a relationship between certain parameters (minimum thickness of columns, maximum dimensions



```
b = box width:100 height:100 length:100 pos:[50,50,0] visibility:false
for i = 1 to 50 do (
        a = sphere radius:(random 5 10)
        a.pos = [(random 0 100),(random 0 100),(random 0 100)]
        a.wirecolor = color 255 255 255
        if (a.pos.x)+(a.radius) > 100 or (a.pos.x)-(a.radius) < 0 then (a.wirecolor = color 255 0 0)
        if (a.pos.y)+(a.radius) > 100 or (a.pos.y)-(a.radius) < 0 then (a.wirecolor = color 255 0 0)
        if (a.pos.z)+(a.radius) > 100 or (a.pos.z)-(a.radius) < 0 then (a.wirecolor = color 255 0 0)
)
```

Figure 3. An introductory problem-solving exercise. The script code and two randomized outputs.



```
fn bldg ka:3 ky:300 pz:[0,0,0] = (
        for i = 1 to ka do (
                a = box height:10 width:500 length:500 pos:pz
                move a [0,0,(i*ky)]
                box height:ky width:25 length:25 pos:(a.pos + [250,250,-ky])
                box height:ky width:25 length:25 pos:(a.pos + [-250,250,-ky])
                box height:ky width:25 length:25 pos:(a.pos + [-250,-250,-ky])
                box height:ky width:25 length:25 pos:(a.pos + [250,-250,-ky])
        )
)
for i = 1 to 15 do bldg ka:(random 2 6) ky:(random 150 300) pz:(random [0,0,0] [2000,2000,0])
```

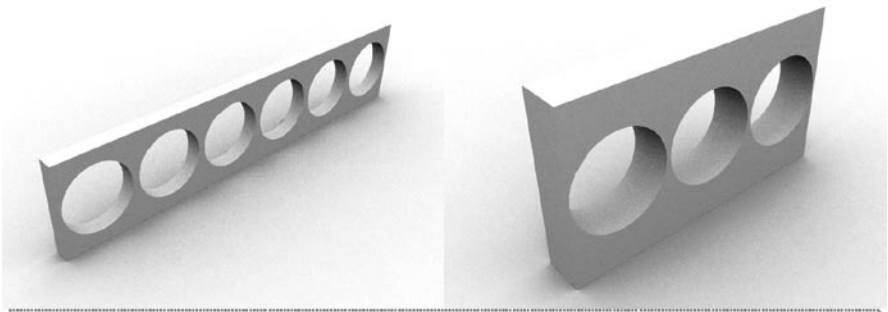
Figure 4. Scripted function that creates frame structures and it's executions with randomized inputs.

of floors, etc.) and the materialization process of their architectural counterparts.

After defining the function, students keep playing with their script entering random or user-defined parameters. **Figure 4** illustrates students' experiments.

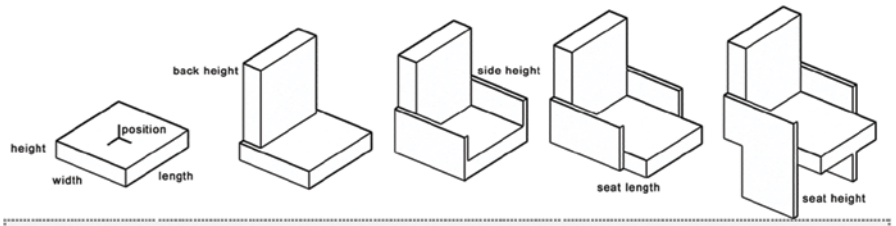
The second exercise introduces the parametric design. Students are asked to develop a scripted function that opens circular holes (windows) on any given box object (wall). The only parameter of the function is the number of holes to be opened. As the box dimensions vary, students deal with geometric and arithmetic problems. Some outputs of this function are illustrated in **Figure 5**.

The third exercise introduces object-oriented logic. It consists of two parts. In the first part, students design a parametric theater chair illustrated in **Figure 6**. In the second, they design a parametric theater that uses the chair object designed in the first part.



```
fn window wall nr = (
        window = mesh(); wall2 = copy wall; convertToMesh wall2
        dia = (wall.length/ nr); if dia > wall.height then dia = wall.height-20
        for i = 0 to nr-1 do (
                cylinder height:(wall.width*2) radius:((dia/ 2)-10) sides:32 isSelected:on
                $.pos = [(wall.pos.x),(wall.pos.y),(wall.height/ 2)]
                move $ [-(wall.width),(i*dia)-(wall.length/2)+(dia/2),0]
                rotate $ (eulerangles 0 90 0)
                window = window + $; delete $
                )
        wall2 = wall2 - window
        delete wall; delete window
        update wall2
)
window (box length:1400 width:50 height:300) 6
```

Figure 5. Script code and some results of the function.




```
fn theaterSeating koord:[0,0,0] wdth:50 dpth:45 seat:10 hight:30 hightB:45 hightA:15 = (
        box height:seat width:wdth length:dpth pos:koord isSelected:true
        convertToPoly $
        $.slice [1,0,0] [22,0,0]; $.slice [1,0,0] [-22,0,0]; $.slice [0,1,0] [0,11,0]
        polyop.setFaceSelection $ #{22}; $.extrudeFaces hightB
        polyop.setFaceSelection $ #{2,8,16,20}; $.extrudeFaces hightA
        polyop.setFaceSelection $ #{13}; $.extrudeFaces 15
        polyop.setFaceSelection $ #{1,19}; $.extrudeFaces hight
        move $ [0,(-dpth/2.0),hight]; $.pivot = koord
        addModifier $ (Optimize())
)
```

Figure 6. Generation process of the parametric theater chair and it's scripted function.

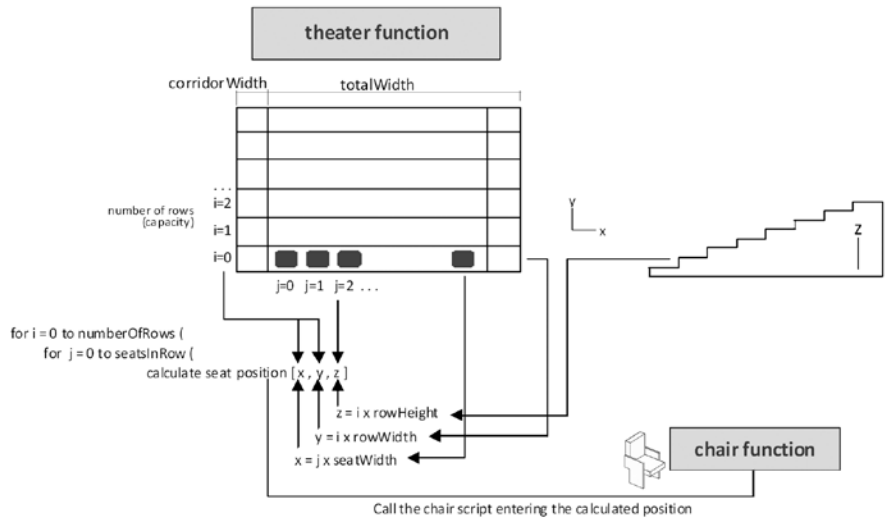


Figure 7. Explanation of the loop, defining theater seating positions.

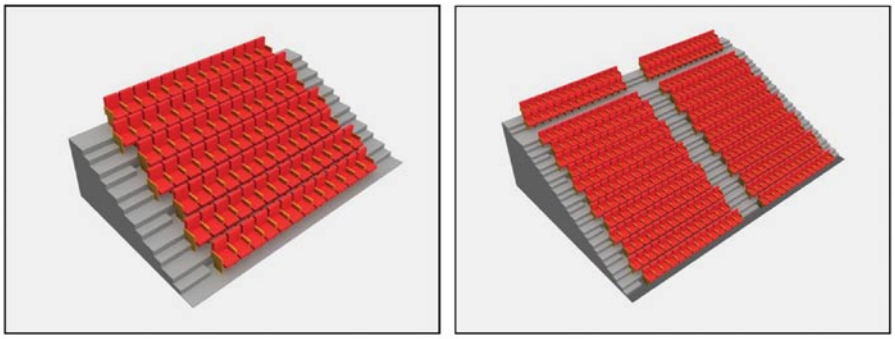


Figure 8. Some different theaters created by the function.

The function that describes theater chair can be called from another script by entering general parameters, leaving all other parameters to be calculated automatically through relational geometry. This exercise introduces students with open source coding and the importance of object databases for CAD tools. In the second part of this exercise, students learn the benefits of this logic by using their custom object class in another script.

The second part of the exercise is designing the theater seating. The parameter inputs of the function given above are the total capacity, and at least one dimension (depth or width) of the theater. The students should define various constraints like maximum depth of a theater, maximum number of chairs between two corridors etc. The script creates a basic model of theater seating and calls the calculated number of chairs with correct dimensions to appropriate positions. In order to calculate the number and the position of seats, students have to create a calculation algorithm. As shown in **Figure 7**, two program loops are created (j'th seat in i'th row) to set the positions of the chairs one by one. Various parameters like rowHeight, rowWidth or seatWidth are used to define chair positions multiplying with i or j. Two different theater seating compositions are illustrated in **Figure 8**.

After students gain the experience with predefined exercises, they are assigned for a final design project of their own. They are asked to define a computational design problem and a solution in which they will use the knowledge and skills gained in the exercises. The final projects include following phases:

- Statement of a need, providing a design problem with an architectural counterpart,
- Formal analysis phase, including typological categorization,
- A design brief, developed by selecting a particular category,
- Parameterization, resolving the design brief and exploring variations,
- Utilization, coding the hypothetical design using the parameters,
- Testing the code, evaluating it's expected benefit, performance and usability,
- Returning to the parameterization or utilization phase if necessary.

Four groups of students (each consisting of two students) developed their own final project. An example of final group work is explained below. The design problem defined for the final project by the project Group 1 (students are Uğur Işık and Eda Erkan Altunbaş) is a parametric canopy

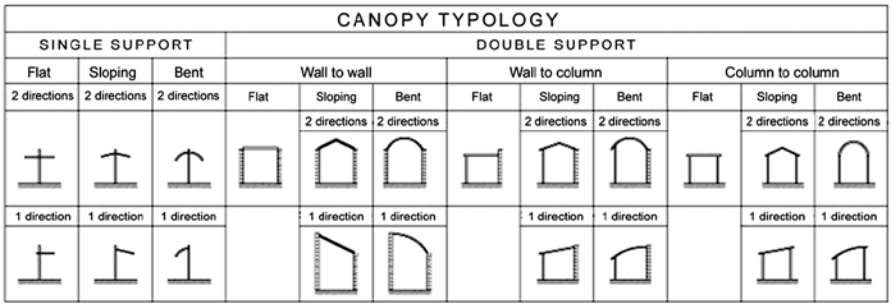


Figure 9. Typological analysis.

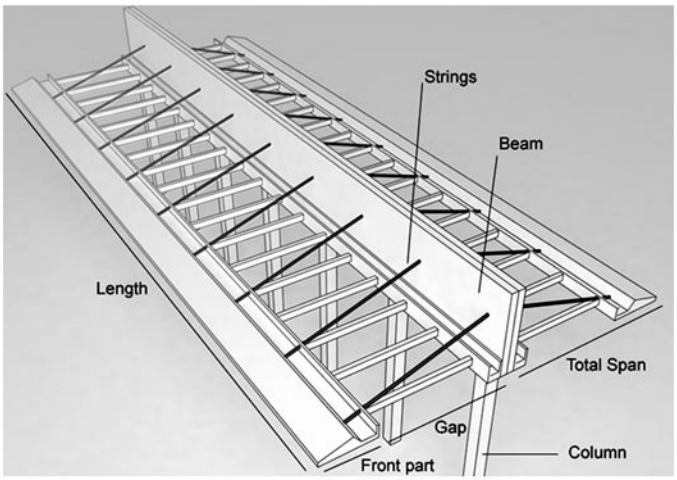


Figure 10. The design outline for the final project.

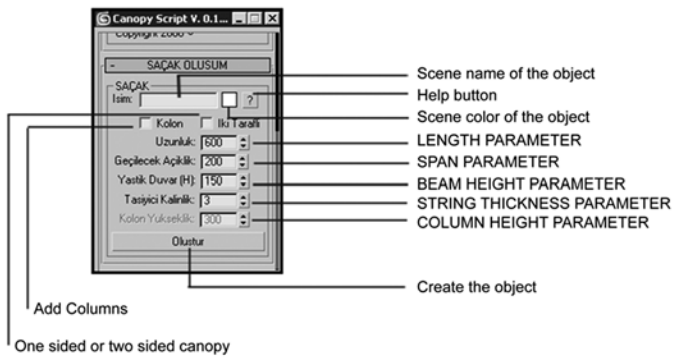


Figure 11. Graphical user interface of the student project.

design. In the formal analysis phase, the group first did research on canopy typology (**Figure 9**), and then selected a single support canopy for parameterization (**Figure 10**).

In the parameterization phase, students defined six main parameters of a canopy design shown in **Figure 10**. These parameters are span of the canopy, total arcade length, beam height, height and thickness of strings, height of columns and the Boolean parameter of whether the canopy is two sided or one sided. After defining the constraints between the parameters the student group coded the script that creates parametric canopy designs.

The script interacts with the user by a graphical interface while it creates desired canopy structures by only a few mouse clicks. In **Figure 11**, the graphical user interface of the canopy script is illustrated. In the test phase of the project, students discussed with other groups and experimented

with each other's project, evaluating the usability of the script and the reasonability of its outputs.

CONCLUSION

The last generation digital design tools provide both a programmatic and visual way of building geometry that can be used to interactively control the computational design model. They are concerned with complex geometry; they apply complex user-defined computations to a design and control it. A designer in order to use these tools in an explorative design context needs to be 'geometrically aware' and 'computationally enabled' (Aish, 2005). The traditional architectural education does not put any effort to keep up with the rapid change of the digital technology and the computational theory behind it. While computational design is progressing with the full speed, a gap between the architectural education and this new realm of design still exist.

Designing the Design" is an exploratory course that aims to fill the gap in the architectural education between the prospective architect and his/her design tool by introducing the logic and the mechanism on which these tools operate. It introduces students with computational thinking and the mechanization of it in the realm of design.

The course is structured following developmental pedagogy. First students are introduced with computational thinking that is abstract, recursive, onward, procedural and logical. And then they are navigated to understand its automation, the mechanization of layers of abstraction, and their relationship using formal language. Simple exercises, complexity of which were raised step by step, are constructed to teach the automation of computational design thinking.

Computational design tools do not provide a designerly way of doing as does the intelligence acquired through design experience. However, they spread computational thinking which takes an approach to solving design problems and designing systems based on concepts fundamental to computer science. They are auxiliary design tools in which the design, from concept till production, can be controlled.

The course emphasizes that computational thinking will be a fundamental skill to be used by designers in the near future.

REFERENCES

AISH, R. (2005) From Intuition to Precision, *Education of Computer-aided Architectural Design in Europe*, (*eCAADe*) 23.

CACHE, B. (2003) Towards a Fully Associative Architecture, *Architecture in Digital Age -Design and Manufacturing*, Kolarevic, B., ed., Routledge: Taylor and Francis.

CELANI, G. (2002) Beyond Analysis and Representation in CAD: a New Computational Approach to Design Education, unpublished Ph.D. thesis submitted to Department of Architecture, MIT.

CELANI, G. (2002) CAD- The Creative Side - An Educational Experiment that Aims at Changing Students' Attitude in the Use of Computer-Aided Design, *SIGraDi 2002*.

CELANI, G. (2004) The Symmetry Exercise: Using an Old Tool in a New Way, *SIGraDi 2004*.

DUARTE, J. (2007) Inserting New Technologies in Undergraduate Architectural Curricula, *Predicting the Future 25th eCAADe Conference Proceedings*.

GOULTHORPE, M. (2003) "Scott Points: Exploring Principles of Digital Creativity," in *Architecture in Digital Age-Design and Manufacturing*, Kolarevic, B., ed., Routledge: Taylor and Francis.

MITCHELL, W.J., LIGGET, R.S and KVAN, T. (1987) *The Art of Computer Graphics Programming*, Van Nostrand Reinhold, NY.

NAGAKURA, T. (1998) *Formal Design Knowledge and Programmed Construct*, course thought at MIT, http://cat2.mit.edu/arc/4.207/.

OUSTERHOUD, J. K. (1998) "Scripting: Higher Level Programming for the 21st Century", IEEE Computer.

TERZIDIS, K. (2002) *Algorithmic Architecture*, course thought at GDS Harvard, http://www.gsd.harvard.edu/cgi-bin/courses/details.cgi?section_id=6847&term=f2004.

YAKELEY, M. (2000) "Digitally Mediated Design: Using Computer Programming to Develop a Personal Design Process," unpublished Ph.D. thesis submitted to Department of Architecture, MIT.

ZELNER, P. (1999) *Hybrid Space: Generative Form and Digital Architecture*, Rizolli Inti Publications.

Alındı: 20.09.2007

Anahtar Sözcükler: bilgi-işlemsel tasarım düşüncesi; bilgi-işlemsel tasarım; algoritmik düşünce; bilgi-işlemsel tasarım eğitimi.

TASARIM EĞİTİMİNE YENİ BİR YAKLAŞIM: MİMARLIKTA HESAPLAMALI TASARIM ÖĞRETİMİ

Bu makale "Tasarımı Tasarlamak" adlı deneysel tasarım atölyesi eğitimini anlatmaktadır. Atölyede algoritma, mimarlık öğrencilerini bilgi-işlemsel tasarım mantığı ile tanıştırmak için araç olarak kullanılmıştır. Atölyenin amacı mimarlık öğrencilerini bilgi-işlemsel tasarım mantığı ve yeni tasarım dili ve yapma yöntemi ile tanıştırmaktır. Tasarım sürecini destekleyici olan bu yöntemin tasarım eğitimine entegre edilmesi ile ilgili yapılan deneysel çalışmalar atölye sürecinden örnekler verilerek anlatılmıştır.