TEACHING CAD PROGRAMMING TO ARCHITECTURE STUDENTS

ENSINO DE PROGRAMAÇÃO CAD PARA ESTUDANTES DE ARQUITEURA

Gabriela CELANI
Architect, Ph.D. professor of the State
University of Campinas, São Paulo, Brazil
celani@fec.unicamp.br

ABSTRACT

The objective of this paper is to discuss the relevance of including the discipline of computer programming in the architectural curriculum. It starts by explaining how computer programming has been applied in other educational contexts with pedagogical success, describing Seymour Papert's principles. After that, the historical development of CAD is summarized and three historical examples of educational applications of computer programming in architecture are presented, followed by a contemporary case of particular relevance. Next, a methodology for teaching programming for architects that aims at improving the quality of designs by making their concepts more explicit is proposed. This methodology is based on the author´s experience teaching computer programming for architecture students at undergraduate and graduate levels at the State University of Campinas, Brazil. The paper ends with a discussion about the role of programming nowadays, when most CAD software are user-friendly. As a conclusion, it is suggested that the introduction of programming in the CAD curriculum within a proper conceptual framework may transform the concept of architectural education.

Key-words: Computer programming; computer-aided design; architectural education.

RESUMO

O objetivo deste trabalho é discutir a relevância da inclusão de uma disciplina de programação de computadores no currículo de Graduação em Arquitetura e urbanismo. Ele começa explicando como a programação tem sido aplicada em outros contextos educacionais com grande sucesso pedagógico, e descrevendo os princípios de Papert. Em seguida, é apresentado um resumo da evolução do CAD e três exemplos históricos de aplicações da programação no ensino de arquitetura são apresentados, seguidos por um exemplo contemporâneo de grande relevância. Finalmente, é proposta uma metodologia para o ensino de programação para arquitetos, com o objetivo de melhorar a qualidade dos projetos, tornando os conceitos arquitetônicos mais explícitos. Essa metodologia é baseada na experiência da autora de ensino de programação para alunos do curso de graduação em arquitetura na Universidade Estadual de Campinas. O trabalho termina com uma discussão sobre o papel da programação nos dias de hoje, quando a maioria dos programas de CAD são amigáveis. Como conclusão, sugere-se que a introdução da programação no currículo de CAD, dentro de um arcabouço teórico apropriado, pode vir a transformar o conceito de ensino da arquitetura.

Palavras-chave: Computer programming; computer-aided design; architectural education.

1. INTRODUCTION

The objective of this paper is to discuss the relevance of including the discipline of computer programming in architectural education. Some people may argue that programming is not necessary and was only defensible when CAD programs needed it to improve performance.

During ECAADE 18 and 19, held respectively at Weimar in 2000 and Helsinki in 2001, there was a series of discussions about digital design curriculum in architecture schools. Although most of the attendees' concerns were restricted to the representational and analytical roles of CAD, many educators defended the introduction of algorithmic design in the architectural curriculum.

The course sequence proposed by Mark, Martens and Oxman (2001), for example, included a topic called "Computables of Design", which explored "the quantitative basis and invisible geometrical order of shapes found in nature and architecture as explored through writing computer programs." Seebohm's (2001) position went even further, suggesting tool building courses, which, he acknowledged, were "perhaps the most difficult technically, because they [involved] computer programming." However, according to him, these courses could be "potentially very rewarding because they [exploited] one of the greatest underused strengths of using computers in design". Latin America representatives (Montagu et. al., 2001), on the other hand, were less ambitious, proposing the introduction of programming with the sole purpose of providing "skills to post projects on websites and understand and configure CAD software."

In the present paper, I argue that programming can improve logic reasoning and conceptual thinking in design. My conclusions are drawn on the historical development of CAD software, on pedagogical experiences with children and architecture students, and finally on some recent applications of programming in architectural design.

2. PROGRAMMING AS PEDAGOGY

In the late 60's and 70's, Papert (1980) developed his first experiments on computer-based learning at the MIT Media Lab. These included teaching children how to program a computer in a specially developed language, Logo. According to Papert, the best way to learn about something was by teaching others and by designing things. In order to teach a computer, one necessarily needs to have a deep understanding of what is being taught, all the information needs to be intelligently organized, problems must be decomposed in simpler sub-problems, and the program as a whole needs to be planed and designed.

In the 80's Harel and Papert (1991) established the Epistemology and Learning Research Group, and developed what became known as Constructionism. They proposed the use of computers as a learning tool and and a "convivial" tool, rather than a teaching tool. According to them, an active use of computers would allow more enrichment than a passive use. In order to be active, a computer user needs to build his or her own software tools. Their pedagogically successful experiments, led to what became known as Papert´s principle: some of the most crucial steps in mental growth are based not simply on acquiring new skills, but on acquiring new administrative ways to use what one already knows (Minski, 1988).

In the 90's, one of Papert's followers, Mitchel Resnick (1994:112), proposed the development of software as a means to "stimulate the creative processes of the designer's mind". He suggested new methods for using computers in the learning process, with the use of object-oriented programming to make programs easier to create, maintain, extend and understand. He also proposed the use of decentralized systems, parallel computation and parallel programming languages to improve speed and performance.

Following Papert's example, in architectural education, our role, as educators, should be to propose the use of computers as a learning tool. The same type of distinction that Papert and his followers have made between the passive and the active use of computers could be applied to CAD. In the following session we will

see how the use of CAD software became progressively more passive along its 50 years of development.

3. SUMMARY OF CAD DEVELOPMENT

Along the past 50 years, computers and CAD software have evolved in parallel, the former greatly influencing the later and setting up its limits. A brief description of hardware and software evolution is given below, with the aim of reviewing the relevance of programming in architectural education and practice along these five decades of development.

3.1. Hardware

Computer's input and output hardware have greatly influenced the development of CAD, and consequently the need of knowing programming to operate CAD systems. The very first CAD applications, for example, used very little graphical interaction and performed mostly mathematical calculations or data management. User's interface was not very well developed then, so programming skills were a requirement for CAD users. The focus then was on the applications of mathematical methods - such as graph theory, optimization techniques and differential calculus - in the design process, boosted by the Design Methods Movement.

The very first CAD applications used little graphical interaction and performed mostly mathematical calculations or data management. In the 70's, raster displays, based on inexpensive television technology, "contributed more to the growth of the field than did any other technology" (Foley, van Dam, Feiner, Hughes & Phillips, 1977:8). The availability of graphic displays with progressively higher speed, resolution and color range, the introduction of new input devices such as light pens, mice and digitizing tables, and the development of user-friendly interfaces (Figure 1), contributed to make CAD software increasingly more graphic-oriented. In 84, Apple introduced the mouse-and-windows style, to which Microsoft responded with the Windows operational system. The new interaction system has been responsible for the widespread acceptance of computers since then. With the easier interactivity and the new focus on visual representation, computer graphics

substituted the initial original of computer-aided design in architectural education. Since the 80's, photorealistic computer rendering techniques have been taught CAD courses in most architecture schools.

Figure 1: Three moments in the hardware evolution: desktop computer with vector monitor, laptop with raster display, and wirelessly connected tablet/palm top with interactive displays.

More recently, the telecommunications technologies have influenced the development of applications by creating the possibility of remote collaboration and by dynamically feeding CAD systems with on-line information. The first trend has led to the development of remote collaborative CAD tools that allow users in different places to share drawings at the same time as they can see and talk to each other. This technology has produced a new change on the focus of CAD courses, and nowadays many architecture schools are introducing remote collaboration courses in the CAD curriculum.

3.2. Software

Mitchell (1990) has characterized five generations of CAD from the early 60's to the late 80's, starting with Ivan Sutherland's SKETCHPAD. Despite Sutherland's introduction of the concept of ill-specified, "loose" inputs, following developments had a different orientation, and CAD programs were turned into an overly precise tool (Negroponte, 1975).

The 60's and 70's first and second generations of CAD were based on expensive, turnkey systems, operated by technicians (Richens, 1992). At this time, architects were still suspicious about the use of computers in design, their attitude positions

ranging from fear to mystification and resentment that computers would diminish humanity in architecture (Broadbent, 1973).

In the 80's, CAD researchers looked for new methodologies and practices, such as "rule-based and/or frame-based systems that provided the means for codifying the problem-solving know-how of human-experts" (Schmitt, 1987:213). This decade saw the parallel development of three different generations of CAD. The third generation was a natural continuation of the previous two. A fourth generation of simplified CAD was developed to be used in the new 16-bit IBM and Apple Macintosh personal computers, finally making CAD affordable to small firms and independent architects. The fifth generation of CAD was related to the development of a new kind of computers in the 80's: the graphic workstations. For these machines, the developers of both mainframe and PC CAD adapted their products, at the same time as special applications were being developed. Their 3d-modeling capabilities led to a discussion about the role of solids modeling in the design process and in replacing traditional bi-dimensional design representation (Eastman, 1987).

The simplification of CAD systems for personal computers led to a radical change in the CAD culture, which Mitchell identifies as an inflection point in the history of architectural CAD, after which "the wider possibilities were largely ignored" (Mitchell 1990:483). At this point, computer programming was completely unnecessary for users of commercial CAD packages. The new standardized, general-use CAD applications for PC's did not aim to help architects from the early, conceptual stages of design. Such specialized applications remained restricted to academic research, while the great majority of offices kept the use of CAD restricted to drafting and representation.

After Mitchell's fifth generation, it is possible to identify a sixth phase of CAD development since the early 90's, characterized by an increasing specialization of products. AutoDesk's website, for example, displays over one hundred different software products. This generation of CAD also takes advantage of connectivity, featuring services such as automatic upgrade downloads and on-line help.

The drastic change in the primary objective of CAD from a problem-solver to a drafting, representation and communication tool is not only related to technical questions, but also to philosophical ones. One of the reasons could be the fact that each architect has a personal way of designing, and it would be extremely hard to develop a conceptual design system broad enough to fit every architect's methodology. Systems that aimed to completely automate the design process never reached widespread acceptance. That very fact probably explains a lot about the design process.

However, it has been possible to observe the emergence of a new generation of CAD in the past few years, called parametric CAD. Although most of these programs - such as AutoDesk's Revit - market the idea that it is possible develop intelligent, parameterized designs without writing code, Bentley's experimental software Generative Components advocates the opposite, and suggests that architectural concepts should be made explicit through programming.

4. APPLICATIONS OF COMPUTER PROGRAMMING IN ARCHITECTURAL EDUCATION

I will now present three historical examples of the use of computer programming in architectural education that I find particularly interesting, analyzing their differences and similarities. They will be followed by a contemporary example in the next section. The timeline in Figure 2 shows their positions in the past two decades. The examples chosen consist of books that have in common the objective of teaching programming for architects (not necessarily at a beginner's level) through actual examples of codes that generate architectural form. These books differ from regular programming books because they discuss generative design methods, not just programming logic or the syntactic characteristics of a specific programming language, or even the automation of simple drafting tasks, such as AutoDesk's guide to AutoCAD's VBA (Roe, 2001).

I do not intend here to discuss the specific content of these books. It does not matter here if they are teaching genetic programming or shape grammars. I am simply

interested in their pedagogical goals and the fact that they are introducing computer programming in a meaningful way to architecture students.

I also do not claim to be presenting a comprehensive survey of this type of publication, which could be unfair with other authors. Although extremely relevant, books such as Bentley's Evolutionary Design by Computers (1999) and dissertations such as Yakeley's Digitally mediated design – using computer programming to develop a personal design process (2000) do not target architecture students directly.

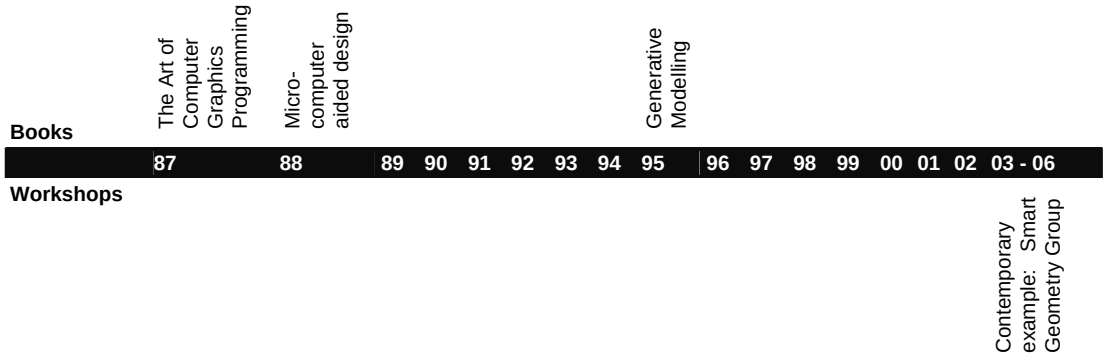


Figure 2: Timeline of the historical and contemporary examples of educational applications of programming in architecture presented below.

4.1. The Art of Computer Graphics Programming

The objective of The Art of Computer Graphics Programming was to teach Pascal graphic programming for beginners, through beautifully illustrated architectural examples. The book presented a series of exercises which are "as concerned with issues of design theory and visual aesthetics as [they] are with computer technology" (Mitchell, Ligget, & Kvan, 1987: vii). As programming concepts were introduced, the authors proposed increasingly complex exercises on vocabularies of shapes, parameterized shapes, repetition, conditionals, hierarchical structures and geometric transformations. Table 1 shows some examples of these exercises, categorized under those concepts. Figure 3 exemplifies the result of one of the proposed programs, which uses randomly generated numbers in a recursive process to generate unique trees.

The purpose of this book can only be understood in a context where commercial CAD packages were still very limited. Nowadays it would not make sense to program simple shapes - such as squares and circles - from scratch, but the concepts introduced by the book are still fundamental to architectural theory.

Concept	**Exercise**
Parameters	"Write a parameterized procedure to generate the basic vocabulary element [for the plan Mies of Van der Rohe's 'Brick Country House']. Use this in a program to replicate the plan. Then use it in a program to produce variations on this theme" (p.198).
Repetition and variation	"It has often been argued that the aesthetic success of a composition is a matter of appropriate balance between 'unity' (which may be established by regular repetition) and 'variety' (which may be introduced by changing parameters from instance to instance). Test this proposition by generating repetitive compositions with different degrees of variation. Provide a critical analysis of your results" (p.250).
Conditions	"Many town plans consist of regular street grids interrupted at various points. Examine some plans of this type. What are the conditions in which the grid is interrupted? Write a set of conditional rules that could be used to produce plans of this type, and discuss their effects" (p.321)
Hierarchy	"All of the elements and subsystems of the Doric order have names. Draw a tree diagram that depicts this hierarchy. Then write a program, structured in the same way, that generates the order" (p.351)
Recursion	"Gothic tracery is often recursive. That is, a large pointed arch is subdivided into smaller pointed arches, each of which is further subdivided in the same way, and so on. Write a recursive procedure to generate such tracery designs" (p.353).
Transformation	"The plan and elevation compositions of the modern architectural masters Le Corbusier, Frank Lloyd Wright and Alvar Aalto rarely display rigid axial symmetry in the classical manner. But on careful inspection, they can usually be found to display less obvious symmetries and carefully broken symmetries. Take a composition that interests you, and the exceptions and distortions that are used to break symmetry. Using the insights that you gain from this analysis, write a concise, expressive program to generate the composition" (p.476).
Biologic analogy	"Examine the leaves along a twig from a plant. Can you characterize the pattern that you see in terms of type and regular repetition? What changes from instance to instance? What kinds of arithmetic and geometric sequences are involved? Write a brief, illustrated analysis" (p.250).

Table 1: Some examples of the exercises proposed in "The Art of Computer Graphics Programming" (Mitchell Ligget, & Kvan, 1987), separated by categories.

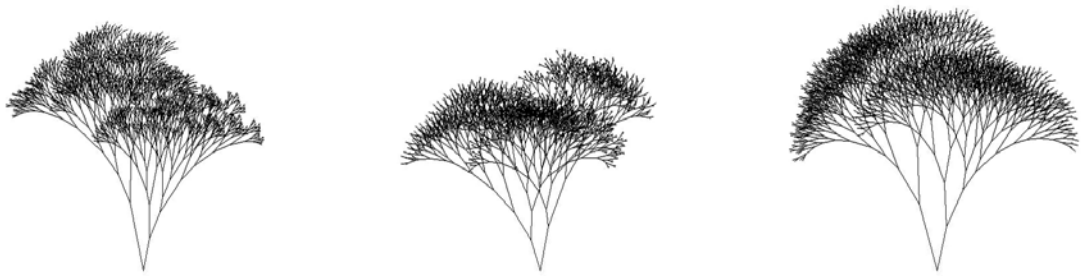


Figure 3: Examples of trees generated by a recursive process with the use of randomly generated numbers in The Art of Computer Graphics Programming.

4.2. Microcomputer Aided Design for Architects and Designers

In the preface to Microcomputer Aided Design for Architects and Designers, Schmitt (1988) asserts that his objective was to "remove the unnecessary mystique surrounding the use of computers for architects and designers" (p.vii). The book was divided in two parts. The first section presented "microcomputers in the traditional design approach", i.e., CAD as a drafting tool. The second part of the book, on the other hand, aimed at revealing "innovative uses of computers and programs as a possible amplifier of human design intelligence" (Schmitt, 1988: viii). Schmitt criticizes the use of computers just for productivity objectives, and proposes a more creative use of the machine: "Although most designers use computers as a productivity and not as a creativity toll, there is evidence that microcomputers can increase our creativity significantly (...) through algorithmic and rule-based programs" (Schmitt, 1988: 85)

Schmitt established a parallel between architectural design and languages, proposing that both could be characterized by a vocabulary, relations, rules, and a grammar. He also related the ambiguities in natural languages to those in architectural languages. Architectural design was thus presented as a ruled-system, consisting of a set of symbols and a set of rules that relates them. The author mentioned different types of generative systems, such as fractals, recursive generation of self-similar elements, shape grammars, parametric shape grammars, and rule-based design.

After that, Schmitt compared computer programming to architectural programming. According to him, both had in common the "analysis of requirements, relations and concepts and their formalization and generation (...) to address requirements and specifications" (Schmitt, 1988:114). Although the objective of the book was not to teach programming for beginners, Schmitt provided many examples of programs in AutoCad's AutoLisp. His codes implemented the generation of parameterized 3D architectural forms, such as vaults and domes, and of recursive parametric shape grammars

(

```
;;; PROGRAM SHAPE1.LSP
;;; A SIMPLE MANUAL SHAPE MANIPULATOR WITH THREE RULES

;;; RULE 1 FUNCTION - PLACES THE INITIAL SQUARE
(defun rule1 ()
   (setq origin (getpoint "\nPlease enter origin: ")
       lowright (getpoint origin "\nEnter lower right corner of square: ")
       a (distance origin lowright) ;;; make sure lowright is on one
       )                            ;;; horizontal line with origin
   (command "INSERT" "square1" origin a "" "")
   (setq count 0
       mark (getpoint "\nPlease enter point on bottom side of square: ")
       c (- (car mark) (car origin))
       d (- a c)
       alpha (angle mark (list (car lowright) (+ c (cadr lowright))))
       b (sqrt (+ (* c c )(* d d)))
       reduction (/ b a)

   (command "CIRCLE" mark (/ b 10))
)  ;;; END DEFUN
```

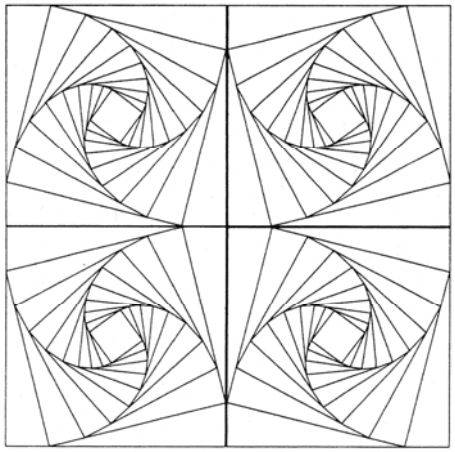


Figure 4).


```
;;; PROGRAM SHAPE1.LSP
;;; A SIMPLE MANUAL SHAPE MANIPULATOR WITH THREE RULES

;;; RULE 1 FUNCTION - PLACES THE INITIAL SQUARE
(defun rule1 ()
   (setq origin (getpoint "\nPlease enter origin: ")
       lowright (getpoint origin "\nEnter lower right corner of square: ")
       a (distance origin lowright) ;;; make sure lowright is on one
       )                            ;;; horizontal line with origin
   (command "INSERT" "square1" origin a "" "")
   (setq count 0
       mark (getpoint "\nPlease enter point on bottom side of square: ")
       c (- (car mark) (car origin))
       d (- a c)
       alpha (angle mark (list (car lowright) (+ c (cadr lowright))))
       b (sqrt (+ (* c c )(* d d)))
       reduction (/ b a)

   (command "CIRCLE" mark (/ b 10))
)  ;;; END DEFUN
```

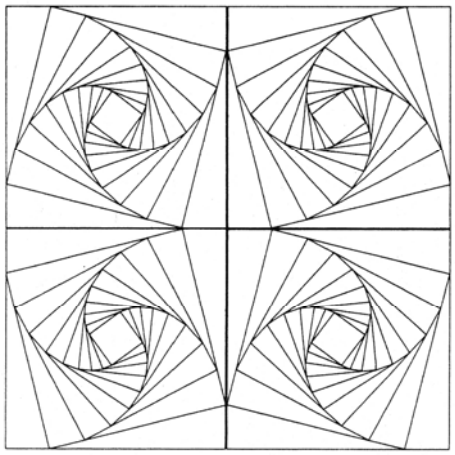



Figure 4: A parametric shape grammar program and resulting design from Microcomputer Aided Design for Architects and Designers.

At the end of the book, Schmitt exposed the advantages of the computer-assisted design process: the availability of a great amount of information in databases and the possibility of using machine inference, helping solve limited aspects of the design problem based on previous design solutions. According to him, eventually, the designer's role would be to simply criticize machine-generated alternatives.

Once again, this book must be understood in his particular context. As Schmitt notes, at that time computers in design were just starting to be operated by the architects themselves, and his point of view was that architects should learn to operate computers properly if they wanted to be on control. In other words, architects should learn to program if they wanted to take full advantage of computers in design.

The greatest difference between his book and Mitchell, Liget and Kvan's is that Schmitt's programs are developed inside a CAD package framework (in this case AutoCAD), which avoids the need to program shapes from scratch. As noted by Seebohm (2001), "by programming on top of existing 3D modeling applications, such as AutoCAD with the AutoLisp language, it is possible to obtain useful and rewarding results within a single course (compared to learning a computer language and then developing graphics or modelling software)."

4.3. Generative Modeling

This coursebook was developed in 1995 by Paul Coates and his assistants for both an undergraduate and a graduate programs in architecture at the University of East London, where he was the director of the Centre for Evolutionary Computing in Architecture. It included an introductory text with many examples of applications of algorithms in design, especially in the modeling of urban morphology, with references to Hillier's space syntax theory.

According to the author, the purpose of the book was to "introduce the idea of computer generative modeling as a means of exploring key ideas in the design of the built environment, by way of a series of worked exercises ... which go beyond the standard customizing issues to allow experimentation with the automatic generation of form, or generative modeling" (Coates, 1995:7).

The exercises in the book explored techniques such as random numbers, recursion, shape grammars and cellular automata, and were developed in different programming languages: AutoCAD's AutoLisp (

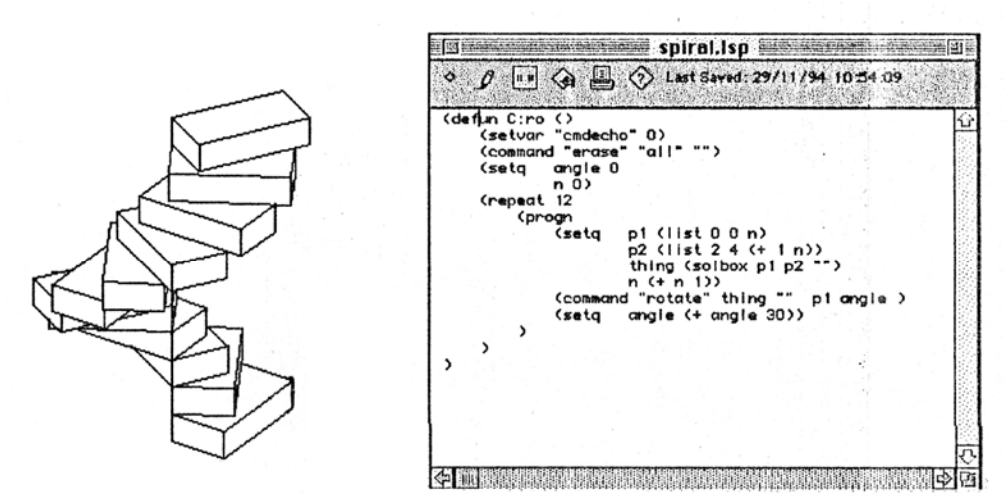


Figure 5), MiniCAD's Mini Pascal, GDL (Geometric Description Language), Macromedia Director's Lingo, and ArchiCAD's built in scripting language.

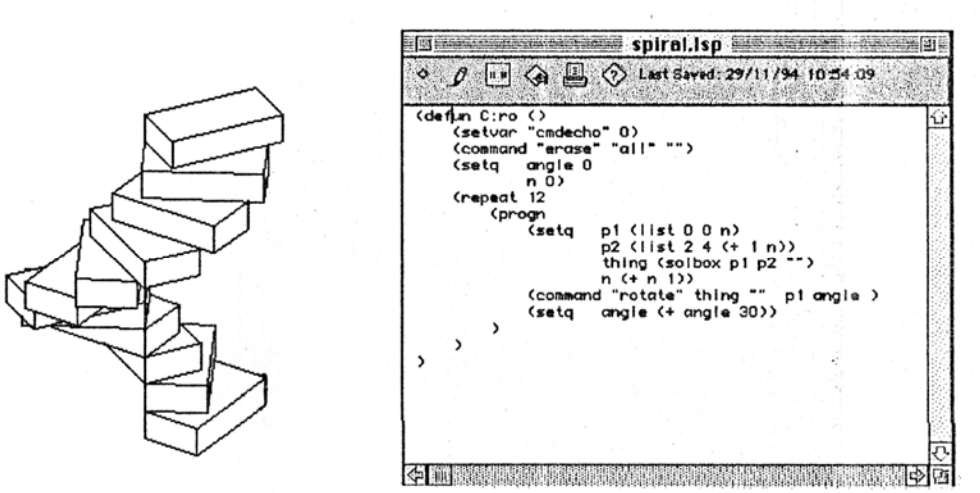


Figure 5: AutoLisp program for generating a 3D spiral with simple blocks, as an example of nested expressions in Coates's course book Generative Modelling.

As in Schmitt's book, Coates's exercises took advantage of a CAD package's pre-existing commands and macros for building basic shapes. However, by presenting examples in different languages, Coates avoided the focus on a specific CAD package, emphasizing the consistency of programming logic throughout different environments. According to him, "essential to book was the idea of an algorithm and ways of implementing algorithms in a CAD system." He went on saying that "the exigencies of computers today means that to get something done we have to make use of a computer language and do some coding, but it should be remembered that the code is just a means to an end" (p.7).

Coates's ideas about architectural design were very similar to Schmitt's: design was based on vocabularies and rules. But Coates introduced other ideas from the field of Artificial Intelligence, such as cellular automata, artificial life, chaotic behavior and self-organization, pointing out the complexity of architectural systems and how computation and programming could help us understand that complexity though modeling.

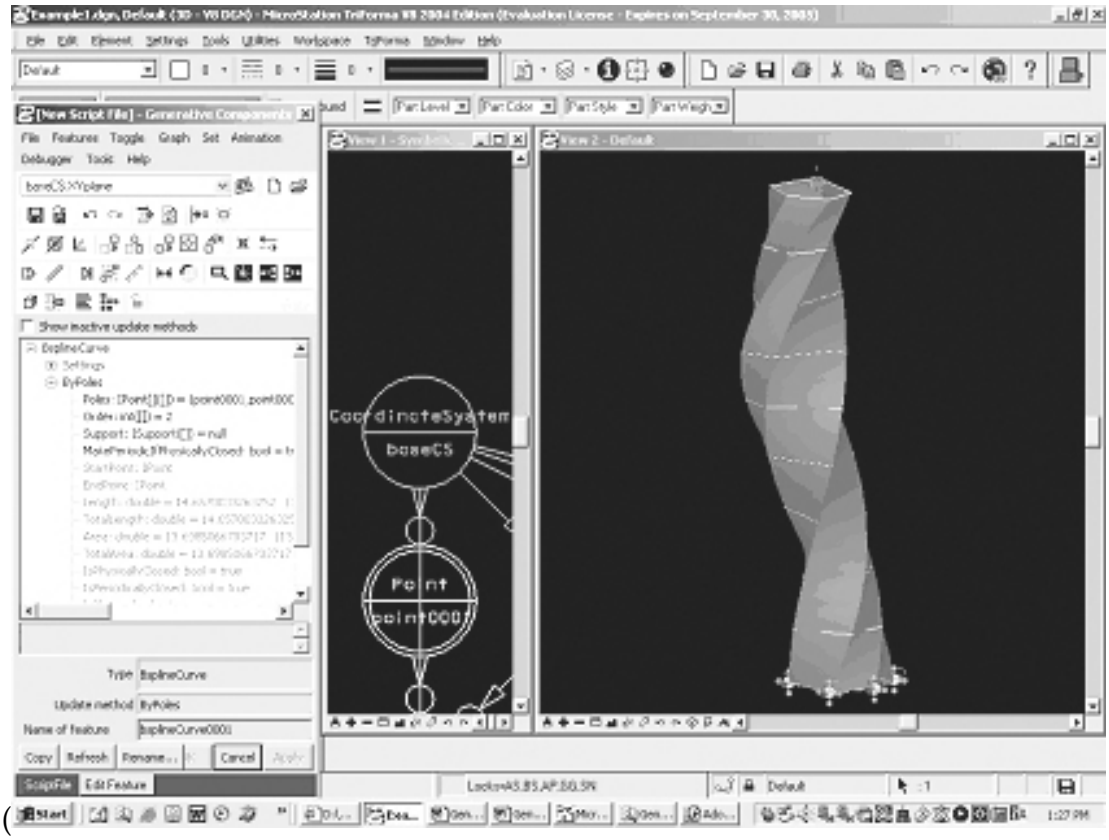
5. A CONTEMPORARY EXAMPLE OF THE USE OF PROGRAMMING IN ARCHITECTURAL DESIGN

As noted by Moneo (2005), it has become fashionable to design organic-shaped buildings. However, DiCristina (2001) emphasizes that this "topological tendency" does not simply mean curved surfaces, but the study of the geometrical properties that remain unchanged when figures undergo continuous transformations, something that requires computational knowledge for understanding and implementing.

The new generation of architects must be able to develop designs that are adaptable to a continuously changing urban environment, and programming may play an important role in modeling these concepts to develop design through conditional dependencies. In other words, contemporary, architecture is fundamentally about relationships, and state of the art construction is characterized by the use of expensive materials produced with great acuracy, frequently through automated processes. A new generation of CAD software is being currently developed to respond to these new requirements.

Since 2003, the SmartGeometry Group, an educational charity sponsored by Bentley systems, have been organizing conferences, workshops and Summer schools with the aim "furthering advanced education and research in the area of advanced 3D CAD applications", and with the belief that "Computer Aided Design lends itself to capturing the geometric relationships that form the foundation of architecture" (Smart Geometry, 2006).

Their educational programs feature Generative Components



(Figure 6), "a model-oriented design and programming environment, which combines direct interactive manipulation design methods based on feature modeling and constraints, with visual and traditional programming techniques." (Smart Geometry, 2006). In this type of CAD software shapes can be defined in terms of topological relationships that can also include conditional statements and variable values (Aish, 2005). For example, beam 1 may be defined as starting at the edge of beam 2 if it is long enough, or else starting at the middle of beam 3.

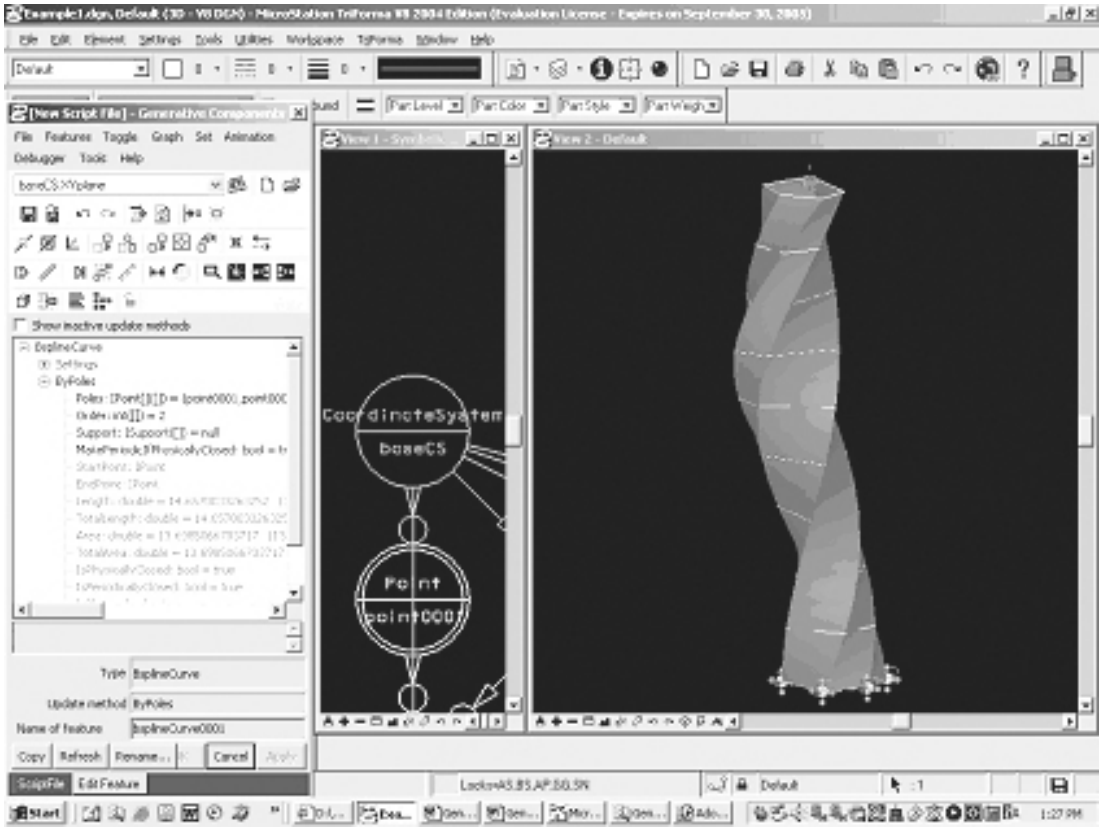


Figure 6: Generative Components software, with code window on the left.

However, according to Day (2006), to fully take advantage of a software like this, the designer needs to be "part programmer", which requires "a logical approach to problem solving." He cites Aish, who affirms that "geometric skills, compositional skills and algorithmic skills will be the key to future design." Generative Components has already been tried in the design of complex buildings such as

6. A METHODOLOGY FOR TEACHING CAD PROGRAMMING

In this section propose a framework for the introduction of programming in the architectural curriculum in five steps. My proposal aims at making the discipline of programming both a means for the integration of other subjects, like in Papert's principle, and a way to improve the quality of the design because it can encourage students to make their design concepts more explicit.

This methodology is based on my own pedagogical experience at the School of Civil Engineering, Architecture and Urban Design at the State University of Campinas,

Brazil, where I have taught programming courses for undergraduate and graduate architecture students since 2002. Examples for steps one through three, with full VBA codes, have been published in Celani (2003).

Although I have used AutoCAD's Visual Basic for Application (VBA), a scripting version of Visual Basic, in my courses for practical reasons, I do not want to associate my pedagogical proposal to any specific language. Most CAD packages nowadays include their own scripting languages and built-in programming environments, and the ideas presented below should be possible to implement in any context, except for the last module, which requires an object-oriented language, such as VBA. Although not all the object-oriented features are available in VBA, it includes "class modules" that allow to define classes with properties, methods and event handlers. The advantage of VBA in relation to other scripting languages embodied in CAD packages is that is provides an easy way to develop interfaces. Besides, programs can be embedded directly in drawing files, which makes loading and unloading code unnecessary.

6.1. Step 1: Implementing calculations

The first step consists of writing simple scripts for implementing calculations - like those introduced in different building technology classes, such as structural analysis and HVAC - in the CAD environment. Through the implementation of simple scripts that make formulas available in a CAD environment, students can start seeing the usefulness of learning programming.

It is very easy, for example, to develop a script that calculates the window area needed for adequately ventilating a given footage of interior space. If the dimensions of a standard-sized window are given, the script can also calculate the number of windows that must be inserted. To be able to implement a script like this, students simply need to know about variable types, operators, and optionally some interface interaction, if the CAD program chosen offers a simple way to develop user interfaces.

6.2. Step 2: Parameterizing shapes

Although most CAD programs include macros for drawing customized rectangles, oblongs end even more complex objects, such as staircases, through the specification of parameters, it can be helpful to learn how to parameterize one's personal design elements.

The proposed approach starts by exploring the CAD package's parametrical possibilities until its limitations are reached Monedero (1997). Then, students can develop their own parameterized shapes. One interesting idea is to encourage them to identify families of buildings that differ only in certain parameters and thus develop a program that allows to draw all these examples by inputting different variables.

6.3. Step 3: Repeating code

The next step in this pedagogical strategy consist of automating repetitive tasks, such as taking measurements and counting items to feed calculations. A script can be written, for example, to automatically calculate the total window area for a room, and then compare it to the room footage to see if ventilation is adequate. Automation of repetitive tasks can also be applied to the drafting of architectural elements such as stairs and railings.

For this purpose, code-looping techniques must be introduced. "For each" structures allow iterating through a collection of objects, searching for those that were specified, such as the windows that need to be measured. "Do while" structures, on the other hand, allow to repeat actions as many times as needed. Such is the case, for example, when steps must be distributed in a staircase, according to given proportions.

6.4. Step 4: Algorithmizing the design process

The next step in our method starts with a reflection about design methods. It starts with discussions about which parts of the design process can be automated and how a design problem can be structured. Design parameters, constants, and constraints

must be established. The concept of conditionals - if then else - is then introduced, along with the idea of nodes in a state-action search tree. The aim is to show students that the same problem may have different solutions, represented by different designs developed with different strategies.

According to Stiny (1978:208), "because an algorithm must be specified in such detail, just the attempt to construct an algorithm for a given process provides an excellent means of exploring the process in all its aspects and features."

A typical example is the algorithmization of a simple layout, such as a bathroom or a parking lot. This can be done by setting up step-by-step procedures that result in the desired layout. However, design intentions must be made explicit in this process, which involves dealing with unknown circumstances along the way, such as "will the width of this lot be wide enough for displaying five car rows at 45o ? ".

6.5. Step 5: Defining architectural types

In The Logic of Architecture, Mitchell (1990) establishes an analogy between architectural types and class definitions in object-oriented programming languages. According to him, both have essential and accidental properties. The former are characteristic of the type, while the latter are related to specific site conditions, being decided at instantiation time. The comparison seems a good opportunity to establish a discussion about using architectural types as heuristics - or shortcuts - to find solutions to design problems that are likely to work out. To be able to implement an architectural type computationally, students must first be introduced to object-oriented programming concepts, such as class definition, object instantiation, object properties and methods, encapsulation, inheritance, abstraction and polymorphism.

A typical example is the definition of a class of detached houses. Although all houses have only one kitchen and living room, different instances may have different numbers of bedrooms and bathrooms. Similarly, although all houses have doors and windows as façade composition elements, some may be yellow while others may be blue. At the same time that examples from architecture may help students

understand the new computational concepts, these concepts may as well teach them how to work with typologies in a more explicit fashion.

7. DISCUSSION

The CAD programming courses taught at FEC-UNICAMP in the last two years have shown that the introduction of computational design concepts before or along the introduction of programming techniques is fundamental to make students understand its relevance. With this knowledge, students are much more interested in learning computer programming, overcoming typical prejudices, because they can foresee meaningful applications for it. More than that, programming techniques can teach them how to structure design problems.

Students can be taught to use programming as an "administrative tool", as in Papert's principle (Minsky, 1988), to develop implementations that sew together knowledge acquired in different courses. Even Architectural History contents may be implemented in the form of programs that generate buildings in certain styles and with certain proportions, which requires students to have a deep understanding of their fundamental principles.

Programming may also be useful for exploring new shapes. Students can develop their own tools, or use the new generation of parametric CAD software, which will probably progressively require more programming skills. Besides, the introduction of programming in the CAD curriculum may shift the present focus on the use of computers exclusively for representation, and emphasize its role on the process of design, rather than on products, such as suggested by Oxman (1999).

In summary, CAD programming is becoming increasingly important in architectural education, and it presents many pedagogical possibilities. For example, the development of applications can help students establish relationships between different subjects, acting as a binding factor. The application of CAD programming in the definition of form, with the use of a new generation of parametric CAD software, on the other hand, may lead to a whole new concept of architectural

aesthetics, in which adaptability plays a crucial role. What is most important is to allow this introduction to occur in a contextualized environment, always with the necessary theoretical framework support. Future work on this topic must necessarily include the development of strategies for effectively introducing programming in the design studio.

8. REFERENCES

AISH, R. From intuition to precision. In: PROCEEDINGS OF ECAADE, 2005, Lisboa. **Anais...** Lisboa: IST, 2005. p. 23-25.

BENTLEY, P. **Evolutinary Design**, San Francisco: Morgan Kaufmann, 1999.

BROADBENT, G. **Design in architecture: architecture and the human sciences.** London: John Wiley & Sons, 1973.

CELANI, G. **CAD Criativo**. Rio de Janeiro: Campus-Elsevier, 2003.

COATES, P.; THUM, R. **Generative Modelling - Student Workbook**. 1995. 125f. Apostila. University of East London, Londres.

DAY, M., The smart revolution. **AEC magazine**. Chicago, vol.53, n.12, p.63-65, 2006. Disponível em: <http://aecmag.com/index.php?option=com_content&task=view&id=82&Itemid=35> Acesso em: 29 mar. 2006.

DiCRISTINA, G. The Topological Tendency in Architecture. In:____. **Architecture & Science**. New York: Wiley, 2001. Introdução, p.21-53.

EASTMAN, C. Fundamental problems in the development of computer-based architectural design models. In: KALAY, Y. (Org.). **Computability of design**. New York: John Wiley and Sons, 1987. Cap. 32, p. 450-465.

FOLEY, J. D.; Van DAM, A.; FEINER, S. K., HUGHES; J. F., PHILLIPS, R. L. **Introduction to computer graphics**. Reading, MA: Addison-Wesley, 1997.

HAREL, I.; PAPERT, S. **Constructionism**. Norwood: Ablex, 1991.

MARK, E.; MARTENS, B.; OXMAN, R. The Ideal Computer Curriculum. In: ECAADE, 2001, Helsinki. **Anais...** Helsinki: University of Technology, 2001. p.229-232.

MINSKY, M. **The Society of Mind**. New York: Touchstone, 1988.

MITCHELL, W. J.; LIGGET, R. S.; KVAN, T. **The art of computer graphics programming.** New York: Van Nostrand Reinhold, 1987.

MITCHELL, W. J. After word: the design studio of the future. In: MITCHELL. W. J. & MCCULLOUGH. M. (Orgs.) **The electronic design studio**. Cambridge, MA: The MIT Press, 1990. Cap.32, p.450-465.

MONEDERO, J. Parametric design: A review and some experiences. ECAADE, 1997, Vienna. **Anais...** Vienna: Vienna University of Technology, 1997. p.29-35.

MONEO, R. **Theoretical Anxiety and Design Strategies in the Work of Eight Contemporary Architects**. Cambridge, MA: The MIT Press, 2005.

MONTAGU, A. et al., 2001, Developments in Latin America – A Field Report, In: ECAADE, 2001, Helsinki. **Anais...** Helsinki: University of Technology, 2001. p.28-32.

NEGROPONTE, N. **Soft architecture machines.** Cambridge, MA: The MIT Press, 1975.

OXMAN, R. Educating the designerly thinker. **Design Studies**. Cambridge, v.20, n.2, p.21-31, mar. 1999.

PAPERT, S. **Mindstorms.** New York: Basic Books, 1980.

RESNICK, M. **Turtles, Termites and Traffic Jams. Cambridge**. MA: The MIT Press, 1994.

RICHENS, P. The next ten years. In: **Computers in architecture: tools for design.** Essex: Longman, 1992. cap.2, p.21-53.

ROE, A. G., **Using Visual Basic with AutoCAD.** Albany, NY: Thomson Learning / AutoDesk Press, 2001.

SCHMITT, G. **Microcomputer Aided Design for Architects and Designers**. New York: John Wiley & Sons, 1988.

SEEBOHM, T. The Ideal Digital Design Curriculumn: Its Bases and its Content. In: ECAADE, 2001, Helsinki. **Anais...** Helsinki: University of Technology, 2001. p.198-202.

SMART GEOMETRY GROUP. **Home page**. 2006. Disponível em: <http://www.smartgeometry.org>. Accesso em: 01/junho/2006.

STINY, G. **Algorithmic aesthetics: computer models for criticism and design in the arts**. Berkeley: University of California Press, 1978.

YAKELEY, M. **Digitally mediated design – using computer programming to develop a personal design process.** 2000, 155f. Tese (Doutorado em *Architecture: Design and Computation*). Massachusetts Institute of Technology, Cambridge, MA.