How the Approach of Digital Tools in Architecture Has Developed: The Case of Creative Programming

Patricia Domínguez(✉), Flavio Celis, and Ernesto Echeverría

Escuela de Arquitectura, Universidad de Alcalá de Henares, Alcalá de Henares, Spain
patricia.dominguez@uah.es

Abstract. The goal of this research is to figure out the different approaches to digital tools in architecture during the technological evolution of the last decades, which have determined the current coexistence of two kind of tools: the most extended ones, which trend to monopolization, and the alternative tools, more empowering, within which are the creative programming codes.

Thus, in a first phase of encounter with computational machines, designers, who also had the role of programmers, asked themselves questions in terms of the *why* and the *what for* of the use of tools in design. Subsequently, and because of the rise of software for commercial designs, users specialized in learning about them, focusing on the *how*, which in turn led to the separation of the designer-programmer into two independent roles. After the 2008 crisis several previous alternative tools used by a minority, such as Open-Source environments, caught the attention of the designers: they could regain the design process control by bridging programming, designing and execution with these technologies.

Despite many of these tools has already been included in design and architecture syllabi, the inclusion of creative programming codes is still an ongoing process, because its learning involves a greater knowledge of the digital environment. Processing code is shown as an example, used in some pioneer schools with the aim of enable future architects to understand digital environments from an active and empowered position.

Keywords: Digital tools · Creative programming · Processing

1 Background: From the *Why* to the *How*

1.1 Theoretical Grounds and Basic Technical Development: 1955–1965

The use of digital tools in design processes dates back practically to the genesis of the first computational machines, and its theoretical basis was strongly developed between the years 1955 and 1965: Noam Chomsky's theory on generative grammar and syntactic structures (1957) directly influenced the evolution of the first programming languages. Authors such as W. Ross Ashby (1956) introduced the concept of cybernetics, and Christopher Alexander, in *Notes on the Synthesis of Form* (1964), highlighted, "we must face the fact that we are on the brink of times when man may be able to magnify his intellectual and inventive capability, just as in the nineteenth century he used machines to magnify his physical capacity" [1]. To find out how that intellectual capability might be increased with the arrival of new computational machines, Alexander suggested a rationalization of the design process, both in the definition of good design as well as the *way* it adapts to a *context*, such as in the proposal to thresh out that *context* with the mathematical theory of sets. The goal is always to limit and verbalize internal processes that happen, whether conscious or unconscious, in the minds of creative designers.

In parallel, and not coincidentally, technological advances gave way to what became known as CAD (Computer Aided Design) technology, which germinated at MIT in a research project of the same name financed by the U.S. Air Force conducted between 1959 and 1967 [2]. The development team behind this technology (composed mostly of engineers) understood that, on the one hand, there was a man-machine process in every design (an idea closely related to the cybernetic theories of the time), and, on the other, that the goal of CAD was not the representation of the designed object, as in traditional graphic representation, but rather the construction in digital format of all the layers that make it up (form, size, materials, weights, costs, etc.).

1.2 Man-Machine in Conversation

At the end of the sixties, when technological development was ripe enough to begin its transfer to new disciplines and the main theoretic lines were underway, artistic, and academic circles witnessed the first results of practical experiments in the field of creativity and computation.

For the most part, this contact with primitive computers took place in universities. Thus, MIT continues to be the leading institution in the ongoing debate. In 1967 Nicholas Negroponte, a former student who participated in the CAD project founded the Architecture Machine Group. It was a multidisciplinary laboratory focused on the use of computational machines in design processes. Far from studying automation to save time and costs, it was about finding a true man-machine synergy that would elevate design possibilities and take it to the next level. Thus, Negroponte defined the scope of the new group as "the intimate association of two dissimilar species (man and machine), two dissimilar processes (design and computation), and two intelligent systems (the architect and the architecture machine)" [3].

Another instance of those first approximations to computational machines from intellectual spheres was a seminar on Automatic Generation of Plastic Forms held at the Computing Center of the Complutense University of Madrid (1968–1974), where study was underway of the use of computers in design processes from a two-fold approach, theoretical and practical. By employing primitive languages such as Fortran, the creators and thinkers were simultaneously programmers.

1.3 The Discourse Mutates

Starting in the 1980s there was a progressive change of attitude in the approach to using digital tools in design and architecture. For illustrative purposes the authors have performed a search in the Scopus (Fig. 1) scientific database. This was conducted by extracting the data of the number of publications of key topics (*creativity*, *computation*, *generative design*, *artificial intelligence*, *CAD* and *cybernetics*) and comparing their relative values (the absolute values evidently differ enormously when it comes to general topics and specialized ones, therefore they were not considered valid for comparative purposes). As can be observed, there is a considerable increase in the number of studies on CAD technology followed by research on Artificial Intelligence, while more general topics such as creativity and computation lose their priority over time, giving rise to more specialized fields.

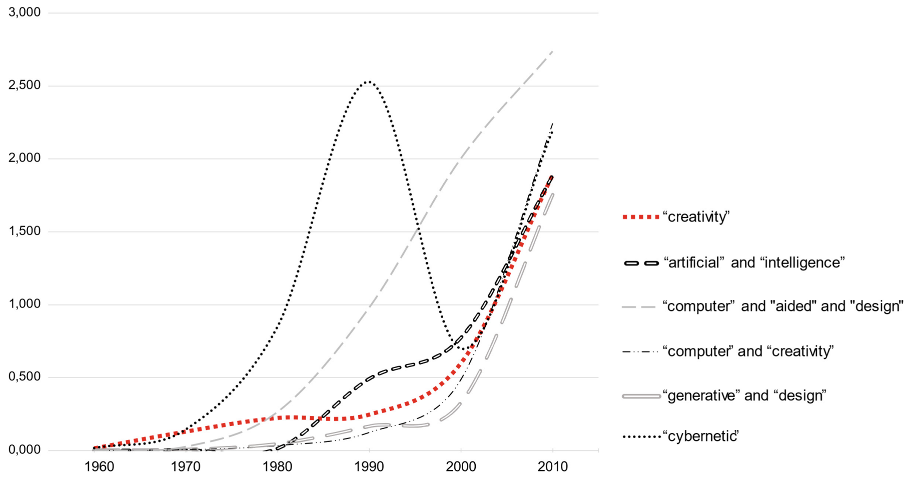


Fig. 1. Publications indexed in the Scopus database by year and topic, from 1940–2010 (relative values). Source: prepared by the authors.

This specialization is what changed the discourse of the *why* and *what for* from the preceding decades to the *how* narrative. The fundamental explanation of this phenomenon lies in the technological development of computation, not just in terms of the device (hardware) but also the content (software).

In the first years, companies that manufactured computers paid no special attention to software. Each machine had its specific programs and operating system. The gradual monopolization of the industry, however, (which enabled a certain standardization of physical components) along with the development of programming languages with a high degree of abstraction (which made them easier to understand and easier to use, and hence, had greater possibilities for development), changed the course of the market. It focused on the development of software and more advanced programming languages as a competitive strategy [4]. For example, C++ and Python were developed in the mid-to-late eighties.

Large software corporations concentrated their efforts on developing increasingly more complete and powerful applications, such as Autodesk and the release of AutoCAD in 1982. This signified a true paradigm shift in the production of engineers and architects. These powerful tools, however, had a disadvantage. Previously persons who used computers were also capable of programming them, or at least possessed basic knowledge of digital operation which enabled them to use the machines. But with the advent of large programs, the complexity of their operation was such that users had to devote all their time and energy to learn the use of the tools, putting aside the understanding of their programming and internal operation. Thus, there was a separation of the user–programmer [5], and they became independent roles increasingly specialized in their respective fields. More and more, they grew distant from each other and self-absorbed.

1.4 Does Form Follow Software?

This fixation on the *how* reached its zenith in the 1990s and in the first decade of the 21st century, in the years preceding the 2008 economic crisis. The gradual evolution of computer technology (boom of personal computers, the Internet) spurred the growth of increasingly gigantic design programs that required a staff with more qualifications and heightened expertise of use. Authors such as Lawson [6] questioned the contribution of CAD programs in creative discourse arguing that "the problem is that if the computer uses the wrong metaphor (…), it can inhibit the creative integration." Others the likes of Terzidis [7] pointed out that a CAD application offers a limited quantity of operations, hence it is not always possible to execute designers' ideas with commands.

Adopting a more neutral approach, what appears evident is that the metaphor employed by machines may delimit the creative process in some way.

Large architecture studios were aware of this condition and many chose a different path. They implemented specialized groups in computation that developed a specific software for the formal generation of each project. Some examples are the Specialist Modelling Group of Foster and Partners, founded in 1998, the Digital Technology Group at Herzog & de Meuron, and the Gehry Technologies company, which was founded as a group within the Frank Gehry studio to develop formal designs and was consolidated as an independent company in 2002, working not only for Gehry but also for other large studios seeking to materialize singular computer-generated geometric shapes [8].

Thus, it was in these years that specialization reached its zenith. This approach, focused on tools, brought about important advances in aspects such as BIM technology and generative design tools where the designer no longer designed the end product. Instead, it was designed by the digital tool. This self-preoccupation, however, also left in its wake a formal pretense that disregarded the relationship between architecture and its environment as "bubbles and bundles designed by computers" [9], and a theoretical foundation in terms of the use of digital tools that were far too light and optimistic.

2 Turning Point

2.1 From the 2008 Crisis to the Present Day

The 2008 economic crisis signified a halt, especially for mass architecture, which was left without resources to sustain and maintain itself over time. Joined with the notion of the climate crisis, the revision of architecture projects from a standpoint of environmental and economic sustainability necessarily caught up with the approach of what should be done in terms of the use of digital tools. Thus, they became demonized in a way, as they had been an integral part of the formal definition of exorbitant works of architecture.

Some authors, specialists in generative design, such as Asterios Agkathidis, reminded us that “tools and techniques are not to be blamed for unfortunate or irrelevant decisions of the planner. Neither does the computer liberate architects from their responsibility to the city and the society” [10].

This critical revision explains the surge of alternative tools stemming from the Open-Source and maker movements. Users, who extended beyond the so-called “nerd” community, found in these instruments and forms of production a path to empowerment and control. This control implies being aware of the *why* and the *what for* of the tools used in design, obtaining a holistic perspective of the creative process.

2.2 Incubation of the Open-Source Community

In parallel to the specialization phase of the nineties, some critical voices spoke against software monopolies. They explored alternative paths based on open collaboration. The Open-Source initiative, a methodology of creating programs free of charge, in an open, transparent, and collaborative way arose in 1998. Its methodological, ethical, and legal principals are mainly, the Free Software Foundation (FSF) founded in 1985 by Richard Stallman (also connected to MIT) and the General Public License, a software license promoted by the FSF to protect free software, and which has “an effect opposite to that of traditional copyright: (…) it prevents anyone, even the author, from limiting distribution” [4].

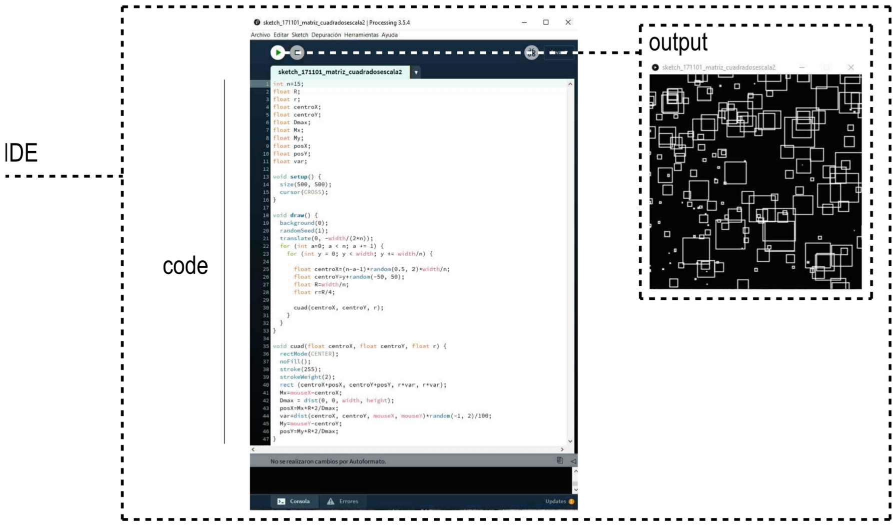
The structure of Open-Source projects brought about user empowerment, which, as highlighted previously, was no longer a user-programmer and had become “someone who uses software written by other” [5]. The new technological possibilities enabled a role reversion, and not only in the field of programming. Thus, just as the role of the user-programmer was recovered, a new role emerged in the field of design. It was the maker, a concept with united conception (design) and execution (production). This idea retrieved past roles: the artisans that had traditionally been designers and manufacturers, just as those who erected the great medieval buildings were both architects and builders” [11].

3 Creative Programming Language as Open-Source Tools

Of all the Open-Source tools used in design, creative programming codes are especially interesting. This is because given their language nature, they cover aspects of basic operation in digital environments. We are dealing with programming languages with eminently visual or sound outputs so that the creations are not produced by employing a sequence of commands but rather with an oftentimes written programming code.

3.1 Processing: A Creative Programming Language

Processing (Fig. 2) is an open code programming language that has its own programming environment (IDE, Integrated Development Environment), focused on the generation of multimedia creations (images, sound, videos, etc.) and hugely popular in different communities having no prior programming knowledge (students, designers, artists). Thus, they can learn the basic tenants of the language with an exploratory methodology. This "learning by doing" approach prevails in the maker movement, which is also an Open-Source environment. Plus, this language constitutes one of the most powerful tools in generative art.


IDE
code
output
sketch_171101_matriz_cuadradosescala2 | Processing 3.5.4
Archivo Editar Sketch Depuración Herramientas Ayuda
sketch_171101_matriz_cuadradosescala2
No se realizaron cambios por Autoformato.
Consola
Errores
Updates
sketch_171101_matriz_cuadradosescala2

```
int n=15;
float R;
float r;
float centroX;
float centroY;
float Dmax;
float Mx;
float My;
float posX;
float posY;
float var;

void setup() {
  size(500, 500);
  cursor(CROSS);
}

void draw() {
  background(0);
  randomSeed(1);
  translate(0, -width/(2*n));
  for (int a=0; a < n; a += 1) {
    for (int y = 0; y < width; y += width/n) {

      float centroX=(n-a-1)*random(0.5, 2)*width/n;
      float centroY=y+random(-50, 50);
      float R=width/n;
      float r=R/4;

      cuad(centroX, centroY, r);
    }
  }
}

void cuad(float centroX, float centroY, float r) {
  rectMode(CENTER);
  noFill();
  stroke(255);
  strokeWeight(2);
  rect (centroX+posX, centroY+posY, r*var, r*var);
  Mx=mouseX-centroX;
  Dmax = dist(0, 0, width, height);
  posX=Mx*R*2/Dmax;
  var=dist(centroX, centroY, mouseX, mouseY)*random(-1, 2)/100;
  My=mouseY-centroY;
  posY=My*R*2/Dmax;
}
```

Fig. 2. Example of a Processing application. A Processing language code is entered in the work environment (IDE) to generate visual and/or sound results. Source: prepared by the authors.

Processing originated in a beta version at MIT in the year 2001 as a multifunctional tool: programming language, software and learning system. After testing it for several years at the university, Ben Fry, one of the authors of the project, said the following, "I hope that it will also enable others to create new design tools that come not from corporations or computer scientists, but from designers themselves" [12].

3.2 What it's All About and What it Teaches

Processing is a text-based compiled language that feeds from Java (Fig. 3). When executing a program written in Processing, "you are actually running a Java program. (…) once you compile your code, the output is converted to Java class files (…) and the class files are interpreted within the Java Virtual Machine as your program runs" [13]. That is, while in the more popular CAD design programs there is no connection between its internal structure (programming) and its external structure (user operation), in Processing it is the same structure because the user (oftentimes designer) is also the programmer.

The approach to design using this tool deals with the analysis of the problem to solve (design inputs) from a computable and mathematical perspective.

Processing	Java
`if (key == 'a') {`	`public void keyPressed(KeyEvent e) {`
`  // Statements`	`  char key = e.getKeyChar();`
`}`	`  if (key == 'a') {`
	`    // Statements`
	`  }`
	`}`

Fig. 3. Comparison between Processing and Java codes for the same command: "if you press the 'a' key, …". Source: prepared by the authors based on Reas & Fry, 2007 code.

3.3 Current Teaching Experiments

The possibilities of use within design imply user empowerment. Makers become designers who also build and program their own design tools.

Although digital production has become a part of the teaching of architecture, programming is a pending task in most higher education institutions. In Spain, although there is a progressive inclusion in the curriculum of aspects related to digital tools, programming is still assigned an insignificant place. At the European level there are, however, some interesting examples (Fig. 4).

The University College of London (Bartlett School of Architecture) offers several Masters degrees in programming applied to architectural design with courses such as "Introduction to Programming for Architecture and Design." This subject teaches the basic tenants of programming applied to design using Processing.

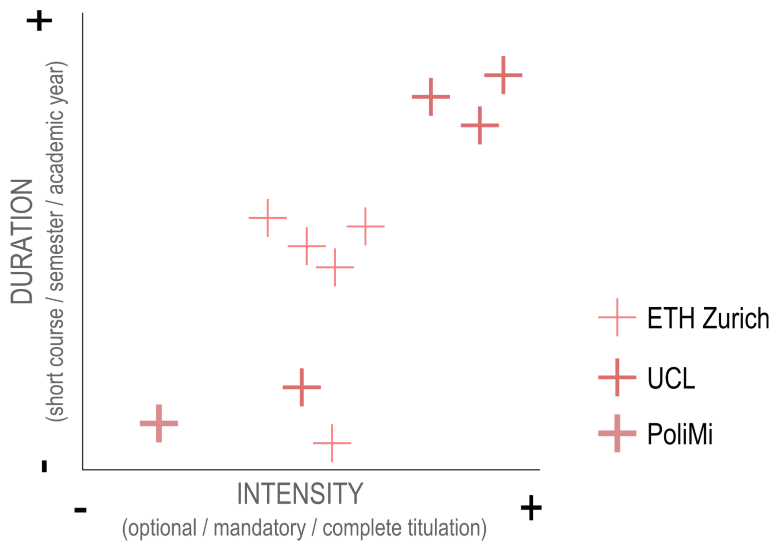


Fig. 4. Subjects related to creative programming in architecture and design degrees of three European universities (ETH Zurich, University College London and Polytechnic of Milan). Source: prepared by the authors.

Another noteworthy example is that of the series of subjects entitled "Mathematical Thinking and Programming" provided in the Architecture Bachelors program of ETH Zürich. Led by professor Ludger Hovestadt, it has a more holistic and pragmatic approach, "It's not about the how but the what. It's not about virtuosity in using digital tools but about understanding the code" [14].

Some university degrees have years of advantage in the inclusion of programming applied to other areas of design. For instance, the Creative Coding course (which also employs Processing as a learning instrument), offered by the Polytechnic of Milan as an elective for students that pursue a variety of studies such as interior design.

In all these references, the goal is shared: to pursue a basic alphabetization of programming by providing basic notions of the nature of computer operations, visualizing them in creative processes.

4 Conclusions: Empowerment Alternatives to Flee the Swarm

To conclude, the development of computational technology has been accompanied by diverse ways of approaching the technology from the fields of design and architecture (Fig. 5). The questions have changed over the last decades: from the *what* to the *what* for and finally the *how,* recovering the initial questions of a time where designers have the necessary tools to become empowered and play a part of the global process, leaving behind their role as passive users which characterized the intermediate phases.

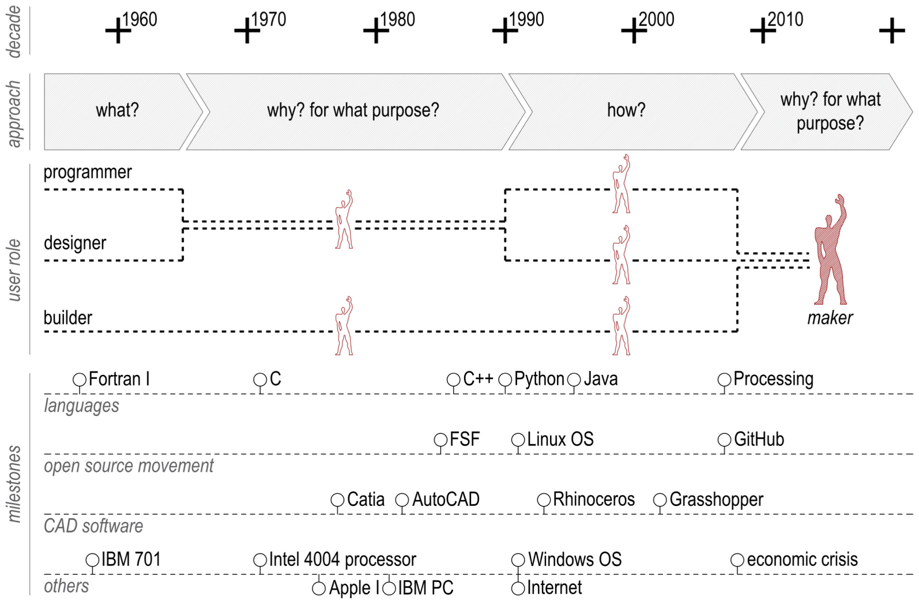


Fig. 5. Summary of the different approaches and roles in using digital tools in architecture design. Source: prepared by the authors.

Of all the tools that make this holistic perspective possible, those that flee from software monopolies are especially enriching. These are Open-Source projects and communities, which are more open to improvement and disseminating results. An innovative approach that is already being tested in architecture schools is the inclusion in the teaching curriculum of programming applied to design. Thus, future architects will possess the necessary skills to understand the operation of digital environments and even be capable of generating their own tools.

This maker approach in the training of new professionals also represents a certain position against the digital swarmed coined by B.C. Han to describe societies immersed in the digital environment [15]. An environment that is increasingly opaque to users, whose empowerment is made possible thanks to learning and using alternative means of programming.

References

1. Alexander, C.: Ensayo Sobre la Síntesis de la Forma, 5th edn. Infinito, Buenos Aires (1986)
2. Cardoso, D.: Algorithmic tectonics: how cold war era research shaped our imagination of design. Archit. Des. **83**(2), 16–21 (2013)
3. Negroponte, N.: The Architecture Machine. MIT Press, Cambridge (1970)
4. Fogel, K.: Producing Open Source Software: How to Run a Successful Free Software Project. Sebastopol O'Reilly (2005)
5. Nyhoff, L.R., Nyhoff, J.L.: Processing: An Introduction to Programming. CRC Press, Boca Raton (2017)
6. Lawson, B.: Cad and creativity: does the computer really help? Leonardo Int. Soc. Arts Sci. Technol. **35**(3), 327–331 (2002)
7. Terzidis, K.: Algorithms for Visual Design Using the Processing Language. Jossey-Bass, San Francisco (2009)
8. Peters, B.: The Building of Algorithmic Thought. Archit. Des. **83**(2), 8–15 (2013)
9. Fernández-Galiano, L.: Crítica y crisis. Cuaderno de Proyectos Arquitectónicos (1), 10–12 (2010)
10. Agkathidis, A.: Computational Architecture. BIS Publishers, Amsterdam (2012)
11. Sennett, R.: El artesano. Anagrama, Barcelona (2009)
12. Reas, C., Fry, B.: Processing a Programming Handbook for Visual Designers and Artists. MIT Press, Cambridge (2007)
13. Greenberg, I.: Processing: Creative Coding and Computational Art. Friends of Ed, Berkeley (2007)
14. Hovestadt, L.: Mathematical Thinking and Programming (Teaching Guide, ETH Zurich). Zúrich, Suiza. http://www.vvz.ethz.ch/Vorlesungsverzeichnis/lerneinheit.view?semkez=2021W&ansicht=KATALOGDATEN&lerneinheitId=147470&lang=en. Accessed 28 Nov 2021
15. Han, B.-C.: En el Enjambre. Herder, Barcelona (2019)