Computational Design Thinking and Thinking Design Computing

Arash Soleimani, Ph.D., Assoc. AIA
Assistant Professor
Kennesaw State University
Marietta, Georgia, USA
arash.soleimani@kennesaw.edu

ABSTRACT

In alignment with the rapid advancement of cyber-physical technologies in an information age, we are faced with complex problems that go beyond the kinds of challenges that designers had to deal with in the past. For many of these challenges we do not have established theories, methods, or tools to solve the problems. Therefore, it is critical for architects to not only have expertise in established design methods, but also to be able to rapidly and creatively develop new theories, skills, and technologies. This paper seeks to contribute to the core curriculum of architecture programs by exploring opportunities that benefit from advancements in computation as an innovative approach to teaching digital tools. The paper explores how computational thinking can be used in design as a new way of thinking, making, solving problems, and developing techniques and technologies to nurture creative processes, practices, and design outcomes.

The paper presents how advancements in technology and computation may change the process of design. Intelligent Design Systems are introduced as a successful example of teaching "Computational Methods" by the author in several architecture schools' core sequences in the United States. Computational Methods introduces students to computational thinking and fundamental concepts of computation through explorations with generative and analytical technologies. The goal of the course is to explore and elaborate the potential of computation and the role it can play as a part of one's design process; not as a collection of specific tools, but as a way of thinking about design.

INTRODUCTION

There is a growing interest in computational thinking (CT) as a novel genre in education for all disciplines (Berland and Lee 2011, and Grover, Cooper, and Pea 2014). CT has been defined for different subject matters; but in general, CT is considered an approach to understanding and solving problems. More precisely, CT is about understanding human behavior, planning and designing systems, and solving problems by employing fundamental concepts from computer science (Wing 2006). CT has been contextualized as a range of skills useful for successful problem solving in various areas of the STEAM, not just computer science. The increasing attention to teaching CT capacities across curricula by the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) highlights its significance (CSTA 2011). CT is viewed as a way to "magnify problem-solving skills needed to address authentic, real-world issues" in order to assist today's students in meeting "workforce demands of the future," and "to help solve some of the most pressing, complex problems of our time" (CSTA 2011).

The advancement in computing has also become a driving force in contemporary architectural design processes. To produce buildings of the future, designers must learn to think and work computationally, through the specification of processes, rules, and relationships. Mastering the state-of-the-art requires more than knowledge of software commands; it requires developing CT skills such as sketch storming, problem solving, and decision-making. CT enhances the ability to solve complex problems and innovate by effectively leveraging the strength of computers and people.

This paper presents how advancements in technology and computation may change the process of design. *Intelligent Design Systems* are introduced as the basis for the course entitled "Computational Methods", which has been taught by the author in several architecture schools' core sequences in the United States. *Computational Methods* introduces students to CT and fundamental concepts of computation through explorations with generative and analytical technologies. The goal of the course is to explore and elaborate the potential of computation and the role it can play as a part of one's design process; not as a collection of specific tools, but as a way of thinking about design.

DEFINING INTELLIGENT DESIGN SYSTEMS

Embedded in the course of *Computational Methods*, the discipline of *Intelligent Design Systems* (IDS) concerns people, environments, rules, and things. Such a discipline requires a clear conceptual framework, methodologies, and epistemologies. An in-depth definition of *Intelligent Design Systems* can be achieved through a focus on three themes: 1) Computational Thinking; 2) Parametrics; and 3) Digital Fabrication.

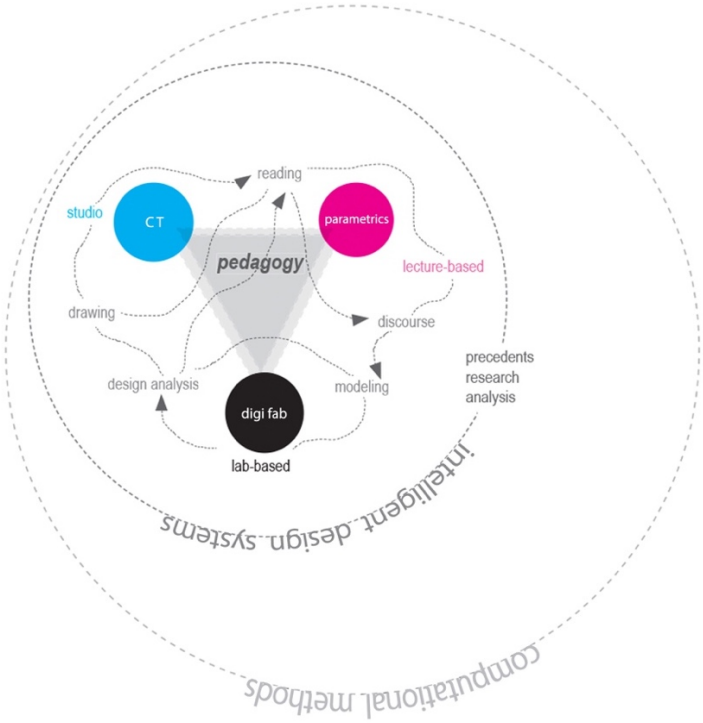

Figure 1. The discipline of Intelligent Design Systems embedded in the course of Computational Methods comprised of lecture- and lab-based sessions

1. Computational Thinking. the first foundation refers to the importance of taking advantage of the computational power of the computer. Computational tools offer the opportunity to design systems for solving problems that no one of us can tackle alone; instead a network of people and machines will replace the individual effort. CT reconstructs a seemingly complex problem to the one(s) that we know how to solve by decomposition, transformation, reduction, visualization, or simulation. CT is about developing models and systems not just for correctness and efficacy but also for aesthetics, simplicity, and elegance. Jeannette Wing (2008) introduced two A's to computational thinking: *abstractions* and *automation*. Abstractions are our mental tools to understand the surrounding world whereas automation is the mechanization of the abstractions. The abstraction process involves choosing the right abstractions as well as simultaneously managing at multiple layers of abstraction. For example, when you give driving directions, you would think about several steps at different scales such as transportation tools, starting- and end-points, main streets, turn-by-turn routes, traffic flows, and weather conditions. You may already know the steps, and then you would simply *recall* the address, or if you do not know the address, you may sketch out a high level map on paper and think about several ways to find the best route to the destination. In this example, *abstraction* is how effectively you map out the directions, and *automation* is how you design a system to be used in similar cases such as Waze and Google Maps.

A professional practice cycle that may reiterate through the computational thinking practices includes *decomposition*, *pattern recognition*, *pattern abstraction*, and *algorithm design* (Barr and Stephenson 2011). This practice cycle starts with breaking down a complex problem or task into smaller and more manageable parts (decomposition), which allows for noticing similarities and common differences that will help make predictions or lead to shortcuts (pattern recognition). Subsequently, during the problem solving process, unnecessary information will be removed making it easier to solve the problem and complete tasks

(pattern abstraction). Lastly, a set of step-by-step instructions is developed to complete the problem-solving process through creating an algorithmic solution (algorithm design) (Wing 2006). In the driving direction example, we thought about the problem at different levels of abstraction; some were recognized and recalled, some were intentionally removed, and some were defined in order to develop a step-by-step algorithm, which direct us to the destination. The CT practice cycle provides the confidence to reasonably use, alter, rearrange, and reconstruct a large complex system without understanding its every details. The integration of CT practices into different disciplines can offer new modes of thinking, designing systems, and solving problems.

Not very long ago, ubiquitous computing was a dream of living in a cyber-physical world where there is no line between the physical and digital worlds in everyday life. In the same way that yesterday's hope has become today's reality; CT is tomorrow's reality (Wing 2006). Computational thinking will be integrated into everyone's routine life when words like system, algorithm, decomposition, feedback, and debugging are part of people's everyday vocabulary. This paradigm shift is increasingly ingrained in various disciplines such as biology, mathematics, chemistry, physics, and also architecture. For instance, CT's contribution to biology goes beyond the ability to search through big data to find patterns. It is envisioned that computational models and algorithms can predict and represent protein structures' behavior in different environmental conditions (i.e., phenotype). In chemistry, CT can improve the optimization and searching of algorithms to identify best elements for chemical reactions while enhancing yields. An example of CT in geology includes modeling of the earth's atmosphere and thermal behavior for climatic predictions. Similarly, quantum computing is changing the ways physicists think; Nano-computing, the ways material scientists think; cloud computing, the ways sociologists and economists think; and associative (parametric) computing, the way architects think. Computation has a profound impact on both the perception and realization of systems (Wing 2008). Computational design thinking should represent an accumulation of multilayered concepts (abstractions) ranging from morphology and developmental biology to system theory, machine learning, and cybernetics. Kostas noted that:

"*The dominant mode of utilizing computers in architecture today is that of computerization; entities or processes that are already conceptualized in the designer's mind are entered, manipulated, or stored on a computer system. In contrast, computation or computing, as a computer-based design tool is generally limited. The problem with this situation is that designers do not take advantage of the computational power of the computer.*" (Menges and Ahlquist 2011)

The nature of architectural design processes can benefit from CT practices where an iterative design process is the core activity for designing systems. Marc Gross's comparison of the two words of '*program*' and '*design*' demonstrates a cogitable analogy between these two terms. The first part of the words, [pro] and [de], means *out*, *off*, or *forward* whereas [gram] and [sign] highlight the act of making a *mark* or *written form*. "*Therefore*, *design has always been computational*." (Gross 2007) For example, Notre-Dame de Chartres Cathedral in France, constructed between 1194-1220, represents ways of thinking computationally from a very long time ago. The architect mathematically and computationally calculated the structure to increase the size of the windows with external buttressing for enhanced level of illumination (Figure 2, top-left). For another example, San Galgano Abbey, built around 1218 in Italy, shows an algorithmic design with the integration of different levels of abstraction, pattern recognition, and generalization to formulate a systematic approach to designing the building (Figure 2, right). As a contemporary example, Frank Gehry's Dancing House (Figure 2, bottom-left) was designed and built with various modern computational tools such as NURB surfaces, mass customization of panels, double curvature, and a lot of computational techniques. What could be learned from theses precedents is the act of using *computation* during the process of design versus designing a building and then putting it into the computer, which contrarily is called *computerization* (Menges and Ahlquist 2011).

2. Parametrics. the second foundation of *Intelligent Design Systems*, *parametrics*, concerns rules governing the design process, starting from George Stiny's Shape Grammars (Stiny 1975) and evolving through the newly found interest in parametric design and algorithmic processes. Parametric design is increasingly becoming not only a method, but also a design philosophy. It offers a realm of possibilities rather than a dictated solution to designers, users, and clients. Parametric design involves a scripted association of multiple subsystems (i.e., automated algorithms), where there are correlations between subsystems such as the structural, façade, zoning, and circulation systems. An alteration in any subsystem results in changes in other systems. In other words, the focus is given to understanding different parameters

(abstractions) during the process of design. Taxonomy of parameters may include, but not limited to, mathematical, geometric, topological, representational, material, environmental, and human parameters (Jabi 2013). Parameters may be given different priorities during the design process in accordance to the needs, processes, and desired outcomes. In order to ingrain *parametrics* within the core curriculum of architecture program, I argue that three interrelated approaches (*systematic*, *algorithmic*, and *interdisciplinary approaches*) must be considered to advance the parametric paradigm.

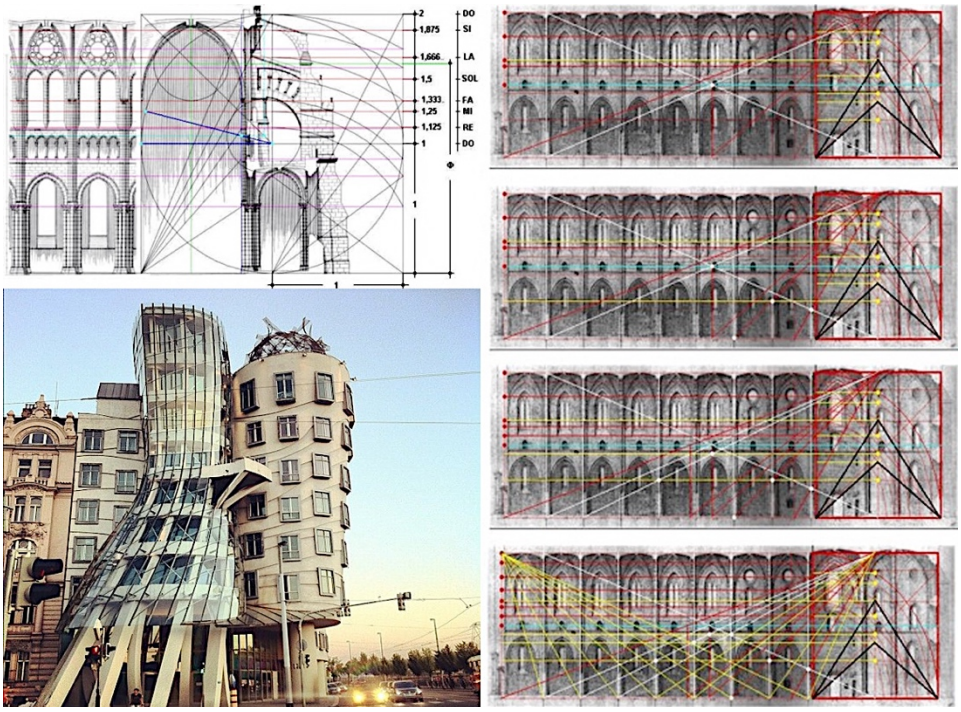

Figure 2. CT examples in architecture: Notre-Dame de Chartres Cathedral (top-left), San Galgano Abbey (right), and Dancing House (bottom-left)

a. System-based approach: architecture is an accumulation of several subsystems and in order to understand the whole system, we cannot simply study parts in isolation. The goal is to move from an object to a complex system comprised of spatial interactions, material parts, social constructs, and structural articulations. Such a system is in constant dialogue and exchange with the built and natural environments, and it requires some sort of intelligence and adaptiveness to coexist and coevolve.

b. Algorithmic, rule-based approach: an algorithm is a computational procedure including a set of step-by-step rules to solve a problem through decomposition, abstraction, generalization, deduction, and induction. Algorithmic approach is a mode of computational thinking, which operates mechanisms, functions, and relationships to create active relationships between the design intent and the design outcome. In other words, an algorithmic approach, as a vehicle for exploration, involves an encapsulation of processes and subsystems that lead us to alternative design outcomes.

c. Interdisciplinary approach: in today's world, we are faced with complex challenges and problems that might be different from the ones that we had to deal with in the past. For many of these challenges we do not have conventional methods and tools to solve them. It is critical to be able to rapidly and creatively develop new methods that may root in and benefit from various disciplines. The interdisciplinary approach relates to the intellectual foundation of the field of *Intelligent Design Systems*, which builds upon the confluence of different fields ranging from philosophy and biology to mathematics and computer science.

3. Digital Fabrication. the third foundation of *Intelligent Design Systems* is concerned with the relationship of process to product. It advocates the view that design is embedded in the tradition of attention to materials, details, and prototyping. As coined by Leach (2004), digital fabrication expands the traditional

notion through the use of emerging cyber-physical technologies. Despite earlier computational tools, new technologies are not incompatible with traditional understanding of materiality and craft. Instead, they can facilitate creative processes by offering parametrically fabricated models. Digital fabrication technologies allow precise control over the production of physical artifacts with offering automation, optimization, mass-customization, and quality control.

In a vibrant school of architecture, one should expect to witness all different approaches and streams of design ranging from CT to parametric design and digital fabrication. Pursuing these themes in a collaborative atmosphere would further ensure that the discipline of *Intelligent Design Systems* (IDS) is formulated in a rigorous and collective manner.

TEACHING COMPUTATIONAL METHODS

Teaching Methodology. in order to teach and integrate fundamental concepts of the discipline of *Intelligent Design Systems* into the core curriculum of architecture program, the course of "Computational Methods" was developed at two levels (Comp Methods I&II). The first class focuses on generative and analytical scripting where digital visualization, simulation, and information modeling are integrated in the early stages of design using analysis results as inputs of parametric systems. The second class explores the integration of cyber-physical technologies in the built environment where things are programmed to be adaptive, responsive, and reconfigurable. Both classes are offered twice a week comprised of lab and lecture components. The lecture component explores various theories and concepts informing the field of architecture such as Johann Wolfgang von Goethe's *Formation and Transformation*, Ernst Mayr's *Variational Evolution*, Christopher Alexander's *Systems Generating Systems*, and Gordon Pask's *Architectural Relevance of Cybernetics*. In each class, *Four-corner Discussion* is employed where students are randomly grouped to discuss their quotes from the readings. The groups will thus share their findings with the whole class. A follow-up lecture will be offered for more in-depth elaboration of the subject matter.

Students' scripting skills are nurtured during the lab sessions once per week. Most students attend *Comp Methods* classes with a basic knowledge of Rhino. The lab pedagogy is in contract with the typical computing lab culture, where students have to follow along with passive tutorials. Instead, the lab is comprised of several group activities, one-to-one instructions, and in-class assignments. Additionally, each lab session is videotaped and becomes available through the university's digital platform. This allows students to refer back to the class content and follow the instructions at their own pace.

In *Computational Methods*, students develop an in-depth understanding of different design parameters and the art of orchestrating those parameters into automated algorithms (e.g. developing Grasshopper and Dynamo scripts). The algorithms are developed in response to various given conditions such as structural systems (e.g. studying modulation and network), environmental control systems (e.g. studying lighting and acoustic behavior), and interactive systems (e.g. studying kinetics and programmable structures). During the first quarter of the semester, students choose a topic to pursue further research on associated terminologies, tools, and methodologies in order to discover how the given condition(s) can be transcribed numerically as inputs for an algorithmic design. The goal is to improve students' skills in translating and converting qualitative/quantitative data to readable information for functioning design algorithms.

The projects presented below cover examples of metric-based systems designed and developed in the areas of environmental control systems and interactive systems by undergraduate students in different architecture schools in the United States. Following the selected topic, each project shows student's articulation of Computational Thinking Practices i.e., Decomposition (D.), Pattern Recognition (PR.), Pattern Abstraction (PA.), and Algorithm Design (AD.).

COMP METHODS: RESPONSIVE ENVIRONMENTAL CONTROL SYSTEMS

Architects' desire to achieve high performance design is increasingly becoming popular with the advancement of digital technologies and environmental analysis tools. The initial step to an environmental design approach is to attain a clear understanding of environmental conditions, which can inform the design process. This increases the possibility of making environmentally responsive design decisions during different phases of design. Parametric tools allow for making informed decisions based on multilayered

complex information such as environmental data. The following projects, *Shading Analysis* and *Acoustic Analysis*, investigate how light and sound, as design parameters, can inform and be integrated into different steps of a design process.

Shading Analysis. one obvious change in architectural design processes that has occurred with the emergence of the Information Age is using analysis and simulation engines to make architecture more responsive to environmental changes and human needs. This project asked students a challenging question: "How can a façade system be enhanced in response to the building program and lighting variations?" With no prior knowledge of scripting, students developed their parametric understanding of solar behavior and its relation to a given site location (a south-facing gallery space with high level of transparency in the architecture school). In accordance with the existing program, a responsive, exterior shading device was designed with the primary focus of maintaining limited and controlled daylight for the gallery space (Figure 3). For this reason, much of the current illuminance is negated to allow for a more inward focus for the building. Even so, the facade still allows for light to penetrate through the openings based upon the angles of the sun during the extreme points of the year in relation to the human proportions and the depth of the space (Figure 3, bottom-right). The light penetration becomes smaller during instances that the sun may hinder the ability to view artwork and becomes larger in the areas that the sun does not interfere with the program inside. Additionally, the penetration becomes larger where the rest of the building potentially blocks out the western sun (Figure 3, bottom-left). The dual-rotation of the façade modules, based on azimuth and altitude angles, allows the facade to track the sun path. Through sun tracking, the modules can be normal to the sunrays, to block out all direct sunlight, or parallel, to allow full light through the penetrations. This duality allows the light entering the space to be controlled through an environmentally responsive façade system. The design was developed in Rhino's Grasshopper and DIVA for modeling, lighting analysis, simulation, and graphic representations.

CT PRACTICES	KEY PARAMETER(S): LIGHT
D. [a]	*How can a façade system be enhanced in response to the building program and lighting variations?* Program: gallery requires controlled, indirect sunlight for the visibility of the exhibits. Lighting: day and night lighting variations.
PR. [b]	Program: gallery is used from 9:00am to 7:00pm. Lighting: harsh solar gain happens around noon (south light) and during sunset (west light).
PA. [c]	N/A
AD. [d]	Generate a script, which reads azimuth and altitude angles as input parameters to inform the direction (rotation angle) for each façade module.

[a] Decomposition(D.), [b] Pattern Recognition(PR.), [c] Pattern Abstraction (PA.), [d] Algorithm Design (AD.)

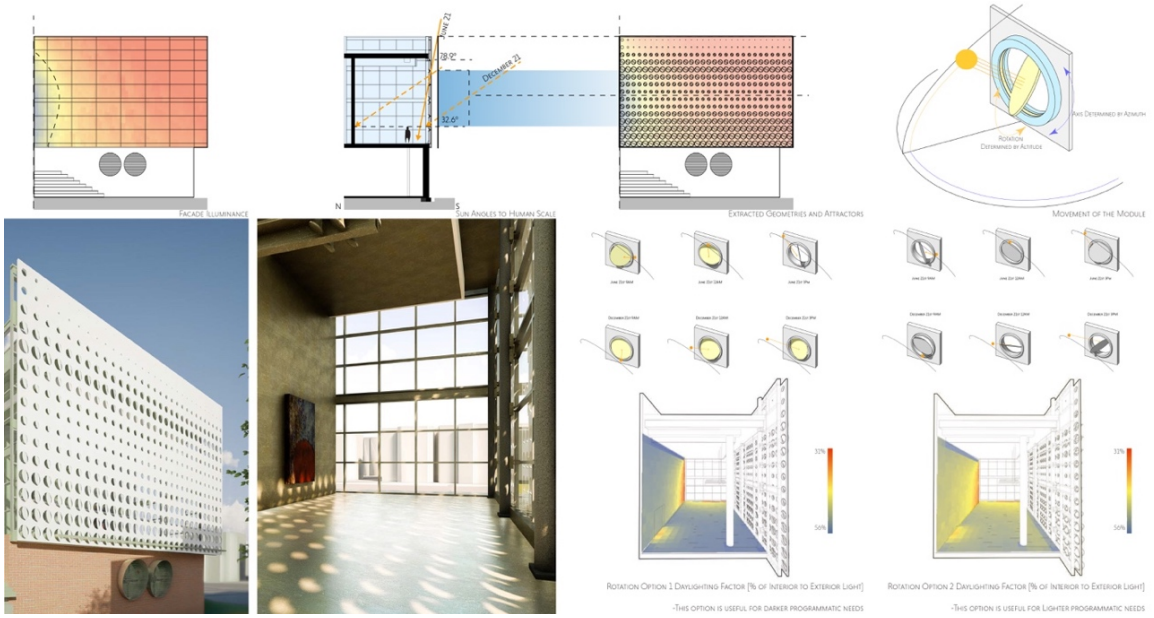

Figure 3. Student project: parametric façade system responsive to the solar system

Acoustic Analysis. acoustics are important factors for architecture; however, in most cases architects rarely consider sound as a driving force for design, except, perhaps, when designing acoustic rooms such as concert halls. The acoustic quality of a space can be controlled and enhanced by parametric modeling and computer programming techniques. This involves the design of the volumetric geometry of the room as well as its surface characteristics. For this project, students were asked to investigate three primary types of acoustic surfaces which can enhance the overall room acoustic quality: diffusers, reflectors, and absorbers. Digital computing was used as a method for pattern-finding solutions, acoustical simulations, ray tracing, and the analysis of multilayered complex sets of information. In particular, Grasshopper and Pachyderm plugins were used to first understand sound behavior within the space and then develop ray tracing diagrams. The same site location was assigned, the architecture gallery's south-facing façade, with the purpose of enhancing the acoustic quality of the space while accommodating different programs. To design an interior skin attached to the south façade, students developed several ray tracing diagrams to understand how sound behaves within the gallery with regards to different activities. Diffusers were directly used at the center of the acoustic surface to increase the presentation quality. Reflectors adorned the corners to deflect sound onto the nearby walls. The gallery's high ceiling could result in superfluous echoes and excess noise; therefore, absorbers were employed moving up the façade. The proposed acoustic system includes a series of diffusers at the bottommost sector of the existing building. These are then dissolved into a set of reflectors. In addition, the spacing between the reflectors gently allows controlled daylight into the gallery. The resultant design creates a balanced harmony of the visual and sonic environments.

CT PRACTICES	KEY PARAMETER(S): SOUND (primary) + LIGHT (secondary)
D. [a]	*How can a façade system be enhanced in response to the building program and acoustic qualities?* Acoustics: gallery requires a balanced room response that is not too dead or lively. Lighting: filtered light is desired.
PR. [b]	Program: gallery is used from 9:00am to 7:00pm. Acoustics: external distractions happen from 11:00am to 3:00pm. Lighting: harsh solar gain happens around noon (south light) and during sunset (west light).
PA. [c]	Acoustics: diffuse sound at center, reflect sound at corners, absorb sound on ceiling. Lighting: break direct sunlight through a shading mask.
AD. [d]	Generate a script for an acoustics surface, which allows reverberation time between 0.8-1.4 second with less than 45dB background noise.

[a] Decomposition(D.), [b] Pattern Recognition(PR.), [c] Pattern Abstraction (PA.), [d] Algorithm Design (AD.)

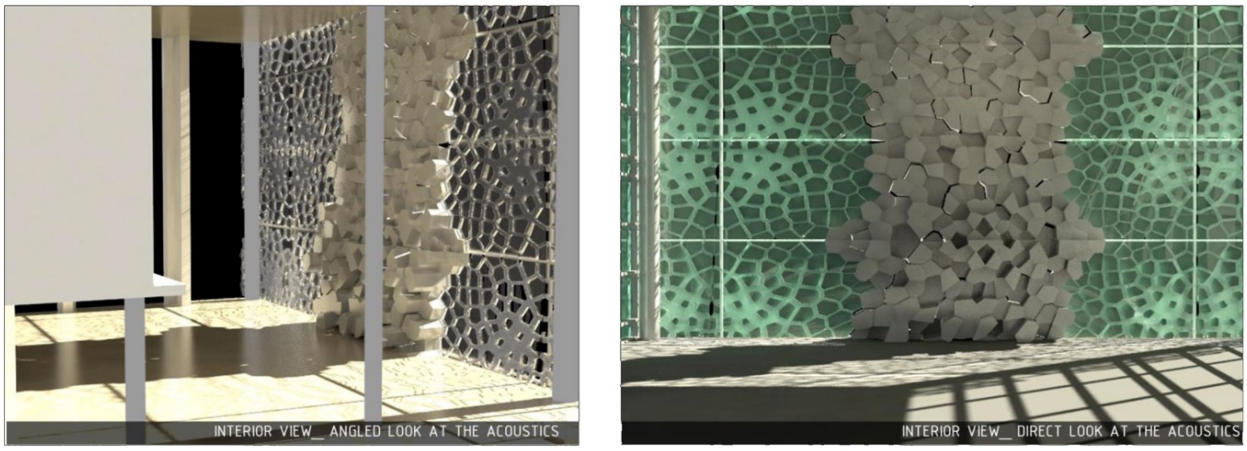

Figure 4. Student project: parametric acoustic surface comprised of diffusers, reflectors, and absorbers

Figure 5 represents the Grasshopper script exploration of the design process were parameters and rules define an algorithmic design. In this project, several parameters such as sound absorption coefficients and reverberation time as well as daytime, and altitude and azimuth angles were used to develop the design algorithm.

Unlike conventional design processes, a computational design supports an infinite range of underlying values to facilitate and improve decision-making and problem solving processes. Furthermore, a

computational procedure is not a linear, one-directional process; instead it offers non-linear correlations between the problem and the solution as they coevolve. In other words, during a computational procedure, the designer analyzes the problem and develops a step-by-step process through a scripting platform in order to achieve different architectural solutions. The final outcomes are not the end of the story since the designer may revise the initial steps and view the results through a live update. Scripting platforms such as Grasshopper facilitate the process through color-changing definitions and pop-up notes for diagnosing problems, debugging processes, and replacing constituent parts.

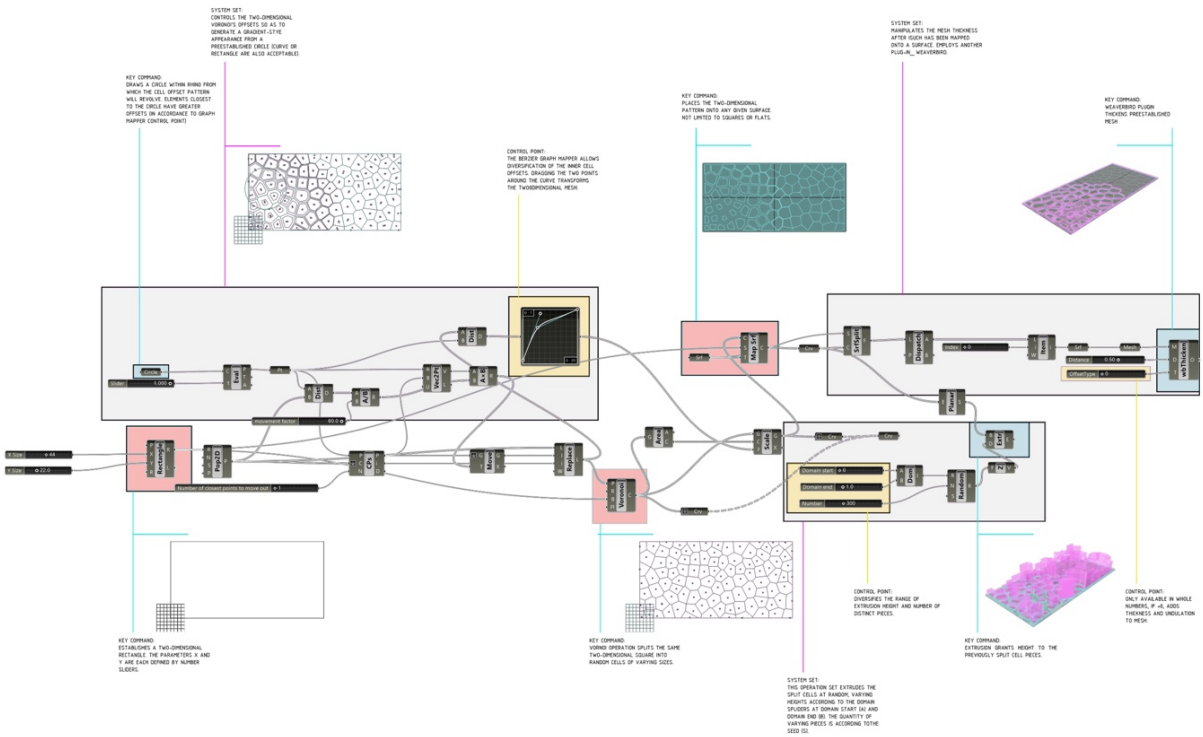

Figure 5. Student project: acoustic creations and exploration process of an acoustic surface through Grasshopper scripting

COMP METHODS: INTERACTIVE SYSTEMS

Interactive Interface. along with the advancement of computational devices and digital sensors, the notion that architecture can physically reconfigure to respond to our needs has influenced the architectural thought process. Different concepts from the fields of robotics, mechatronics, and computer science can be applied into an architectural context. This involves the integration of cyber-physical technologies such as electronics and sensors into the built and natural environments to make spaces more intelligent, performative, responsive, and adaptable. This project acquired students to choose a site condition such as light, wind, rain, or sound. With no prior knowledge of robotics and computer programing, students were instructed on programing sensors, which can detect environmental variations and respond through time-based systems. One group of students developed an interactive interface with moveable flaps. The project began with studying different patterns found in nature and ultimately led to a final parametric pattern, which represents the plant cellular structure at a microscopic level (Figure 6, top). Different parametric tools were used to create the desired pattern with appropriate aperture ratios and angles. The intention was to develop a façade system, which adapts to the sun path over time, harnesses solar energy through photovoltaic panels, and provides desired shading during the day. The design was inspired by the flower behavior, which opens up to receive solar energy. In order to mimic and create the actuation system, the Arduino platform was integrated. Arduino is an open-source, physical-computing platform comprised of a microcontroller board (Mellis et al. 2007). In the final prototype (Figure 6, bottom-left), the windows are powered by Arduino to open at certain angles when the light sensors receive daylight. Moreover, the apertures are equipped with photovoltaic cells to absorb solar energy during the day and power LED lights

overnight. In other words, when the photovoltaic cells absorb enough energy, the façade will close and illuminate at night through the embedded LED lights (Figure 6, bottom-right).

CT PRACTICES	KEY PARAMETER(S): LIGHT
D. [a]	*How can a façade system be programmed to be adaptable and responsive to the sun path?* Lighting: different façade configurations are desired for controlling solar gain and harnessing energy.
PR. [b]	Lighting: façade modules should open for solar gain (9:00am-3:00pm) and close for harnessing solar energy.
PA. [c]	N/A
AD. [d]	Generate a script, which reads azimuth and altitude angles as input parameters to inform the aperture size for each façade module. Write an Arduino code, which actuates façade modules to operate over time based on temperature and light variations.

[a] Decomposition(D.), [b] Pattern Recognition(PR.), [c] Pattern Abstraction (PA.), [d] Algorithm Design (AD.)

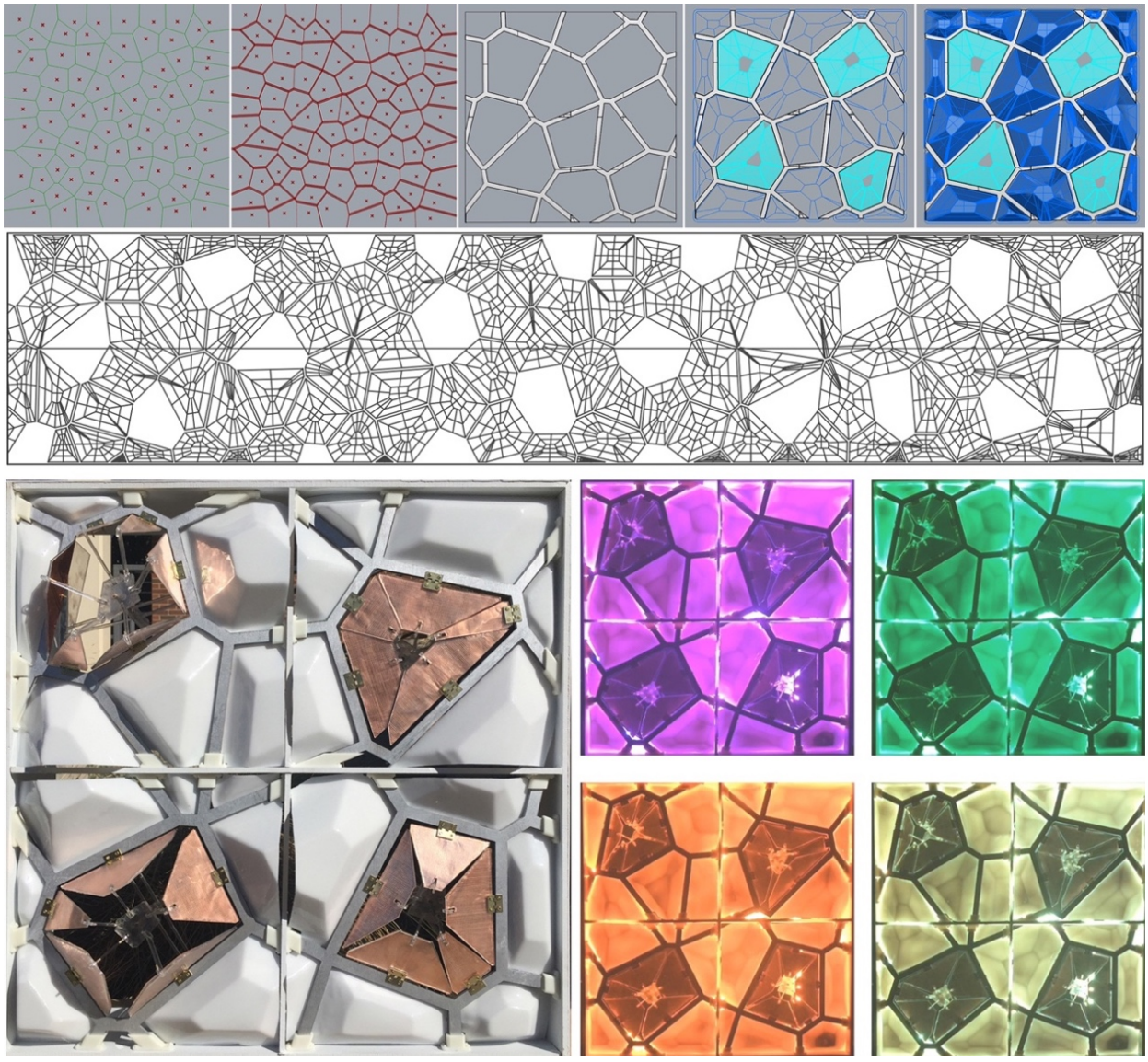

Figure 6. Student project: pattern development process through parametric tools (top), final interactive interface with color changing and movable components (bottom)

ASSESSMENT

To study the effectiveness of the course, post-class surveys and university course evaluations have been the primary tools for acquiring students' feedback and assessing their understanding of computational methods. Collectively, 153 out of 236 students participated in the surveys. The first concern is whether students feel the materials introduced to them such as teaching scripting are not necessary for becoming an architect. Studies indicate that if students do not see the application of what they learn, the class can be discouraging and demotivating (Pugh and Bergin 2007). However, the course has been successful at capturing students' interest. In response to the question: "How relevant is computational design thinking to your future career?", 91.7% of students positively responded to the question, and 45.4% of students said that the course has to remain a requirement in the architecture curriculum. *Computational Methods* has also been well received by the students. In response to the question: "Are you satisfied with what you learned in this class?", 95% of students agreed or strongly agreed with the statement. It has been evidenced in students' feedback that the course has been successful in making a challenging topic approachable for design students:

"*The content of this course contributed to my knowledge and intellectual skills.*" (student feedback, 2017)

"*The course content expanded my knowledge of the role of computational thinking in design and taught me important programing skills to apply in my design processes for future courses. I would attend another level of this course if it is offered in the future."* (student feedback, 2018)

This also shows students' engagement with the course when they express motivation to keep learning about the subject matter after completing the course. In the surveys, more than 80% of students mentioned that they would consider taking an advanced version of the class. Based on the feedback and evaluations, the course appears to be successful at teaching computational design thinking as another way of thinking about design steps. While it is evidenced that the course has raised students' awareness of computational methods in architecture, it is unclear whether students are able to apply the knowledge and skills learned in class to other subjects such as structural and environmental systems. In future versions of the course, I hope to address this issue by introducing cross-curricular contents and assignments in the course in parallel with other classes in the program.

CONCLUSION

Design thinking has always been computational whereas design tools have been evolving over time. For instance, to draw a 'line', we may use: 1) a pencil and a paper to draw the line by hand, 2) AutoCAD to specify the starting- and end-points of the line on the computer screen, or 3) a Grasshopper script and enter numbers to generate the line. In all these cases, there is a step-by-step process to develop a logic for generating a line, which is called computational thinking. CT refers to the thinking process involved in expressing solutions as algorithms, which are developed and performed by a computer (whether a human, a machine, or a network of humans and machines). Computation is not inherently about digital tools; it instead demands an explicit specification of inputs, formalization of behaviors, and expression of rules and relationships within dynamic systems.

With the advancement of computational tools and the emergence of the Information Age, the workflow limitations of the past are no longer a barrier for today's architects. However, the biggest challenge, perhaps, is to appropriately choose from an immersive world of information and integrate them into the design process for informed decision-makings. This requires an iterative development of *Intelligent Design Systems*, which can accommodate multilayered complex information through *computational thinking*, *parametrics*, and *digital fabrication*.

From a pedagogical standpoint, computation should be considered as a thought process, not as a tool. In other words, computation is not just about learning computer programs; it is about effectively structuring information and developing logics. Therefore, learning to design computationally must gradually evolve throughout the core curriculum. From the very beginning of the curricular journey, students should learn how to 'architecture' information during different stages of design. Students should not only learn how to

use software; but, more importantly, they should learn how the software functions. This helps them learn how to develop logics and structure information in various contexts. This mindset should be ingrained within the curriculum with a synergy among design studios, method, theory, technology, and structure classes. It is important to provide an environment where students can develop computational design thinking skills at different conceptual, developmental, technical, and representational levels.

REFERENCES

Barr, Valerie and Chris Stephenson. 2011. "Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?" *ACM Transactions on Computational Logic* 2(1): 48-54.

Berland, Mathew and Victor R. Lee. 2011. "Collaborative strategic board games as a site for distributed computational thinking.", Utah State University, *ITLS Faculty Publications*.

CSTA: Computer Science Teachers Association. 2011. *Computer Science Standards*. New York: ACM Publications. https://csta.acm.org/Curriculum/sub/K12Standards.html.

Gross, Mark. D. 2007. "Design Thinking is Computational Thinking.", *Carnegie Mellon University Seminar Series*.

Grover, Shuchi, Stephen Cooper, and Roy Pea. 2014. "Assessing computational learning in K-12." *Proceedings of ITiCSE'14*. ACM Press: 57-62.

Jabi, Wassim. 2013. "Parametric Design for Architecture." London: Laurence King Publishing.

Leach, Neil, David Turnbull, and Chris Williams. 2004. "Digital Tectonics." Michigan: Wiley.

Mellis, David A., Massimo Banzi, David Cuartielles, and Tom Igoe. 2007. "Arduino: An Open Electronic Prototyping Platform." *Proceedings of CHI'7*. San Jose, CA.

Menges, Achim and Sean Ahlquist. 2011. "Computational Design Thinking." West Sussex: Wiley and Sons, 10.

Pugh, Kevin J. and David A. Bergin. 2007. "Motivational Influences on Transfer." *Educational Psychologist* 41(3): 147-160.

Stiny, George. 1975. "Pictorial and Formal Aspects of Shape and Shape Grammars." Switzerland: Birkhauser Basel.

Wing, Jeannette M. 2006. "Computational thinking", *Computations of the ACM* 49(3): 33-35.

Wing, Jeannette M. 2008. "Computational Thinking and Thinking about Computing." *Communications of the ACM* 49(3): 33-35.