Digital Architectural Design as Exploration of Computable Functions

Toni Kotnik

Digital Architectural Design as Exploration of Computable Functions

Toni Kotnik

ABSTRACT

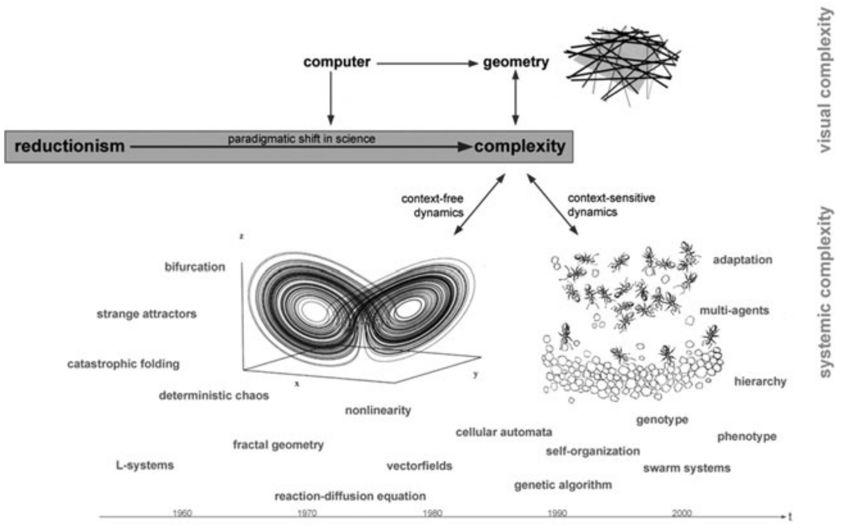
In recent decades, new methodologies have emerged in architectural design that exploit the computer as a design tool. This has generated a varied set of digital skills and a new type of architectural knowledge. However, up to now, a theoretical framework is missing that would allow for a comprehensive pedagogical agenda for the teaching of digital design in architecture. The present paper offers an attempt towards such a theoretical grounding based on the concept of computable functions. This approach results in an abstract and formal perspective on digital design that enables a grouping of contemporary digital design methods and an understanding of their logical relationship. On a theoretical level, it opens a path for the study of the mechanism that facilitates the transfer of concepts from various scientific disciplines into architecture.

1. Introduction

In 1963, Ivan Sutherland's Sketchpad program demonstrated that computers could be used for drafting and modelling [1]. By the mid-1990s, architectural practice without graphics software had become unimaginable, and today, digital design technologies have been adopted almost universally as the predominant means of production in architectural practice. Furthermore, digital technologies have enabled new methods of design, which has led to a re-examination of current design theories and educational concepts [2,3]. That is, architecture is taking part in an "intellectual revolution [that] is happening all around us, but few people are remarking on it. Computational thinking is influencing research in nearly all disciplines, both in the sciences and the humanities. . . . It is changing the way we think." [4]

A good example of such reshaping of discipline-immanent thinking by means of computation is the paradigmatic shift in sciences like physics or biology caused by the introduction of the computer as the primary tool for simulating and modelling natural processes [5]. Since the 1950s, this has resulted in a successive modification or even replacement of reductionism as the predominant paradigm of research. That is, the mechanistic understanding of nature and the continuous top-down reduction of the whole into parts has been exchanged from patterns of local interaction to the overall global arrangement of the parts as an emergent bottom-up property of the overall system (Figure 1). It is not surprising that architects became interested in these systemic models of nature due to related new methods of organisation and form-generation provided by computers and appropriate software [6]. As a result, over the past decade, systemic notions and concepts from science have diffused into architectural discourse and are currently being explored for design purposes [7,8,9].

► Figure 1. Paradigmatic shift in physics and related mathematical concepts in digital design.



The computer itself as the main bearer of this “intellectual revolution”, however, is not part of the ongoing discourse on the digital in architecture. It is overlooked constantly; the machine and its functionality is the blind spot of change. Computers, however, are actively shaping the way we as users approach design questions. It was Merleau-Ponty who pointed out that we as humans have to see our bodies not only as the physical context or milieu of cognitive mechanisms, but also as a living, experiential structure that is both biological and phenomenological [10]. Human understanding of the world, therefore, depends in large part on the interaction of the body with its environment. Every tool mediates this interaction because of its specific usage, thereby influencing the perception of the user and his way of thinking [11]. The paradigmatic changes in many scientific disciplines demonstrate that this is true in particular for the computer despite the seemingly unspecific neutrality of the machine that enables its versatile application.

Starting point of the following investigation is the functionality of the computer that is a closer look at the basic concept which enables its versatile application. As a physical machine, computers have their origin in John von Neumann’s design principles for a computing machine. These principles are implementations of Alan Turing’s more general abstract conception of computation and the related idea of a universal machine [12].

2. Computable function

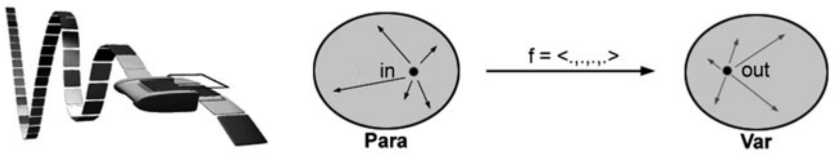
The versatility of a computer is built upon the generality of its main constituents: a machine, hardware, manipulating data according to a set of instructions, software. In this general setting, every piece of data takes the form of a finite sequence of bits, which is why it can be coded as a natural number. Thus, a program **p** can be viewed as a partial function on the set of natural numbers $\mathbb{N}$ meaning a function defined on a subset **Para** $\subseteq \mathbb{N}$ with output **out** $\in$ **Var** $\subseteq \mathbb{N}$ as the result of a computation of the input **in** $\in$ **Para**, that is **p**(**in**) = **out**.

It is this abstract setting for a computing device that facilitates the principal question of computability, i.e., for which function **f** does there exist a program **p** such that **f(in)** = **p**(**in**) for every valid input **in** $\in \mathbb{N}$. This question existed prior to the invention of modern digital computers. In the 1930’s, various mathematicians, including Alonzo Church, Kurt Gödel and Alan Turing, started to develop precise, independent definitions of what it means to be computable in order to find an answer to the Entscheidungsproblem, a mathematical challenge posed by David Hilbert in 1928 [13]. The problem asks for an algorithm that will take as input a description of a formal language and a mathematical statement formulated in this language and produce as output a Boolean value according to whether the statement is true or false. Church and Turing both were able to show that in general a solution to the Entscheidungsproblem is impossible, a result know as the Church-Turing theorem [14,15].

Intuitively, a task is computable if one can specify a finite sequence of instructions that when followed will result in the completion of the task. This intuition must be made precise by defining the capabilities of the machine that is to carry out the instructions because machines with different capabilities may be able to complete different sets of instructions, and therefore, may result in different classes of computable functions. However, it is a remarkable mathematical fact that all of the different precise definitions of computability lead to the same class of functions. In a practical sense, this means that if one can provide an intuitively convincing formal description of a step-by-step computation of a function, then one can find an algorithmic procedure in one of the precise definitions as well. This somewhat vague formulation is a commonly accepted hypothesis known as Church-Turing conjecture [16].

A by-product of Alan Turing's publication on David Hilbert's Entscheidungsproblem was an abstract mathematical machine-like model of what it means for a function to be computable as well as the description of what is now called a Turing machine [15]. This simple abstract device is one of the earliest and most intuitive ways to make the intuitive idea of computability precise, and the underlying logic is closely connected to the later development of computers.

► Figure 2. Abstract Turing model of computability.



A Turing machine **T** consists of an infinite one-dimensional tape divided into cells, a movable read-write head with a specified starting position, and a table of transition rules (Figure 2). Each cell of the tape contains one symbol, either 0 or 1, and the head can move along the tape to scan one cell at a time and perform three different activities:

- READ: read the content of the cell,
- WRITE: change the content into the opposite, and
- MOVE: advance to the next cell to the right or left along the tape.

A table of transition rules serves as the program for the machine. Each rule has a quadruple $<state_{actual}, symbol, action, state_{next}>$ meaning. If the machine is in $state_{actual}$ and the current cell contains *symbol* then take *action* MOVE or WRITE and move into $state_{next}$. The transition rules are labelled as $state_n$, and the execution of the program consists of the successive transition from one state to another. Furthermore, the program terminates if it reaches a situation in which there is not exactly one transition rule specified for execution [17]. Turing machines are very basic but powerful devices, and it can be proven that for every partial function **f** defined by a program **p** there exists a Turing machine **T** with **f**(**in**) = **T**(**in**) for every input **in** $\in$ **Para**.

In general, a mathematical function **f** is a well-defined relationship between two sets **A**, the domain, and **B**, the codomain, i.e. for every **x** $\in$ **A** there exists exactly one **y** $\in$ **B** with **f**(**x**) = **y**. Turing's computable function are mathematical function which, in addition, are quantifiable and algorithmic. That is the domain and codomain are subsets of the natural numbers $\mathbb{N}$ and the relationship between elements can be described through a sequence of simple formal rules. Already Turing himself was able to show that there are examples for relationships between two sets of natural numbers which are not computable [15]. That is the notion of a mathematical function is much more general than the concept of computability. However, the above discussion on Turing machines shows that everything that can be done on a real computer can, at least in principle, be formulated as Turing machine as well.

As a consequence, using a computer always means, without exception, the necessary limitation on computable functions as mediator between input and output. At the same time the set of computable functions is rich enough to allow for versatile applications of computers and to foster changes in the way we understand and think about design problems in architecture.

3. Formal description of design

Like every mathematical function, a computable one defines a relationship between two sets using a fixed number of rules: the domain of the function is the set **Para** of possible input, which is known as the parameter space in architecture, gets mapped by the function onto the codomain, the set **Var** of possible output, known as the space of variations.

Such an apparently abstract and theoretic description is a direct formal translation of the way one works with computers and becomes obvious upon drafting and modelling architecture using contemporary CAD-software. Every task in such a system is governed by these three constituents of a mathematical function: the activation of the tool, e.g., draw a line, as algorithmic rules of transformation; the selection of a pair of points as chosen element of the domain, which is the set of all possible pairs of points in space; and the resulting graphic output, the line between the points, as related elements of the codomain, which is the set of all possible lines.

This simple example of drawing a line shows that one can view the complete CAD-software as a finite collection of Turing machines $\{\mathbf{T}_1^{cad}, \ldots, \mathbf{T}_m^{cad}\}$, each one mediating between possible user input and consequential output, which are typically in graphical form, and displayed on the screen with every available tool of the software defined as a different Turing machine $\mathbf{T}_i^{cad}$. Consequently, the sequential process of drafting or modelling architecture by means of CAD-software can be thought of as the successive use of a finite number of Turing machines $\mathbf{T}_i^{cad}$, and thus define a unique sequence of Turing machines ($\mathbf{T}_1^{\text{proj}}, \ldots, \mathbf{T}_n^{\text{proj}}$) related specifically to the project. The concatenation of these machines generates a new Turing machine $\mathbf{T}^{\text{proj}}$ as a whole with project input as a subset of all earlier input, i.e., $\mathbf{i}_n^{\text{proj}} \subseteq \mathbf{in}_1^{\text{proj}} \cup \ldots \cup \mathbf{in}_n^{\text{proj}}$, and the displayed final model as the project output, i.e., $\mathbf{out}^{\text{proj}} = \mathbf{out}_n^{\text{proj}}$. It is this process of designing through formal nesting of Turing machines that constitutes the basis of the latest generation of CAD-software, like Bentley's Generative Component or the Grasshopper plug-in for Rhinoceros.

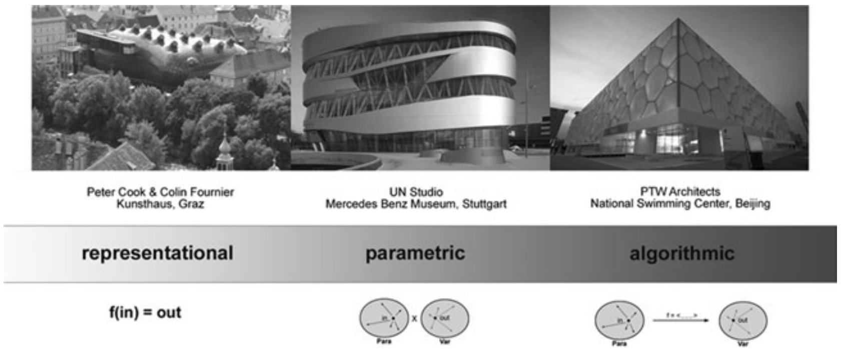
Consequentially, even the mere representational use of computational tools for drafting or modelling of architecture inevitably leads to a mathematical description of the achieved form in terms of an algorithm; however, this typically occurs unintentionally. But it is precisely the degree of awareness of the computational background and its intentional use that can be seen as the defining characteristics of digital design and at the same time as a possibility to mark the threshold between digital and non-digital design [2]. This doesn't mean that advances in human-computer interface design have to be ignored. However, the proper design of menus and icons only replaces the abstract and formal language of mathematics by a more intuitive language of signs and forms [18]. But it doesn't change the underlying logic of mathematical functions; it doesn't change the necessary degree of awareness of the computational background and its intentional use.

The purpose in the integration of computers as tools into the design process lies in the conscious exploration of the potential of the defining elements of a computable function as design tools: of the formal relationship between sets of entities, of quantifiable properties of these sets of entities, and of algorithmic transformations and interaction of different quantifiable properties. Digital design, thus, is not about the formalization of design processes or the automatization of decision making like for example in William Mitchell's *The Logic of Architecture* [19] but about the interaction of formal processes with architectural thinking; it's not about computerization but much more about computation [20].

4. Levels of design computability

Based on the formal description $\mathbf{T}^{\text{proj}}$ of digitally defined forms respectively by looking at the acquisition of the constituent elements of the related computational function $\mathbf{f}$ during the design process one can distinguish three major levels of computational utilisation that can be defined as levels of design computability: the representational, the parametric, and the algorithmic (Figure 3).

A representational level is characterised by the utilisation of the computational mainly as an electronic drawing tool. An example of such an application is the design of the Kunsthaus Graz by Peter Cook and Colin Fournier. There, NURBS have been used to digitally describe the



◀ Figure 3. Diagram of levels of digital computability.

shape of the outer skin of the museum based on an existing physical model [21]. In a similar way Coop Himmelblau used modelling software to represent and refine the oblique-angled crystalline form of the UFA cinema in Dresden [22].

In both cases, the computer enabled the activation of a geometric language that could not otherwise be controlled easily due to the incongruence of the geometry with the standard projections used by convention in architectural presentation [23]. That means, however, on a representational level there is no real perception of the computational nature that governs the digital environment. Rather, the design process is still in line with the visual reasoning of a conventional paper-based design approach. What exists is merely an awareness of a potential extension of the traditional geometric language of architecture that is present in the invisible mathematical description of digital design tools, i.e. the recognition of the existence of a relationship **f** between specific input **in** and unique output **out**.

A parametric level is characterised by the utilisation of such a given relationship **f** as a spectra of possibilities between input **in** and output **out** by means of continuous variation along the parameter space **Para**. A well-known example of this type of parametric variation is the design for the Waterloo train station in London by Nicholas Grimshaw. In this design the basic structure was a three-pin bowstring arch with an asymmetric placement of the centre pin due to the geometry of the platforms and its parametric propagation as a series of 36 identically configured trusses along the length of the train shed [24,25]. A more complex parametric interplay between all of the various elements of the architecture was used in the design of the Mercedes Benz Museum in Stuttgart by UN Studio [26]. There, the definition of the geometry of every single element of the building is dependent on the basic layout of the trefoil figure.

On the parametric level there is already a clear understanding of the existence of a computational relationship **f** between input **in** and output **out** and its integration into the design process as a scheme of interdependency between various parts of the design. The algorithmic description of the relationship, however, is not actively activated as design tool. The relationship is fixed and the focus is rather on the possibility of quantification of the input **in** that enables a controlled variation of the output **out**.

An algorithmic level opens up this relationship between input and output and is characterised by the utilisation of the formal description of **f** itself and its application as a design strategy. One of the first built examples based on an algorithmic design approach was the pavilion for the Serpentine Gallery by Toyo Ito and Cecil Balmond. The use of an iterative subdivision of adjacent sides resulted in a dense field of lines that defined the location of structural members as well as the distribution of openings for the enclosed cubic space [27]. The design of the National Swimming Center in Beijing by PTW Architects is another example of design development based on an algorithmic construction of the underlying geometric structure. The formal description of the space filling behaviour of foam bubbles and its abstraction as Wearie-Phelan geometry enabled the use of complex polyhedral cells as a construction system - a rational and efficient solution that appears to be random [28].

On the algorithmic level, therefore, the focus is on the development of computational design logic that is a sequence of algebraic, analytic, and geometric operations for the manipulation of data and its translation into architectural properties. It is the algorithmic description of the computable function **f** itself and the possibility of an individualised overcoming of the limitations of the inbuilt functionality of the used software.

All the definitions imply that the distinction between the three levels of design computability is not based on an evaluation of the architectural quality of the resulting design but rather on the level of understanding and maturity in the exploitation of the computational nature of the digital tools. That means the levels are not about forms but rather about forms of thinking and these forms of thinking have to be seen as a means of measure of digital craftsmanship that is a measure for the computational skilfulness in the use of the tools [29].

Clearly, the distinction between the different levels of design computability is gradual and they are opening up a spectrum of computational involvement with the representational and algorithmic as opposing poles. The above examples show that on the representational level the involvement with computation is very low and the form of design thinking is not driven by the computational at all. If digital design is defined by design methods that are driven by an occupation with computability than representational design methods have to be seen as non-digital. The spectra of design computability, thus, comprise the transition from non-digital towards digital design processes.

Parametric and algorithmic design computability are digital design methods and as part of the spectra they mark an extension of traditional non-digital design methods by computability. Kostas Terzidis has pointed towards such a characterisation of digital design by stating that "recent theories of form in architecture have focused on computational methods of formal exploration and expression. . . . For the last two decades, designers have been concerned with the use of computational mechanisms for the exploration of formal systems. These practices have attempted to readdress formal issues using new techniques and methods. Computational tools are central protagonists in this exploration" [30].

5. From geometry to functional description

In a digital design process, attention shifts away from the form-generating geometry itself towards the logic of the underlying computational function. Already René Descartes has demonstrated the possibility of such a transformation with the invention of analytic geometry, an algebraic description of geometric forms and operations, first published in his *Discours de la Méthode* in 1637 [31]. And it is analytic geometry that enables the functioning of CAD-software through the translation of geometric operations into computational functions. This implies that, from a designer's perspective, every computational function **f** and every algorithmic description **T** can be viewed as a generalised geometric operation. This understanding offers the possibility to look at the history of architecture in a new way and detect parametric and algorithmic potential in baroque geometric constructions [32] or antiquity [33].

The computational generalisation of geometry allows for new forms of creative expression since the use of computational methods in design requires some formalization of design thinking. The concept of computation, therefore, differs greatly from the concept of computerisation, which is a process of automation and mechanisation of data handling. "Computation is about the exploration of indeterminate, vague, unclear, and often ill-defined processes; because of its exploratory nature, computation aims at emulating or extending the human intellect. It is about rationalisation, reasoning, logic, algorithm, deduction, induction, extrapolation, exploration, and estimation. In its manifold implications, it involves problem solving, mental structures, cognition, simulation, and rule-based intelligence, to name a few." [20]

In digital design, the necessary formalization of design thinking by means of a computable function results in a precise description of the causal relationship **f** between input **in** and output **out**. The underlying notion of a mathematical function was first introduced by Gottfried Wilhelm Leibniz in 1692 in order to describe quantities that vary along the parameterisation of a curve [34]. This notion of function differs from the notion of functionality traditionally used in architecture. Functionality in an architectural sense is, in general, not stated explicitly as a causal relationship that can be computed, but is instead used more vaguely on a conceptual level as a link between an often non-quantifiable architectural purpose and its expression through form and materiality [35]. That is, functionality has to be seen as a design idea, i.e., an inner logic as driver of the design process, which results in an image of rationality, rather than a computable field of formal relationships.

With the rise of digital design tools in architecture, a family of formal strategies has emerged based on the use of of computable function to questions of performance [36]. These formal strategies constrain architectural *utilitas* to a measure of fitness that is an algebraic combination of quantifiable entities. In general, performative design approaches are driven by the numerical output of a formula as a measure of performative fitness. Consequentially, performative design strategies show a close affinity to bionic engineering because of the similarity in design thinking, which in both cases is based on the quantification of phenomena in nature [9,36]. From a theoretical point of view, such a turning towards nature is not unprecedented, but rather a recurring theme that runs through architectural history since Vitruvius [37].

For architectural design, the output-driven perspective onto the computational function of a performative design strategy is a pitfall because it encourages a tendency towards optimisation and with it, an economisation and closing up of architectural thinking towards parametric manipulation. The importance of an algorithmic description of the computational function, however, does not lie in the possibility of computing an optimal solution, but rather in the ability to control precisely the geometric relation between architectural elements under consideration. Formal design strategies are about the systematic exploration of the architectural potential of new types of organisation and the rediscovering of "the jubilation brought on by the perception of a hidden order, which Vitruvius's disciple felt when he came to perceive the power of proportions. . . . The poetics of computation becomes a *poiesis* in the strongest sense of the term: an art of procreation, . . . the architect becomes the demiurge of a world of forms in perpetual evolution." [38]

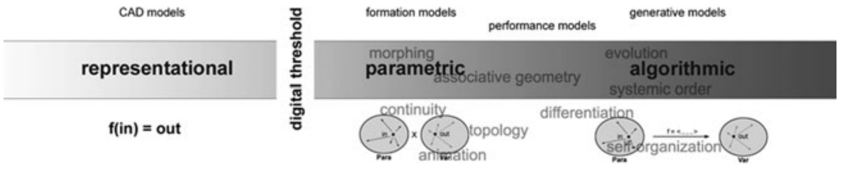
The notion of functionality in digital architecture, therefore, is not normative like in a performative design strategy but rather operative and comparable to the traditional vague understanding of functionality in architecture. This means, the efficiency of a computational function as formal design strategy has to be evaluated by architectural criteria, rather than numerical ones. The formalization of design thinking by means of computable functions cannot replace the design process but it can act as a framework for a more systematic investigation into digital design strategies used in architecture.

6. Digital design models

The investigation into the concept of computability and its importance for digital design in architecture shows that the computer is not a neutral tool, but rather is actively shaping the way designers are approaching the question of design. The defining elements of a computational function, that is a well-defined quantifiable and algorithmic relationship between two sets, have guided the definition of levels of representational, parametric, and algorithmic computability.

These levels are sufficient to classify contemporary digital design methodologies due to the limitation of all digital endeavours to computational functions. An example for this thesis is the taxonomy developed by Oxman [2]. In an attempt to organise current design theories and methodologies, Rivka Oxman has proposed five paradigmatic classes of digital design models according to various relationships between the designer, the conceptual content, the design processes applied, and the design object itself: CAD models, formation models, generative models, performance models, and integrated compound models.

For Oxman, CAD models are descriptive by employing various geometrical modelling and rendering software, but have little qualitative effect on design thinking and are essentially isomorphic with paper-based design methods. Thus, CAD models depict methods of digital design on a level of representational computability. Formation models are defined by Oxman as a structured geometric or formal digital process providing the designer with a high level of digital interaction and control resulting in "a performance of digital enabling that is perhaps the first charactering quality of digital phenomena." That is, for Oxman, formation models mark the threshold between digital and non-digital design methodologies (Figure 4). She subdivides the paradigmatic class further into topological models, associative design models, and motion-based models. In all of these subclasses, the interaction and control is based on the change of a set of parameters incorporated into a fixed field of geometric relationships. All formation models, therefore, are methods of digital design on the level of parametric computability. Generative models of digital design are characterised by Oxman by means of provision of a computational mechanism for formalised generation processes. Accordingly, this paradigmatic class is formed by methods of digital design on the level of algorithmic computability.



◀ Figure 4. Levels of digital computability and Oxman's digital design models.

In Oxman's approach, a performance-based design model is considered as a process of formation or generation that is driven by a desired performance. As a result, performance models do not form a set of methods distinct from the other paradigmatic classes, but rather are a subclass that stretches along formative and generative design models. That is why Oxman differentiates between performance-based formation models and performance-based generation models. Performance models play an important role in architecture with respect to the question of functionality of the digital design, but they do not provide a new level of digital computability due to the embedded nature of the class. The same is true for compound models, which are defined as a class of future paradigmatic digital design media.

7. Conclusion

The discussion on Oxman's taxonomy of digital design methods demonstrates that all of the various contemporary methodologies can be subsumed and compared under the concept of computability and the related concept of levels of digital computability (Figure 4). Furthermore, it demonstrates that the computer is not a neutral tool, but rather is actively shaping the way designers are approaching the question of design. The abstract concept of a computational function has the potential to serve as a theoretical framework for the conceptualization of digital design and future surveys on the development of digital design methods thereby answering a call "to formulate a theoretical framework that is suitable to the conceptualization of the subject. Such a framework must be capable of contributing a relevant theoretical structure to the field, whereas its own theoretical disciplinary contents must also illuminate seminal issues." [2]

A mathematical function makes explicit the governing rules between cause and effect. It is this characteristic that made the abstract concept of a function into one of the essential rationales of modern mathematics [39,40], and subsequently into the main objective in the ongoing process of mathematisation of scientific disciplines like economy or sociology [41]. Similar to this process in other disciplines, one can observe a tendency towards mathematisation of contemporary architectural discourse caused by the utilisation of formal design strategies [6,7,8,9,42,43].

Linking methods of digital design with the concept of a computational function offers a better understanding of such process of transfer of formal mathematical concepts into architectural discourse. The acquisition of digital methods into the design process has opened up a path along which mathematical methods of investigation from fields like topology, differential geometry, or complexity theory can diffuse into architecture. Concepts like continuity and smoothness, differentiation, bifurcation, morphing, or self-organization describe properties of mathematical functions that are now being used as concepts in design processes (Figure 4). The levels of digital computability enable a closer look at this process of diffusion and offer the possibility of a theoretical grounding of the relationship between the sciences and architecture on both the operative and theoretical levels.

Such abstract view of the digital in architecture opens up a new line of theoretical discourse in architecture that differs from the traditional metaphoric understanding of the relationship between architecture and the sciences [44]. The utilisation of the computer in architectural design offers a solid foundation for the systematisation of knowledge and methods in design. Computational functions make tangible *The Formal Basis of Modern Architecture* that Peter Eisenman has started to explore in his famous Ph.D. thesis in 1963, where he considered architectural form "as a problem of logical consistency, in other words, as the logical interaction of formal concepts" [45]. With the introduction of the digital, therefore, one is able to observe a major shift in architectural thinking towards formal methods that will fundamentally change our perspective of architecture as an academic endeavour and its relation to other disciplines like mathematics.

References

1. Kalay, Y. E., *Architecture's New Media: Principles, Theories, and Methods of Computer-aided Design*, MIT Press, Cambridge, 2004.
2. Oxman, R., (2006) Theory and design in the first digital age, *Design Studies*, 2006, 27(3), 229-265.
3. Oxman, R., Digital architecture as a challenge for design pedagogy: theory, knowledge, models and medium, *Design Studies*, 2008, 29(2), 99-120.
4. Bundy, A., Computational Thinking is Pervasive, *Journal of Scientific and Practical Computing*, 2007, 1 (2), 67-69.
5. Mussmann, F., *Komplexe Natur - Komplexe Wisssenschaft: Selbstorganisation, Chaos, Komplexität und der Durchbruch des Systemdenkens in den Naturwissenschaften*, Leske + Budrich, Opladen, 1995.
6. Weinstock, M., *The Architecture of Emergence: The Evolution of Form in Nature and Civilisation*, Wiley & Sons, Oxford, 2010.
7. Foreign Office Architects, *Phylogenesis: FOA's Ark*, Actar, Barcelona, 2003.
8. Lynn, G., *Animate Form*, Princeton University Press, Princeton, 1999.
9. Hensel, M. and Menges, A., *Morpho-Ecologies: Towards Heterogenous Space In Architecture Design*, AA Publications, London, 2007.
10. Varela, F.J., *The Embodied Mind: Cognitive Science and Human Experience*, MIT Press, Cambridge, 1991.
11. Gallagher, S., *How the Body Shapes the Mind*, Oxford University Press, Oxford, 2006.
12. Rojas, R. and Hashagen, U., *The First Computers: History and Architecture*, MIT Press, Cambridge, 2000.
13. Davis, M., *Engines of Logic*, Norton & Company, New York, 2000.
14. Church, A., An unsolvable problem of elementary number theory, *American Journal of Mathematics*, 1936, 58, 345-363.
15. Turing, A., On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, 2nd Series, 1936, 42, 230-265.
16. Copeland, B., *The Church-Turing Thesis*, Stanford Encyclopedia of Philosophy, 2002.
17. Barry Cooper, S., *Computability theory*, Chapman & Hall, Boca Raton, 2004.
18. Sharp, H., *Interaction Design: Beyond Human-Computer Interaction*, Wiley & Sons, New York, 2007.
19. Mitchell, W. J., The *Logic of Architecture: Design, Computation, and Cognition*, MIT Press, Cambridge, 1990.

20. Terzidis, K., *Algorithmic Architecture*, Architectural Press, Oxford, 2006.
21. Bogner, D., ed., *A Friendly Alien*, Hatje Cantz, Ostfildern, 2004.
22. Coop Himmelblau, Planning of the UFA-Palast with CAD, *in•form•Z, auto•des•sys Newsletter*, 1999, 14.
23. Szalapaj, P., *Contemporary Architecture and the Digital Design Process*, Architectural Press, Oxford, 2005.
24. Szalapaj, P., *CAD Principles for Architectural Design*, Architectural Press, Oxford, 2001.
25. Kolarevic, B., ed., *Architecture in the digital age: design and manufacturing*, Spoon Press, New York, 2003.
26. Un Studio and HG Merz, *Buy Me A Mercedes-Benz*, Actar, Barcelona, 2006.
27. Ito, T. and Balmond, C., *Serpentine Gallery Pavilion 2002: Toyo Ito with Arup*, Verlag der Buchhandlung Walther König, Köln, 2002.
28. Pohl, E. B., *Watercube: The Book*, dpr editorial, Barcelona, 2008.
29. Sennett, R., *The Craftsman*, Yale University Press, New Haven, 2008.
30. Terzidis, K., *Expressive Form: A conceptual approach to computational design*, Spoon Press, New York, 2003.
31. Descartes, R., *Discours de la méthode pour bien conduire sa raison et chercher la verité dans les sciences,* Librairie Philosophique J. Vrin, Paris, 2005.
32. Saunder, A., Baroque Parameters, in Puglisi, L. P., ed., *Theoretical Meltdown*, Wiley & Sons, Oxford, 2009.
33. Cache, B., *Objectile: Fast Wood: A Brouillon Project*, Springer, New York, 2007.
34. Kline, M., *Mathematics in Western Culture*, Oxford University Press, Oxford, 1953.
35. Curtis, W. J., *Modern Architecture since 1900*, 3rd edn., Phaidon, London, 1996.
36. Kolarevic, B. and Malkawi, A. M., eds., *Performative Architecture: Beyond Instrumentality*, Routledge, New York, 2005.
37. Portoghesi, P., *Nature and Architecture*, Skira Editore, Milano, 2000.
38. Picon, A., Digital architecture and the poetics of computation, *Metamorph: Focus*, la Biennale di Venezia, Venice, 2004, 58-69.
39. Scriba, C.J. and Schreiber, P., *5000 Jahre Geometrie*, Springer, Wien, 2002.
40. Lawvere, F. W. and Schanuel, S. H., *Conceptual Mathematics*, Cambridge University Press, Cambridge, 1997.
41. Kotnik, T., Das Experiment als Entwurfsmethodik: Zur Möglichkeit der Integration naturwissenschaftlichen Arbeitens in der Architektur, in Moravansky, A. and Kirchengast, A., eds., *Experiments in Architecture*, Jowis, Berlin, 2010.
42. Silver, M., *Programming Cultures*, Wiley & Sons, Oxford, 2006.
43. Sakamoto, T., *From Control to Design: Parametric/Algorithmic Architecture*, Actar, Barcelona, 2008.
44. Picon, A. and Ponte, A., eds., *Architecture and the Sciences: Exchanging Metaphors*, Princeton Architectural Press, New York, 2003.
45. Eisenman, P., *The Formal Basis of Modern Architecture*, Lars Muller Publishers, Baden, 2006.

Toni Kotnik
Swiss Federal Institute of Technology
(ETH) Zurich
Faculty of Architecture
Wolfgang-Pauli-Str.15
8093 Zurich
Switzerland

kotnik@arch.ethz.ch