BACK TO BLACK BOXES?

An Urgent Call for Discussing the Impacts of the Emergent AI-Driven Tools in the Architecture Design Education

VICTOR CALIXTO[1] and JULIANA CROFFI[2]
[1] Monash University, [2] The University of Adelaide
[1]victor.calixto@monash.edu, 0000-0003-2793-1916
[2]juliana.crofficalixto@adelaide.edu.au, 0000-0002-5698-9407

Abstract. In recent years, the advances in data science and Artificial Intelligence (AI) are disrupting all sectors, impacting the industry and academic fields. In the AEC sector, there have been a rising number of user-friendly "computational design services" generative and parameterised solutions driven by AI engines. However, if in one hand these services provide rapid solutions with minimal cognitive load, on the other hand, they obscure logical processes from computational design thinking, transforming them into black boxes and limiting the designer on making use of technology to create novel solutions. To overcome these challenges, the teaching of computational design thinking should be integrated in architecture education on undergraduate and master programs. This study conducts a critical literature review and proposes a framework to be implemented in architecture education, discussing the complexity involved in the learning process. The framework provides a layered approach that unfold the levels of abstraction of the nested black boxes of computational design and AI in an educational context.

Keywords. Computational Design Thinking, Architecture Education, Black Box, AI.

1. Introduction and Problem Statement

In recent years, data science and Artificial Intelligence (AI) have made significant strides, causing substantial disruptions across various industries. In the domain of architectural design, there has been a noticeable increase in the utilisation of third-party AI services for generating design proposals, with a growing number of market-driven encapsulation of "computational design services", offering user-friendly generative and parameterised solutions driven by AI engines. These services provide rapid design solutions with minimal cognitive load, obscuring the underlying logical processes, potentially curtailing designers' freedom, flexibility, and deeper understanding regarding the design proposition, as the paradigm of black boxes (Flusser, 1985).

Moreover, the role of designers concerning computational tools has been a subject of extensive debate since the inception of this field in the 1970s (Mitchell 1975; Terzidis, 2006; Oxman, 2006; Burry, 2011; Carpo, 2017), up until the concept of computational designers emerged as digital toolmakers (Fischer, 2003; Burry, 2011). Nonetheless, architectural education continues to face challenges in accommodating the educational needs related to these debated topics (Oxman, 2008; Leitão et al., 2012; Celani & Vaz, 2012; Akbar et al., 2023). The integration of AI in the design process adds a further layer of complexity to this issue. Consequently, there is an urgent call to initiate discussions regarding the role of architectural education in addressing this demand.

2. Theoretical Background

During the early 1960s, the advent of the design methods movement initiated scholarly discussions aimed at fostering a more systematic comprehension of architectural design processes. The primary objective was to externalise these processes, aiming for greater complexity and the utilisation of computers to automate repetitive aspects of the design process (Celani & Vaz, 2012). Fast forward six decades, with an excess of fifty million hundred thousand transistors available as computational power (Moore, 1972), the challenges associated with teaching computational design thinking persist. Moreover, new strata of complexity have begun to overlay and obscure the current landscape, driven by the emergence of data science and AI-powered tools that increasingly dominate the design process. These advancements obscure the underlying design steps by encapsulating them within nested black boxes (Glanville, 1982).

The notion of a black box, crucial for understanding the current state of nested black boxes within computational design thinking education, warrants delineation. Subsequently, this discussion is followed by an examination of computational tools, practices, and process essential to the realm of computational design thinking. Finally, an overview is provided concerning the rise of data science and AI tools within the Architecture, Engineering, and Construction (AEC) domain. This approach aims to elucidate the intricate interplay and encasement of design processes within nested black boxes, reflecting the evolving scenario of computational design thinking education.

2.1. BLACK BOXES

The inception of the black box concept finds its origins in the realm of cybernetics during the mid-1950s, notably with Ashby's seminal work, "Introduction to Cybernetics" (Ashby, 1956). Ashby elucidates this concept by employing the canonical example of a child learning to manipulate a door handle to open a door, oblivious to the internal workings connecting their action (input) to the latch's movement (output). This example serves as a metaphor, illustrating that everything we perceive functions as a black box, and our role as observers involves interacting with these black boxes to formulate functional descriptions contained within them.

Glanville (1985) subsequently expounded upon the process of "whiting" a black box, delineating it as the construction of a perceptible relationship between the input and output of a black box by an external observer. This observer maps the underlying structure, progressively enhancing their confidence in this description through iterative feedback loops between themselves and the black box. Glanville also notes that to a second external observer interacting solely with the inputs and outputs of the initial system, comprised of the first observer and black box, this system also appears as a black box. This nested progression of black boxes and observers forms the bedrock of second-order cybernetics.

Flusser (1985) also conceptualise black boxes through the description of an interaction between a photographer and a camera. Here, he defines the camera as an apparatus, derived from the Latin verb "apparare", signifying readiness or preparation, internally manipulating number-like symbols. The photographer, while unaware of the internal workings, manipulates the inputs and outputs of the camera to achieve the desired photographic outcome. Flusser suggests that as apparatuses proficiently control numerical symbols, mechanising the thinking, the human beings will become less competent to deal with it and will tend to rely on more apparatuses (Flusser, 1985).

In a parallel vein, Bruno Latour defines black boxes as encapsulating scientific and technical processes, practices, and objects necessary for their operational success (Latour, 1987). Latour posits that when a sealed machine operates seamlessly, its users focus solely on its inputs and outputs, disregarding its intricate internal mechanisms. Consequently, as advancements in science and technology progress, the internal logic of these systems becomes increasingly opaque (Latour, 1987).

Within the architectural design domain, black boxes have become entrenched in the discourse of computational design, encompassing both the practices and processes of computational design thinking and the objects represented by computational tools.

This interpretation of the black box concept, forged from the interaction between apparatuses and humans, finds resonance in the perspective of designers engaging with computational design tools.

2.2. COMPUTATIONAL DESIGN THINKING AND EDUCATION

The discussion around architectural education integrating new computational technologies in the design process gained momentum during 80', 90', and early 2000', when computers started to become more accessible in most architecture schools (Oxman, 2006; Celani & Vaz 2012).

To establish the critical relation of computational design thinking and black box systems, this paper categorises computational design thinking as computational tools, practices, and processes following based on the Latour's definition of black box (Latour, 1987).

2.2.1. Computational design thinking as computational tools

The emergence of visual programming languages, such as Generative Components and Grasshopper in the mid-of 2000' represented a rapid expanding of interest in computational thinking in architecture education, since visual programming languages has a shallow learning curve compared to textual scripting languages (Celani & Vaz 2012), democratising computational thinking without the need of coding in textual programming languages (Akbar et al., 2023). However, even though visual programming languages represents an ideal approach to reduce cognitive load and smoothing the slope of cognitive barriers (Aish & Hanna, 2017), it also represents challenges related to black box encapsulations of functions and algorithms, as a shortcut that obscures computational design thinking steps in favour of the promptness of results, which can mask the structure of foundational concepts of generative design (Fischer & Herr, 2001), steps of mathematical thinking (Burry & Burry 2010), and canonical structures of computational thinking as recursion and loops.

The concepts of computational design or the medium in where we operate through computational design thinking can be represented in three models of generative systems, iconic, analogue, and symbolic (Mitchell, 1975) at different levels of abstraction (Celani & Vaz 2012). Iconic models visually represent architectural forms and can be represented as direct parametric relations in 3D software, while analogue models use analogous properties and can be represented as visual programming languages, and symbolic models employ symbolic operations and can be represented through textual programming languages (Celani & Vaz, 2012). Novice architecture students may achieve superior results using visual programming languages. Nevertheless, the application of these languages is constrained to parametric exploration in the absence of a foundation in textual programming. In contrast, scripting languages offer a broader scope, allowing for the development of sophisticated generative design methodologies, including the implementation of recursive rule application. Moreover, these scripting languages can be seamlessly integrated with visual programming elements, thereby enhancing interactivity, and enabling real-time outcomes (Celani & Vaz, 2012).

If in one hand the encapsulation of functions and algorithms obscures computational design thinking, on the other hand the abstractions of the medium utilised can create abstraction barriers, acting as another layer in the black box context.

Abstraction barriers is a concept defined as the minimal set of novel abstractions that must be comprehended before utilising a system. (Green & Blackwell 1998). According to Aish (2017), an abstraction evolves into a barrier when users are compelled to grasp it before recognising its utility or relevance to their needs. In the context of computational design tools, the choice of the platform utilised to develop parametric systems can represent different abstraction levels, influencing in its learning curve (Aish & Hanna, 2017).

Besides the medium of communication in which the computational design thinking works, there are concerns platform dependency as part of the design process, obliging learners to follow software-dependent workflows (Akbar et al., 2023), which tend to be proprietary pieces of software that does not provide the freedom for the user to run for any purpose, study and adapt for his own needs, redistribute, and share the improvements with a broader community (Stallman, 2002). The software-dependent workflows based on proprietary pieces of software restricts the access of the functions and algorithms that compound the parametric tools, consequently, represents an additional layer of the black box system that obscures computational design thinking as computational tools.

2.2.2. Computational design thinking as practices

Negroponte (1975) advocated for the importance of fostering direct engagement between designers and computers in the design process, emphasising the significance of a collaborative human-machine interface. He delineated this approach (computation) as divergent from "computerisation", characterising it as a procedural system centred on batch input-output computation for data processing (Negroponte, 1975). Later, Terdizis (2006) further developed the concepts of computerisation and computation defining its boundaries, being the first a design practice that typically uses computers to design as a literal translation of the paper-based design process to the computer screen, and the second the partnership between designers and computers in exploring computer power to extend designers capabilities and creativity. In this context, computing is not just seen as a tool for representing or creating machines, but rather as a platform for thinking and designing (Carpo, 2018).

To explore this partnership with the computer, the practices of computational design thinking have been explored by a set of computational approaches, such as, parametric design, generative design, and algorithm design. Caetano et. al (2019) extensively discussed these computational design thinking approaches proposing a taxonomy that defines parametric design as a design approach based on the use of parameters to describe sets of designs; generative design, as a design approach that uses algorithms to generate designs; and algorithm design, as a subset of generative design that has an identifiable correlation between algorithms and its outcomes (Caetano et. al, 2019).

Despite the increasing adoption of computational design in schools worldwide, concerns arise regarding its integration into the curriculum. Veloso and Krishnamurti (2019) discuss the rise of scripting and visual programming languages in design systems, which externalises design instructions and shifts designers away from traditional architectural drawing practices, fracturing the black box. However, the prevalence of aesthetic disputes and non-standard forms in digital design replaces the subjective of architectural drawings with digital variations, keeping the culture of studio design critic that traditionally subjectively measure design success, exacerbating the opacity of the design process. Another concern is grounded in the observation that many universities are focusing on teaching digital tools and plugins to rapid enable students in the proficiency of "operating" a set of architecture design tools in simple design tasks of limited architectural complexity, without the high-order thinking (Schneider, 2001), showing that encapsulation of ready-to-use computational tools into black boxes can drive to design process to impoverishing of computational design thinking if a solid foundation on computational thinking is not provided (Gaudillière, 2020). Following a different perspective, Gardner et al. (2020) argue that the philosophical tradition of pragmatism provides the for the perspective computational design thinking that "emergent and evolutionary behaviour is acceptable; that the black box process can be trusted as long as the practical goals are realised". These different valid positions shows that still no consensus on the degree to which the designer student should have regarding the internal logic of the process of the black boxes of computational design thinking in their practice, or in other words, what tonality of "grey" these boxes should be.

2.2.3. Computational design thinking as processes

Mitchell (1975) recognises design problem as a special kind of problem-solving process that involves "wicked problems" (Cross et al. 1984), due to some design variables being inversely proportional; for example, the control of illumination against the control of radiation in a room. “Wicked problems” are complex and multifaceted issues that are difficult to understand or solve due to their many interconnected components and lack of clear definitions (Rittel & Webber 1973), this category of problem present a particular challenge due to the creative and iterative nature of the design process.

Kelly and Gero (2021) argue that complex problems necessitate both design and computational thinking, as they offer complementary approaches. They suggest teaching these methods together, proposing an ontology that integrates them. In computational design thinking, designers must transition from a subjective understanding of the problem to an analytical view. Oxman (2006) introduces a schema outlining interactions between designers and computers in digital design, categorising various models. Additionally, Oxman (2017) proposes the Parametric Design Thinking (PDT) framework, arguing that scripting provides a new way of design thinking, being a fundamental component in of knowing in models during design process, and the reflection of the designers relates to their ability to understand and control the computational and scripting tools. The act of scripting gives the freedom for the designer to customise and reconfigure software behaviours, fitting to their way of thinking and working (Burry, 2011). Therefore, scripting becomes one of the ways that computational designers can interact with one of the layers of the black box system, whitening the process of computational design thinking, and becomes a critical skill in an educational context.

2.3. EMERGENCE OF DATA SCIENCE AND AI TOOLS IN ARCHITECTURE

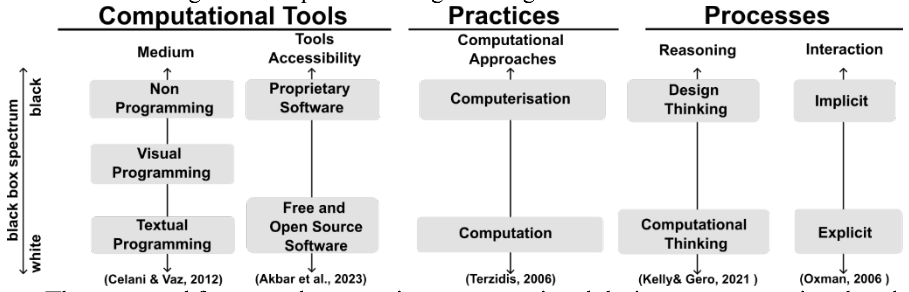
Recent advancements in Data Science and Artificial Intelligence (AI) are revolutionising various industries, fostering improved interactions between humans and machines. In Architecture, Engineering, and Construction (AEC), these disruptions are evident in research and industry, with growing interest in computational design methods and AI-driven design proposals. This shift signifies a move towards a second digital era, characterised by big data and increasing of computational power for form generation based on search and simulation methods (Carpo, 2017).

Computational skills, once cutting-edge, are now mainstream (Basarir, 2022). AI advancements, like language models and Generative Adversarial Networks (GANs), are pushing design boundaries. Techniques like semantic segmentation and Graph Machine Learning further explore AI's potential in architecture and design (del Campo & Leach, 2022; Ma et al., 2021; Alymani et al., 2017). However, if the scenario is accelerated for both industry and research, the architecture educational system does not overcome the challenges regarding a seamlessly integration of computational design thinking in the curriculum (Kelly & Gero, 2021), and the emergence of the new AI and Data Science methods expands this gap, requiring deeper layers of computational design thinking to be used as integral part of the design process, and not only as a ready-to-use true black AI services black-boxes. Inevitably, the new generation of architecture students are already making use ready-to-use AI services to assist their assignments, generating renders through MidJourney and Stability.ai, or architectural memorials using ChatGPT, as mediums of "computerisation", but the challenges on the "computation" side pushes the question on how the current architecture education will tackle the challenge of integrating computational design thinking in the architecture design curriculum, including the new advancements in the field of AI and Data science remains to be addressed.

3. An educational framework for computational design thinking black boxes.

To address the critical gap of the multifaceted impact of emergent Data Science and AI technologies on computational design education and investigating the role of architecture education in preparing future computational designers in consideration to emerging AI tools, this study, first, maps practices, processes, and tools according to its level of obscurity and encapsulation of complexity as a black box system as presented in (Figure 1).

Figure 1. Computational design thinking black boxes framework



The proposed framework categorises computational design as computational tools, practices, and processes, using the Latour's definition of black boxes.

3.1. COMPUTATIONAL TOOLS

Computational tools are categorised based on medium and tools accessibility. The medium describes how the tool operates, ranging from black-box obscurity in non-programming, visual programming, and textual programming, with increasing levels of abstraction while concealing internal logic. Tools accessibility varies from closed-source "click-the-button" proprietary tools to open-source solutions allowing investigation and modification of source code. Grey solutions in the middle ground combine proprietary software with open-source plugins, exemplified by Rhino 3D-Grasshopper (Proprietary) and Ladybug Tools plugins (Open Source).

3.2. PRACTICES

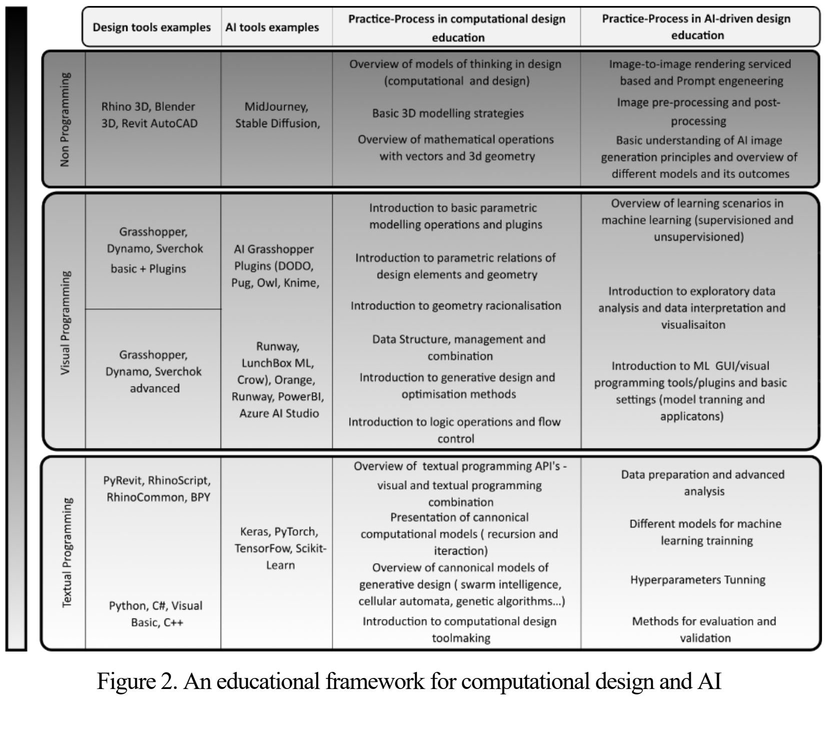
Practices of computational design thinking encompass various computational approaches, ranging from basic computerisation that merely replicates or accelerates traditional tasks without improving design processes, to computational practices where creativity and innovation are actively fostered through a collaborative human-machine interaction through computational methods, enhancing the design process.

3.3. PROCESSES

Processes are categorised into reasoning and interaction. The reasoning spectrum illustrates how the problem-solving process is approached, ranging from subjective understanding in design thinking to analytical interpretation in computational thinking. Interaction delineates how designers communicate with the design, ranging from implicit processes driven by subjectivity to explicit, systematic decision-making and communication.

3.4. AN LAYERED APPROACH FOR COMPUTATIONAL DESIGN/AI EDUCATION

The framework proposes a layered approach to education for architecture students, delineated in Figure 2, targeting the intricate layers of computational design thinking black boxes. This approach is structured around the medium, representing the interface for students' design tasks and enabling practices and processes. The framework suggests using the exemplified computational design and AI tools to introduce computational design thinking in education, gradually delving into black box abstraction levels. This incremental approach aims to avoid creating abstraction barriers and gradually build foundational practices and processes. Each tool, practice, and process overlap with the previous step, facilitating the learning process. By teaching computational design thinking and AI-driven education concurrently at similar abstraction levels, the curriculum enhances both contexts, providing a robust foundation for students.


Black Box Abstraction Levels

	Design tools examples	AI tools examples	Practice-Process in computational design education	Practice-Process in AI-driven design education
Non Programming	Rhino 3D, Blender 3D, Revit AutoCAD	MidJourney, Stable Diffusion,	Overview of models of thinking in design (computational and design) Basic 3D modelling strategies Overview of mathematical operations with vectors and 3d geometry	Image-to-image rendering serviced based and Prompt engeneering Image pre-processing and post-processing Basic understanding of AI image generation principles and overview of different models and its outcomes
Visual Programming	Grasshopper, Dynamo, Sverchok basic + Plugins Grasshopper, Dynamo, Sverchok advanced	AI Grasshopper Plugins (DODO, Pug, Owl, Knime, Runway, LunchBox ML, Crow), Orange, Runway, PowerBI, Azure AI Studio	Introduction to basic parametric modelling operations and plugins Introduction to parametric relations of design elements and geometry Introduction to geometry racionalisation Data Structure, management and combination Introduction to generative design and optimisation methods Introduction to logic operations and flow control	Overview of learning scenarios in machine learning (supervisioned and unsupervisioned) Introduction to exploratory data analysis and data interpretation and visualisaiton Introduction to ML GUI/visual programming tools/plugins and basic settings (model tranning and applicatons)
Textual Programming	PyRevit, RhinoScript, RhinoCommon, BPY Python, C#, Visual Basic, C++	Keras, PyTorch, TensorFow, Scikit-Learn	Overview of textual programming API's - visual and textual programming combination Presentation of cannonical computational models (recursion and iteraction) Overview of cannonical models of generative design (swarm intelligence, cellular automata, genetic algorithms...) Introduction to computational design toolmaking	Data preparation and advanced analysis Different models for machine learning trainning Hyperparameters Tunning Methods for evaluation and validation

Figure 2. An educational framework for computational design and AI

4. Discussion and Conclusion

This study proposes a layered framework for education of undergraduate and master architecture programs. Deepening into the layers gradually allow students to absorb and process the information in a more seamlessly manner, building a solid foundation towards the next layers of complexity. It can also keep the students engaged in learning and deepening their level of understanding towards more complex layers. If high complex subjects are introduced too early during the learning process, it can make the students to lose interest and withdraw.

Deepening the understanding of complex black-box layers without a grasp of foundational concepts built upon earlier steps can introduce abstraction barriers, impeding students' progress. Hence, future studies might benefit from concentrating on achieving a balanced approach between black and white boxes, facilitating a smoother learning process through real-world case studies in educational scenarios.

In an optimal scenario, students should receive education aimed at attaining at least the initial level of proficiency in visual programming layer. This would provide them with a basic understanding of computational design thinking, which encompasses computational design and AI, providing them the foundational skill to investigate the subject independently further. However, in an ideal scenario, students would strive to achieve mastery in the highest level of visual programming, thus establishing a solid foundation to effectively control computational design and AI tools. In this context, students would possess the necessary knowledge to utilise these tools actively in generating solutions, rather than merely relying on pre-packaged third-party tools. By establishing these foundational skills, students can progress towards tackling more advanced design challenges, utilising their acquired knowledge for research, tooling, or software development.

However, once they achieve deeper levels of black box abstractions, they may become contributor to develop encapsulated tools for novices, restarting the cycles, and pushing it back to the black box.

References

Aish, R., & Hanna, S. (2017). Comparative evaluation of parametric design systems for teaching design computation. Design Studies, 52, 144-172.

Ashby, W. R. (1956). An introduction to cybernetics.

Alymani, A., Jabi, W., & Corcoran, P. (2023). Graph machine learning classification using architectural 3D topological models. Simulation, 99(11), 1117-1131.

Burry, M. (2011). Scripting cultures: Architectural design and programming. John Wiley & Sons.

Basarir, L. (2022). Modelling AI in Architectural Education. Gazi University Journal of Science, 35(4), 1260-1278.

Burry, J., & Burry, M. (2010). The new mathematics of architecture. Thames and Hudson.

Caetano, I., Santos, L., & Leitão, A. (2020). Computational design in architecture: Defining parametric, generative, and algorithmic design. Frontiers of Architectural Research, 9(2), 287-300.

del Campo, M., & Leach, N. (Eds.). (2022). Machine Hallucinations: Architecture and Artificial Intelligence. John Wiley & Sons.

Carpo, M. (2017). The second digital turn: design beyond intelligence. MIT press.

Celani, G., & Vaz, C. E. V. (2012). CAD scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view. International Journal of Architectural Computing, 10(1), 121-137.

Cross, N. (1984). Developments in design methodology. (No Title).

Fischer, T., Fischer, T., & Universtität, G. A. (2003). Toolmaking for digital morphogenesis. International Journal of Design Computing, 6, 35-40.

Flusser, V. (2013). Towards a philosophy of photography. Reaktion Books.

Gardner N; Meng LL; Haeusler MH, 2020, 'Computational Pragmatism', in RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th International Conference on Computer-Aided Architectural Design Research in Asia, CAADRIA 2020, Bangkok, pp. 489 - 498, presented at CAADRIA 2020, Bangkok, 05 August 2020 - 07 August 2020

Gaudillière, N. (2020). Evolutionary Tools and the Practice of Architecture: from Appropriated Typology to Becoming the Black Boxes of CAAD.

Glanville, R. (1982). Inside every white box there are two black boxes trying to get out. Behavioral Science, 27(1), 1-11.

Glanville, R. (2013). Cybernetics: thinking through the technology. Traditions of systems theory: Major figures and contemporary developments. Routledge, New York, 45-77.

Green, T., & Blackwell, A. (1998, October). Cognitive dimensions of information artefacts: a tutorial. In Bcs hci conference (Vol. 98, pp. 1-75).

Kelly, N., & Gero, J. S. (2021). Design thinking and computational thinking: A dual process model for addressing design problems. Design Science, 7, e8.

Khean, N., Fabbri A., and Haeusler H. (2018) 'Learning Machine Learning as an Architect, How to? - Presenting and Evaluating a Grasshopper Based Platform to Teach Architecture Students Machine Learning', 95–102. Łódź, Poland. https://doi.org/10.52842/conf.ecaade.2018.1.095.

Latour, B. (1987). Science in action: How to follow scientists and engineers through society. Harvard university press.

Leitão, A., Santos, L., & Lopes, J. (2012). Programming languages for generative design: a comparative study. International Journal of Architectural Computing, 10(1), 139-162.

Negroponte, N. (1975). The architecture machine. Computer-Aided Design, 7(3), 190-195.

Ma, X., Ma, C., Wu, C., Xi, Y., Yang, R., Peng, N., ... & Ren, F. (2021). Measuring human perceptions of streetscapes to better inform urban renewal: A perspective of scene semantic parsing. Cities, 110, 103086.

Mitchell, W. J. (1975). The theoretical foundation of computer-aided architectural design. Environment and planning b: planning and design, 2(2), 127-150.

Moore's law: The number of transistors per microprocessor. (n.d.). Our World in Data. Retrieved December 17, 2023, from https://ourworldindata.org/grapher/transistors-per-microprocessor?time=1972..latest

Oxman, R. (2006). Theory and design in the first digital age. Design studies, 27(3), 229-265.

Oxman, R. (2008). Digital architecture as a challenge for design pedagogy: theory, knowledge, models and medium. Design studies, 29(2), 99-120.

Oxman, R. (2017). Thinking difference: Theories and models of parametric design thinking. Design studies, 52, 4-39.

Rittel, H. W., & Webber, M. M. (1973). Dilemmas in a general theory of planning. Policy sciences, 4(2), 155-169.

Simon, H. A. (1969). The sciences of the artificial. MIT press.

Stallman, R. (2002). Free software, free society: Selected essays of Richard M. Stallman. Lulu. com.

Terzidis, K. (2006). Algorithmic architecture. Routledge.

Veloso, P., & Krishnamurti, D. R. (2019). From Black Box to Generative System. In 107th ACSA Annual Meeting Proceedings, Black Box (pp. 525-533).