Coding as Creative Practice

Corneel Cannaerts¹
¹KU Leuven Faculty of Architecture
¹corneel.cannaerts@kuleuven.be

This paper looks into coding as a creative practice within architecture, more specifically into textual and graphical coding as a practitioner during the design process. It argues that coding is not a mere tool for designing but a particular design medium, with its own affordances and resistances. Using code as a design medium provides a specific form of feedback, it influences the design process and its outcomes. Code is a technological and conceptual support for design thinking. In other words, code and coding can be ascribed agency in architectural design. This research is based on a number of cases from design practice and teaching, ranging from small design experiments, developing software tools for specific design projects and teaching workshops. The cases are grouped into three metaphors, each describing a particular aspect of coding as a design medium.

Keywords: coding, sketching, tooling, structuring

INTRODUCTION

In the last decade coding has become increasingly accessible as a design medium in architecture and design. Design software is more and more extended and customised through the development of scripting interfaces, add-ons, plugins, libraries, integrated development environments and programming languages (Davis and Peeters 2013). Designing through writing custom code has shifted from being practiced by a handful of academics and pioneers to a broad emerging culture of coding in architecture and design (Burry 2011). While there have been an extensive number of exhibitions, conferences and publications documenting this shift, they tend to focus either on code as technique or on the design outcomes, only a limited amount of publications address how code as a design medium informs this culture of coding (Maeda 2004; Silver 2006; Coates 2010; Reas and McWilliams 2010; Burry 2011).

Recently there has been a change in architectural research both in academia and practice, often described as research by design and creative practice research (Fraser 2013; Verbeke 2013; Verbeke et al. 2015). In this mode of research, which combines action and reflection, practice and theory, the act of designing is a substantial and crucial driver for research. The research discussed in this paper uses this mode of inquiry, which can be described as the research into the 'medium itself' (Van Schaik and Johnson 2011; Blythe and van Schaik 2013), as it looks into the role of coding as a design medium in creative practices of architecture and design. Coding is approached not with the aim towards the fundamental understanding of a computer scientist or the professional understanding of a software developer, but rather with the aim of exploring its potential for design practice.

This paper is framed in the two evolutions outlined above and explores the impact of the increased accessibility of coding on practices of architecture
and design. The aim is to demonstrate how coding as a medium informs a design process. The role of coding is explored not isolation but in relationship to other media used in a design process. This paper is based on a number of cases from design practice and teaching, ranging from small design experiments, developing software tools for a specific design project and teaching workshops. Through reflection on the projects the cases are grouped into three metaphors, each describing a particular aspect of coding as a design medium.

**SKETCHING WITH CODE**

Since the adoption of computers in architectural practice there has been a debate on whether CAAD can be used as a sketching medium. The long held belief (Schön 1983; Garner 1992; Goel 1995; Bilda and Demirkan 2003) that design ideation was only possible through hand drawing and CAAD was limited to refining and documenting already existing design ideas is challenged by more recent research (Jonson 2005). Schaeverbeke and Heyligen (2012) argue that both analogue and digital media can used for sketching. Furthermore, contemporary practice architects tend to switch between or use different design media simultaneously, design ideation can be located in this 'in-between' (Schaeverbeke and Heyligen 2012). The appropriation of coding by architects it cannot be seen in isolation but as an extra layer of this fluid and diverse use of media in design practice (Meredith and Sample 2013).

Gänshirt (2007, p.101) makes a distinction between verbal and visual design tools, corresponding with complementary ways of thinking: verbal, linear, logical thinking on the one hand, visual-spatial, associative thinking on the other. Based on McLuhan's (1964, p.22) thesis that media are ordered hierarchically in terms of abstraction, coding can be seen as an abstract verbal design medium, whereas sketching is classified as an intuitive visual medium. Interestingly enough Gänshirt (2007, p.101) defines the computer as being on the intersection between visual and verbal tools. Even though writing code is a verbal design tool the feedback it provides to the designer is partly visual, and if the designer is fluent enough in writing code it can be used as a sketching tool.

This notion of sketching, bridging the gap between verbal and visual and exploring design ideas through writing code is central to Processing, an open-source programming language, development environment and community of creative coders (Maeda 2004; Reas and McWilliams 2010; Reas and Fry 2007; Processing, 2016). Programs written in Processing are called a sketches, and they are collected in a sketch book. Its syntax refers to terms used in design practice and software: drawing, stroke, fill, etc. Its simple interface, reference and syntax is deliberately designed to reducing the gap between writing code and getting visual feedback on screen. Its core functionality can be extended, with third party libraries developed by the community, into a means for prototyping and production.

In my design practice writing sketching through code plays an important role, I have used Processing extensively ranging from quick sketches, design experimentation, coding tools for other architects, designers and artist, writing code for specific fabrication machines (Cannaerts 2015a; Cannaerts 2015b). The sketches shown in Figure 1, can be compared to doodles, or explorations of specific generative systems. Moments of playing, interacting with the graphical representation of code and tweaking values are alter-
nated with changes to the code itself. Rather than describing the specific details of these projects, in this paper I will reflect on the experience of sketching with code and how this informs my design practice.

Working with code as a design medium provides the designer with different kinds of feedback on the screen: a graphical window showing the result of the running code, a textual one showing the actual code itself and possibly textual feedback through the console. Although the running code can be made to respond to various inputs, for example mouse and keyboard, the design mainly progresses by working on the code itself. Text-based coding is an unforgiving medium - forgetting even one character will lead to a syntax error, and it is often hard to tell from the visual feedback alone what is exactly going on in an algorithm. These limitations can be overcome by continuously testing the code, incrementally building on working versions of the code and using the console to provide textual feedback, or by developing a debug mode that renders certain information on the screen. Graphic coding interfaces tend to be a bit more forgiving, as code is contained within blocks with clearly defined inputs and outputs, but they tend to become quite hard to read once definitions get larger.

As stated above, sketching with code adds another layer to the multiple design media used in practice, and does not replace sketching with pen and paper but rather complements it. While coding sketching on paper can greatly help with visualising ideas while simultaneously testing them out in code. Different from sketching with pen and paper, sketches with code develop incrementally, not by retracing a sketch but by building on working previous blocks of code. The reuse of code and gradual increase in complexity allows sketches to be turned into blue prints and actual design projects.

Code as a medium to develop ideas tends to progress in chunks as parts of the underlying algorithm get defined, evaluated and refined. While developing an algorithm, often happens through a simplified version of the design problem at hand, which can take the form of simpler input geometry or low number iterations or variables. Once an algorithm reaches a certain state of development, I tend to increase the complexity, which is often a revealing moment. In graphical programming, a similarly layered feedback exists, allowing work on the geometrical and the algorithmic simultaneously.

MAKING AND USING TOOLS

Using code as a design medium affords control over the algorithms beneath the interface of software tools and, as such, allows designers to develop their own design tools. Going beyond the intended use of a tool or developing your own tools is frequently mentioned by practitioners as a main motivation for using code as part of their creative practice (Maeda 2004). In architecture, this is a prominent argument in many publications on parametric and algorithmic design (Kilian 2006; Meredith 2008; Fischer 2008; Burry 2011). This position is most explicitly stated by Aranda and Lash (2006) under the title tooling, they describe a number of algorithmic techniques which are illustrated by a version of the algorithm in pseudo code, a number of experiments and a project developed with this specific algorithm.

In 2013 I gave a workshop together with Phil Ayres and Hollie Gibbons at CITA, Kopenhagen, Denmark to start the Adaptive Aggregates installation project as the start of the CITA studio master program. The workshop conceived as an introduction to computational design that required no prior knowledge of coding. Three design tools were developed looking into simulating aggregate structures across a number of scales. The design tools used text-based coding in Processing and parametric modelling in Grasshopper, while they could be used without altering the actual code through user interfaces, they introduced various computational design strategies and provided the source code for students willing to learn how the underlying code worked.
The first tool (Figure 2) looked into the fabrication and inflation of the material component, it was developed through iterative material testing and simulated using a particle spring model in Grasshopper and Kangaroo. The components, consisted out of an inflated tube were fabricated prior to the workshop in three scales. The second tool (Figure 3) was coded in Processing and explored different ways of connecting the components, it allowed for exploring branching structures based on the logic of connecting the different components. The user interface allowed for control of the geometry of the components, the rules of connection, the noise introduced by small deviations in the connecting system. The third tool (Figure 4) used a rigid body simulation to simulate the pouring or packing aggregations, the user interface allowed for the position, type and number of components to be poured into the scene, an environment mesh could be used to simulate pouring over different objects and obstacles.

In order to overcome the limits inherent in design software, we used different computational design tools and strategies to develop a spatial proposition. We chose to operate simultaneously in different software environments and use different computational design techniques: explicit geometric modelling in Rhino, visual parametric modelling in Grasshopper and text-based scripting in Processing. The preparation resulted in three design tools; computational techniques were made available through design tools with graphical interfaces. The three introduced tools are described in more detail above. The position, orientation and geometry of the components could be described by the three coordinates and an offset thickness, and all the tools could export and import text files containing this information. Rhino was central in the work-flow; it was used to collect and compile the multiple explorations, prepare the starting conditions and environment meshes as input for the simulations and assemble the different strategies into a final design proposal.

In the tools coded for Adaptive Aggregates workshop, simulations could be found on multiple scales. Through nesting different simulations, it becomes clear that each tool comes with its own assumptions and its own requirements and limitations and all provide a different insight in the design task at hand. All of these simulations have their affordances and capture materiality in discrete encoded elements, from the particles and springs simulating the inflation to the compound shapes assembled out of convex parts in the rigid body simulation. Furthermore, time is encoded as explicit discrete steps: the algorithm that makes up the simulation computes the resulting world one iteration at a time.

Having access to the code that drives software tools can allow for a deeper understanding of the design issues at hand and uncover the assumptions inherent in the tools. Actively developing this code
allows for these assumptions to be questioned and explored differently. In the Adaptive Aggregations workshop I was often reminded of my own assumptions as students were using the tools. To that extent, I think the tooling metaphor is useful, but it also introduces an opposition between tool making and tool using, which in my practice of using code as a design medium are not separate activities, rather they mutually inform each other. The process of making a tool gradually unravels the design problem at hand. Furthermore, coding your own design tools does not generally start from a blank canvas but is instead based on examples, code snippets, add-ons and libraries often developed by others. The environments in which you code are obviously tools themselves, with their own assumptions, limits and potential, programmed by someone else.

STRUCTURING CODE

One of the main reasons architects and designers name for appropriating coding in their creative practice is automating repetitive work (Burry, 2011). Reas and McWilliams (2010), call repetition the 'computers talent' and demonstrates this with a number of examples from art, design and architecture. On could say that the effort and time spent on coding only becomes useful if some degree of repetition and variation is involved, and it can operate on collections, lists or arrays of elements. Code has a bias towards the many and the multiple, and designing with code tends to shift the focus from the one off and the unique to the systemic and the multiple (Burry 2011).

In the Material & Digital Form Finding workshop (Figure 5), material experimentation formed the basis for coding different a number of bespoke design tools that allowed for exploring spatial potential of form finding systems. Conceptually translating material system into a code requires decisions into how to structure the code, ie, what are the material components and how do they relate computational concepts. In this project they were simulated as particle spring-models (Kilian 2006), where springs and particles are modelled as autonomous objects negotiating their environments, the forces at play, and other components.

Much of the syntax of code has to do with structuring repetition and meaningful variation. In text-based coding this is reflected in defining and declaring variables, loops, conditionals, functions, objects, classes etc., which are all means of efficiently structuring code and determine the flow of instructions passed to the computer (Fry 2010). The elements for structuring code are highly hierarchical and are geared toward modularity and reuse, and splitting up a design problem into reusable chunks.

Findings from the Material & Digital Form Finding workshop were further developed. The Folded Strips Pavilion project (Figure 6), which extended the relationship between material and digital to be more it-
erative and cyclical by also including digital fabrication. Since the project required digital fabrication the workflow was based on Rhino and Grasshopper as a more robust modelling environment, which was further extended with Kangaroo, a physics simulation add-on for Grasshopper and Anemone, which allows recursion within the Grasshopper environment. The Folded Strips Pavilion used a physics simulation to derive the overall form of a pavilion. The design simulated a particle-spring model based on hexagonal grid, the grid can be deformed to allow for denser or less dense areas. In order to be fabricated the data tree of the hexagonal grid is reorganised as continuous folded strips rather than hexagonal cells.

In visual programming languages, such as Grasshopper, code is structured as a network of components, where each component computes an output based on data it receives as an input. Different than in text-based programming, the flow of execution is explicitly visualised, which provides a clear feedback of the algorithm. When definitions become more complex, components can work on lists of data and nested lists of data, or data trees, it becomes increasingly harder to understand what is happening in the algorithm which necessitates clearly structuring the definitions through grouping, naming and clustering (Davis et al 2011).
CONCLUSION: AGENCY OF CODE
While the metaphors used in this paper to describe coding as a creative practice - sketching, tooling, structuring - each highlight a particular aspect of how coding operates as a design medium, none describes the use of coding in all its intricacy. Because of its reusability and fluidity, code as a design medium can switch between these different modes of informing the design process. The cases discussed in this paper demonstrate that code is a specific design medium with its own affordances and resistances. In other words code has agency in the design process and each of the discussed metaphors unveils part of that agency (Cannaerts 2015a; Pickering 1995).

The fundamental difference between coding and most other design media, is the layered-ness of the provided feedback. Coding allows for simultaneously working on the geometry representing a particular instance the design and the textual representation of the logic that drives that design. Through having sufficient experience in coding and using intuitive interfaces, coding can become a form of sketching, although it operates through both textual and geometric manipulations. Working on verbal and visual representations of design is also reflected in the difference between making and using tools. Instead of promoting the first over the latter it is through the switching between both that designing with code progresses. Both object oriented programming in Processing and data trees in Grasshopper are elegant structures, computationally but also in visualising the structure of the algorithm, this paper demonstrates that those structures can be a conceptual driver for design.

FURTHER RESEARCH
Code is a shared and collective medium, the work presented here could not be possible without the programmers and designers of the programming languages, interfaces and hardware. Having access to underlying algorithms in design is empowering, but is only pealing a way the first layer of interfaces of design tools. What is revealed is a highly designed and layered cultures of code, the agency of code lies obviously largely within this shared and cumulative history of code and the people who contributed to this.

The cases discussed in this paper approaches coding from the perspective of a designer writing code as part of the design process. The arguments presented here are mainly limited to a conversation between the author as designer and the design at hand. Coding increasingly influences shared practices and design collaboration, further research could be done in sharing code as collaborative practice. Code becomes increasingly a shared language between multiple partners in the design process.

ACKNOWLEDGEMENTS
The Folded Strips Pavilion project was developed by Corneel Cannaerts in collaboration with Bart Mermans and VAC machines. The Adaptive Aggregates project was developed by Phil Ayres, Hollie Gibbons and Corneel Cannaerts, CITA Copenhagen. All other sketches and projects by the author.

REFERENCES
Bilda, Z and Demirkan, H 2003, 'An insight on designer's sketching activities in traditional versus digital media,' Design Studies, 24, pp. 27-50
Blythe, R and van Schaik, L 2016, 'What if Design Practice Matters?', in Fraser, M (eds) 2016, Design Research in Architecture, an Overview, Ashgate, Burlington, pp. 53-69
Burry, M 2011, Scripting Cultures: Architectural Design and Programming, Wiley
Cannaerts, C 2015a, Negotiating Agency: Computation and Digital Fabrication as Design Media, Ph.D. Thesis, RMIT University
Fischer, T 2008, Designing (tools (for designing (tools for ...))), Ph.D. Thesis, RMIT University
Goel, V 2005, Sketches of Thought, MIT Press
Maeda, J 2004, Creative Code: Aesthetics + Computation, Thames & Hudson
Mcluhan, M 1964, Understanding Media: The Extensions of Man, McGraw-Hill
Meredith, M 2008, From Control to Design: Parametric/Algorithmic Architecture, Actar
Meredith, M and Sample, Hilary 2013, Everything All at Once: The Software, Videos, and Architecture of MOS., Princeton Architectural Press
Schaeverbeke, R and Heylighen, A 2012 'In Search of the In Between', In Physical Digitality: Proceedings of the 30th eCAADe Conference, Prague, pp. 49-57
Schön, D 1983, The Reflective Practitioner, Temple Smith
Verbeke, J 2013, 'This is Research by Design', in Fraser, M (eds) 2013, Design Research in Architecture, an Overview, Ashgate, pp. 137-160