N. Gu, S. Watanabe, H. Erhan, H. Haeusler, W. Huang (eds.), *Rethinking Comprehensive Design: Speculative Counterculture, Proceedings of the 19th International Conference of the Association of Computer-Aided Architectural Design Research in Asia CAADRIA 2014,* 000–000.

DIGITAL MINDS, MATERIALS, AND ETHICS: LINKING COMPUTATIONAL THINKING AND DIGITAL CRAFT

NICHOLAS SENSKE
University of North Carolina at Charlotte, Charlotte, NC, USA
nsenske@uncc.edu

Abstract. This paper describes the connections between computational thinking and digital craft, and proposes several ways that architectural education can cultivate better digital craft, specifically: motivating the use of computational strategies, encouraging a conceptual understanding of computing as a medium, teaching computer programming, and discussing digital ethics. For the most part, these subjects are not widely taught in architecture schools. However, moving forward, if the profession values good design, it must also value good digital craft, and ought to instil a way of working in the next generation of architects that makes the most of both the computer and the designer. Computational thinking provides a common foundation for defining and instilling this critical mindset and, therefore, deserves greater consideration within architectural pedagogy.

Keywords. Digital craft; computational thinking; ethics

1. Introduction

Digital craft is a still-evolving notion in architecture. The discipline has not yet reached the point where a majority of architects can recognize and appreciate the craft of procedural logic within a parametric model (to give one example), the same as they might assess and value the craft of a physical artifact. Architects understand even less about digital craft as a process. What sort of mindset do the best digital designers bring into their work with the computer? How are their methods different from typical software users? This paper considers these questions and proposes that computational thinking, a fundamental understanding of the principles of computing as they apply to one's work (Wing, 2006), is essential to defining and producing good digital craft. This proposal has repercussions for how architects create with

all forms of computing and suggests a basis for a more progressive pedagogy of digital skills within architectural education.

2. Thoughtful effort – computational strategies and the digital mindset

According to Richard Sennet, one of the ways a craftsman distinguishes himself from the average worker is through the application of thoughtful effort to the task at hand: using tools with the minimum amount of effort while producing the maximum result (2008). This economy arises not only because the craftsman knows the most effective techniques, but also because he has a sense of his work and his medium that guides him in applying these techniques. Working in this manner is not automatic, as it might appear. Good craft requires a conscious *choice* on the part of the craftsman. When it comes to computing, the vast majority of architects seem to lack this kind of sensibility. They can use software, but their methods are seldom the most efficient and effective for the task. Few of them truly care or even acknowledge the difference. In this sense, their craftsmanship with the computer is poor.

Given the considerable amount of time that the average architect spends at the computer, it is surprising that so few come to develop good digital craft. However, for most computer users, experience with software does not correlate with more thoughtful effort. Some evidence for this comes from human-computer interaction (HCI) research. For instance, two seminal studies by Rosson (1983) and Nilsen et al. (1993) examined skill development with office software (a word-processor and spreadsheet, respectively) over many months. They found that while users did improve their task performance over time, most of their gains came from faster command selection and keystroke times. The studies also identified expert users who performed significantly better than experienced users – often several times better. Rosson and Nilsen found that these differences were not because the experts were the fastest with the mouse and keyboard. Instead, they worked *differently*, utilizing more sophisticated commands and more effective command sequences to accomplish tasks with fewer operations. In other words, expert users were more thoughtful about their work.

The HCI studies document typical user behavior. Most people learn how operate software, and improve their performance with it by learning how to use more commands or work faster, rather than learning more powerful strategies. To give an example, an experienced designer might be able to spend hours in a vector drawing program, repeating the same sequence of commands over and over to create one version of an intricate pattern. While it might accomplish the goal, this method does not make sense when most

programs allow sequences of commands to be recorded, automated, and modified, which is a far more efficient and flexible way of developing a design. However, these operations are oftentimes hidden beneath the surface functions of the program and might never be discovered, even by those with significant experience. Nevertheless, they exist because they are fundamental to the nature of computing. Computational thinking would teach students that these operations exist, or else provoke a search for them within the software. The powerful strategies of expert computer users are derived from these computational attributes of software: exploiting automation, dependencies, and propagation, structuring information, filtering, etc. Knowledge of computational strategies is precisely what is missing in many architects' approach to computing.

What prevents architects from learning computational strategies and improving their craft? One explanation is that learning computers tends to be task-oriented and ad hoc. Architects have limited time to meet their many deadlines. This makes them tend to focus on learning new commands and accomplishing the task at hand rather than learning the most effective ways to use their software – in other words, good craft. Carroll and Rosson (1987) identified this trait in computer users as *production bias*. Architects have been shown to demonstrate this bias (Bhavnani, 1996). There is not much incentive for architects to think deeply about computing as, in their minds, they can only afford to be interested in what is necessary to get the job done. Since command-level knowledge appears to be good enough for their needs, architects tend to limit their actions to what they already know how to do, rather than deliberately trying to improve their performance.

Even if a person has been shown more effective processes, production bias is difficult to overcome. Pea's 1983 study of students learning LOGO programming illustrates how knowledge of computational methods is necessary, but not sufficient for performance improvement (ibid.). Pea's students were taught about looping structures, but, in their programs, most defaulted to writing and modifying lists, line-by-line. This is a more explicit, but far less powerful method than implementing a simple algorithm. When the students where asked why they did this, one replied that it was "easier to do it the hard way." In this particular student's mind, it was less work to write everything out and change each statement manually than to plan and implement a procedural representation. The benefits did not seem to outweigh the assumed cognitive and time costs. This may explain why many architects fail to develop better digital craft.

To prevent this from happening, architects need to acquire the craftsperson's mindset for recognizing and choosing the best technique. They must

learn how to consciously think about the way they work at the computer. The research of Bhavnani, John, and Flemming illustrates how this can be taught within in an architectural context[1]. By articulating computational strategies and explicitly teaching why and when they apply, their work demonstrates how novices can obtain the expert performance of a craftsman in a relatively short amount of time. To achieve this, Bhavnani and John compiled a set of strategies for CAD by conducting a cognitive task analysis of drawing activities (Bhavnani and John, 1997). When novice CAD users received training in the strategies, they were found to perform complex drawing tasks in less time and with more accuracy than users who learned only commands (Bhavnani et al., 1999). In later comparative studies, Bhavnani and his colleagues went on to repeat this training effect in users of other kinds of authoring software such as Unix, Microsoft Office, and Dreamweaver (Bhavnani et al., 2008). This research might seem less poetic than an intimate internship with a master digital craftsperson, but the authors' ability to instill a sense of thoughtful effort in large groups of users is noteworthy. Their research suggests that teaching good digital craft to large numbers of students at a time is possible within a standard curriculum.

Ask an architect whether digital craft is important and most will answer in the affirmative. However, craft is seldom emphasized in the way that architects learn computing. Most computing courses, tutorials, and books focus on the purely operational details of software — which are not difficult to teach and learn (Pea, 1983; Soloway, 1986; Kay 1993)— and do little to cultivate a sense for what it means to use a computer well. To change this, educators must teach computational thinking: demonstrating computational strategies and encouraging students to use them by motivating deliberate practice. If students can be shown the advantages of thoughtful effort, they will be more likely to overcome their production bias and their craft may improve.

3. Material intimacy – mental models, computational concepts, & code

Besides technical skills, the craftsperson is intimately acquainted with her chosen medium. Adapting to the constraints and opportunities of the medium, rather than slavishly following procedures, leads to improvisation and innovation. This responsiveness is recognized as a mark of true craftsmanship. Unfortunately, most users have a superficial understanding of the digital medium (Sheil, 1983). Generally speaking, they do not understand that the details of what happens inside of the computer — which seem unimportant — are, in fact, critical to making the best use of one's time and ef-

fort. Because many architects do not speak the fundamental language of computational concepts, structures, and code, most digital work tends to be rote and superficial, the very antithesis of good craft.

Evidence of this can be found in the upper-year studios at most any architecture school. There, one will often see the same formal tropes repeated time and again – out of control NURBS surfaces, attractor fields, Voronoi tiling, and so forth. This "computational aesthetic" may be the product of fashion, but it is also due to the narrow perspective many students have regarding computing. A criticism of contemporary digital design is that it tends to draw from a limited set of algorithms and techniques (Watz, 2009). One can often determine which software created which forms. Such rigid and unoriginal expression runs contrary to the powerful flexibility and open-endedness of the medium. Where is the craft?

Reflecting on the use of software by artists and designers, John Maeda once said "skill in the digital sense is nothing more than knowledge, and we implicitly glorify rote memorization as the basis of skill for a digital designer (1999)." Most computing and programming courses (and their learning materials) over-emphasize command knowledge and syntax (Robins et al, 2003; Lockhard, 1986; Kölling, 2003). While the software controls must be learned, this alone does not help students develop a sense of the digital medium underlying their work. What they pick up from courses, books, online tutorials, and each other are piecemeal techniques, disconnected from the big ideas of computation. And so, students tend to copy solutions and call upon aesthetics without an understanding of how these work or where and when they apply.

This dependency on rote patterns contributes to a "plug and chug" mentality towards digital design. Because the architect has no grasp of first principles and no sense of what it means to think computationally[2], the only answer seems to be to try all known procedures or to make the design fit one of them. In this sense, the architect who simply knows a large number of procedures may feel skilled. However, as anyone who plugged and chugged his way through math or physics class knows, this does not represent understanding. It is not an effective practice in all circumstances and not conducive to producing original ideas.

The problem is that may architects tend to focus on command recall and either ignore – or have difficulty visualizing – structure within the symbolic context of the computer. As Malcolm McCullough (1998) explains in Abstracting Craft, possessing a mental model is the key difference between mindless rote work and mindful practice. To understand operations conceptually, a designer needs to internalize a high-level model of the computer and

its software. This includes things like system state, but also a sense of the program's data structure and procedural flow. Designers who have a mental model are able to choose or plan their actions, rather than following predefined, brittle routines. They can anticipate the output of an operation and make judgments without running the full program. In short, they have the sort of intuition that separates experts from unskilled and undisciplined users. Any notion of craft or thoughtfulness with design software requires a robust mental model.

Another cause of poor digital craft could be that architects do not understand what is happening computationally while they are using software. Pea and Kurkland (1984) refer to this as "production without comprehension." To cite a specific example: The author once had a student who encountered a problem while making a digital model. Suddenly, for no apparent reason, her program began to create geometry upside down. She tried toggling a series of settings, but because she did not know the nature of the (seemingly) strange behavior, or the functions of many of the settings, she did not get far in her efforts. After several minutes of this, she overcame the problem by continuing to make the upside-down geometry and then manually flipping it to the correct orientation. This kind of experience is common, and it illustrates how students who spend years in school working with software often maintain unclear ideas about how it works. The author has observed other students respond to the same problem by creating an upside-down camera. Some even resort to recreating the model in a new file[3]. These methods are adaptations to what appears to be a software bug. However, the program is not broken, as the students suspect. They are simply misusing it.

The modeling problem occurred because the internal state of the program changed. The coordinate system was reversed; probably by an errant keystroke. The students ran into trouble because they did not know anything about the system beyond the interface and their 3D model. Only what was visible on the screen seemed immediately important to them. This surface-level mindset extended to their response. The only way they knew how to address the problem was through their knowledge of commands: e.g. flipping the geometry, making a different camera, and starting a new file. Granted, these responses are solutions or adaptations of a sort –the students were able to keep working—but valuable time was wasted because they did not comprehend the problem well enough to come up with the correct solution. What they saw as a bug was actually a different state within the system, which they failed to manage.

In the above example, the students made models with the computer, but did not comprehend the system they were using to create them. Their prob-

lems illustrate how the inability to visualize and reason computationally can result in poor digital craft. Why is it so difficult for the average user to figure out these systems? The problem is one of abstraction and transparency. Whether it is a CAD program, a spreadsheet, or a word processor, most software is a "black box." The user can interact with the interface on the outside, but the internal logic remains hidden. This makes it difficult, if not impossible, to understand how or why the program works, which can lead to misconceptions about what the computer is doing (Sheil, 1983; Norman, 1988). The user, expecting a simple cause-and-effect relationship between an action and a response, may not be aware of the procedural logic involved. For instance, the computational state of the software may have been affected by an earlier operation. As described in the example of the upside-down model, a user who does not understand this might interpret future results as erroneous, when these are logical within the system (Blackwell, 2002). Without a computational mindset, the user has no way to properly diagnose the source of the problem. Attempts to fix the problem might involve randomly toggling options and strange workarounds. Thus, a lack of awareness regarding the digital medium often results in haphazard, undisciplined user behavior. In this manner, poor craft can lead to lost productivity and, potentially, designs of poor quality.

A good mental model can help guide how a person uses the computer, but this only addresses part of the problem. Another cause of rote work is an attitude that views computational processes as static and linear; a recipe to obtain a certain result rather than something to be explored and refined itself. Because computers can become nearly anything, restricting oneself to a fixed palette of tools and techniques is missing out on the medium's potential. An architect who does not take advantage of modifying their tools — to overcome limitations or to explore— is severely limited.

Although most people do not think about it this way, programming is an important skill for making effective use of almost any software and expressing computational thinking. Knowing how to code is indispensable part of learning and practicing good digital craft.

Unfortunately, many still believe that teaching architects to write programs – even small scripts – is unnecessary and too much additional work. However, others argue that the benefits are worth the effort. Reas and Fry (2006) observe that it was not unusual in the past for artists to mix their own pigments or prepare brushes to get the effect they wanted in their work. Besides giving them more control, these tasks gave the artist a greater intimacy with the material. Reas and Fry contend that, in a similar manner, making digital tools can give designers a sense of the computer as a material. And

so, programming knowledge can help architects regain some control and ownership over how they work with computers, making them less of an operator and more of a craftsperson.

4. Computing ethics – what-for, avoiding distractions, respecting human values

Another aspect of good craftsmanship is the ethic that one brings into their work. It is not enough to have "know-how." Good craft must also answer the questions of "why" and "what-for." Taking part in a craft's culture involves learning its ethics. For instance, there are norms of appropriateness, constructive aims, and good taste associated with various styles of writing. It is fair to assume that the same is true for computing. However, computing ethics are seldom articulated, discussed, or enforced. Thus, a lack of a commonly shared ethic is another cause of poor digital craft.

Good digital craft demands a philosophy of why, when, and how one should engage (or not engage) in computing. This is not something that many users consider. There is often an assumption that if something can be done on a computer, then it probably should be. This is untrue, of course. Ethics are needed because the capabilities of computing can be a source of distraction.

One example of this distraction is when learning new technology gets in the way of learning and making architecture (McCullough, 1996). Some students spend so much of their time attempting to master the latest tools and software that quality, engagement, and common sense get left behind. Overshadowed by the pursuit of methodology, design projects become one-liners, or worse, go incomplete; the means become the end. Without digital ethics, it is far too easy to get carried away and use the computer to make things overly complicated, garish, and wasteful.

To give another example of the need for ethics, sophisticated computer programs, such as those that perform environmental analysis, are increasingly used by our students as part of their design projects. A problem is that students often abuse these by offloading much of their thinking to algorithms. They forget (or seem to disregard) that programs cannot synthesize output and make decisions; that simulations do not automatically design a good building. Over-dependence on programmed logic can cause a person to think deterministically, to depend too much upon systems as the only source of design solutions rather than intuition and sensibility. More importantly, too much faith in algorithms can make a person overlook when the solution is simply wrong. Like the proverbial drunk looking for his keys, many stu-

dents confine their searches beneath the streetlight of computing. They lack a sense of ethics in their craft.

An ethical perspective recognizes the different strengths of both people and computation. This is critical if architects are to engage the full potential of computers in a manner that respects human values and intentionality. Recognizing the unique contributions of humans and computers within design is a critical insight for architects to comprehend. At the moment, this subject does not receive enough attention within architecture. While arguments over analog versus digital methods are familiar territory, these tend to revolve around traditions of drawing and modeling. Less common are serious discussions about the roles of human and machine thinking in architecture, now that the automation of design and production has entered into widespread use. To prevent logic from getting in the way of design, schools and professionals need to articulate and discuss the limitations and pitfalls of computing.

5. Conclusion

This paper described the connections between computational thinking and digital craft, and proposed several ways that architectural education can cultivate better digital craft, specifically: motivating the use of computational strategies, encouraging a conceptual understanding of computing as a medium, teaching computer programming, and discussing digital ethics. These subjects are not widely taught in architecture schools. However, moving forward, if the profession values good design, it must also value good digital craft, and ought to instil a way of working in the next generation of architects that makes the most of both the computer and the designer. Computational thinking provides a common foundation for defining and instilling this critical mindset and, therefore, deserves greater consideration within architectural pedagogy.

Endnotes

1. For a full summary, see (Bhavnani and John, 2000).
2. By which I mean, one does not have a computational design process: a sense of how to apply first principles to produce a solution. The only strategies one knows are exhaustive ones such as pattern matching and random trial and error.
3. This technique succeeds because it resets the system state.

References

A Bhavnani, S. K.:1996, CAD usage in an architectural office: from observations to active assistance, *Automation in Construction*, **5**: 243-255.

Bhavnani, S. K. and B. E. John: 1997, From Sufficient to Efficient Usage: An Analysis of Strategic Knowledge, *Proceedings of CHI '97 Conference on Human Factors in Computing Systems*, 91-98.

Bhavnani, S. K., B. E. John, et al.: 1999, The strategic use of CAD: an empirically inspired, theory-based course, in *Proceedings of the SIGCHI conference on Human factors in computing systems*, Pittsburgh, Pennsylvania, United States, ACM.

Bhavnani, S. K. and B. E. John: 2000, The strategic use of complex computer systems, *Human-Computer Interaction* **15**: 107-137.

Bhavnani, S. K., F. A. Peck, et al.: 2008. "Strategy-Based Instruction: Lessons Learned in Teaching the Effective and Efficient Use of Computer Applications." ACM Transactions in Computer-Human Interaction **15**(1): 1-43.

Blackwell, A. F.: 2002, What is Programming?, *14th Workshop of the Psychology of Programming Interest Group*, Brunel University.

Burry, M.: 2011, *Scripting Cultures*, Chichester, Wiley.

Carroll, J. M. and M. B. Rosson: 1987, Paradox of the Active User, *Interfacing thought: Cognitive aspects of human-computer interaction*, in J. M. Carroll. (Ed.) Cambridge, MA, MIT Press, 80-111.

Kay, A.: 1993, The Early History of Smalltalk, *ACM SIGPLAN Notices*, **28**(3), 69-95.

Kölling, M.: 2003, The curse of hello world, *Workshop on Learning and Teaching Object-orientation - Scandinavian Perspectives*, Oslo.

Lockard, J.: 1986, Computer programming in the schools: What should be taught?, *Computers in the Schools*, **2**(4), 105-113.

Maeda, J.: 1999, *Design by Numbers*, Cambridge, MIT Press.

Maeda, J.: 2004, *Creative Code*, New York, Thames & Hudson.

Mateas, M. and A. Stern: 2005, Procedural Authorship: A Case Study of the Interactive Drama Façade, *Digital Arts and Culture (DAC)*, Copenhagen.

McCullough, M.: 1998, *Abstracting Craft*, Cambridge, MIT Press.

McCullough, M.: 2006, 20 Years of Scripted Space, *Architectural Design*, **76**(4): 12-15.

Nilsen, E., H. Jong, et al.: 1993, The growth of software skill: a longitudinal look at learning & performance, *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, Amsterdam, The Netherlands, ACM.

Norman, D. A.: 1988, *The Design of Everyday Things*, New York, Doubleday.

Pea, R. D.: 1983, Logo Programming and Problem Solving [Technical Report No. 12.], *American Educational Research Association Symposium*. Montreal, Canada.

Pea, R. D. and D. M. Kurkland: 1984, On the cognitive effects of learning computer programming, *New Ideas in Psychology,* **2**, 137-168.

Perkins, D. N. and G. Salomon: 1989, Are Cognitive Skills Context-Bound?, *Educational Researcher*, **18**(1), 16-25.

Reas, C. and B. Fry: 2006, Processing: programming for the media arts, *AI & Society*, **20**(4), 526-538.

Robins, A., J. Rountree, et al.: 2003, Learning and teaching programming: a review and discussion, *Computer Science Education*, **13**(2), 137-172.

Rosson, M.:1983, Patterns of experience in text editing, in *Proceedings of CHI '83 Conference on Human Factors in Computing System*s, 171-175.

Sennet, R.: 2008, *The Craftsman*, Yale University Press, New Haven.

Sheil, B. A.: 1983, Coping with Complexity, *Information Technology & People*, **1**(4): 295-320.

Soloway, E.: 1986, Learning to program = learning to construct mechanisms and explanations, *Communications of the ACM*, **29**, 850-858.

Watz, M.: 2012, The Algorithm Thought Police, *http://mariuswatz.com/mwatztumblrcom/the-algorithm-thought-police.html.* Accessed January 23, 2014.

Wing, J. M.: 2006, Computational Thinking, *Communications of the ACM,* **49**(3), 33-36.