The Nature of Computational Thinking in Computing Education

Thomas Hvid Spangsberg and Martin Brynskov

***Abstract*—Computational Thinking has gained popularity in recent years within educational and political discourses. It is more than ever crucial to discuss the term itself and what it means. In June 2017, Denning articulated that computational thinking can be viewed as either "traditional" or "new". New computational thinking highlights certain skills as desired in solving problems, whereas traditional computational thinking is a skill set resulting from engaging in traditional computing activities. By looking at computational thinking through the perspective of semiotics, it is possible to dissolve the traditional vs new distinction and concentrate on computational thinking having both an explicit and implicit nature. In this perspective, a computer program becomes an algorithmic sign which can both be interpreted by humans and machines. The double interpretation allows for a dialectic relationship between computing activities and Computational Thinking instead of the dualistic traditional vs new approach.**

***Index Terms*—Computational thinking, constructionism, computing education, the notional machine.**

I. Introduction

Computing education has gained much interest over the last decade. The importance of computing skills in disciplines outside of traditional computer science (CS) is apparent in initiatives like *Computing at School* [1] in the UK and *Computer Science for ALL* [2] in the USA. There are many ways to talk about *computing*. This paper presents a suggestion based primarily on computational thinking (CT). In addition to CT, it also includes prior articulations of Seymour Papert's *constructionism* [3]-[5] and Benedict Du Boulay's *notional machine* [6]. CT has become popular in education from K-12 to higher education [7], [8] and many attempts have been made to define an operational description of the term to implement it in an educational context. Recently, a critique has been raised against CT for having lost its heritage from traditional computing disciplines [9], [10]. This opens for a discourse of a dualistic understanding of CT – *traditional* and *new* as Denning [9] labels it. It means that when turning to CT in education, decision-makers have to choose which kind of CT to teach. In itself, that is not a problem. The problem arises when the popular version of CT is chosen because this version is overexposed, even though it may not be the most suitable version for the task at hand. This paper seeks to reform the dualistic understanding of CT into a dialectic relationship between an implicit and explicit nature of CT.

Manuscript received October 30, 2017; revised May 14, 2018. This work was supported in part by the Centre for Computational Thinking and Design which is under formation at Aarhus University in Denmark.

The authors are with the Department of Digital Design and Information Studies, Aarhus University, Denmark (e-mail: tbhs@cc.au.dk, brynskov@cavi.au.dk).

A. Methodological Considerations

The analysis and discussions in this paper builds on previous exploration [11] and are based on analysis and reasoning from the presented positions. No empirical studies have been made. The conclusions reached here are therefore suggestions for further studies, which may take them up as hypotheses. Only a limited selection of the rich history of computing education can be covered. The examples presented in this paper are not the only voices in the field, but their articulations seem to be a fitting introduction to the field with the purpose to give the reader a sound starting point for understanding the main concern for this paper.

II. Talking about Computing Education

In this section, constructionism, the notional machine and CT will be presented.

A. The Notional Machine

Benedict du Boulay is recognised to be the first scholar to articulate the concept of the notional machine [12]. The term is used in the context of computing education, explicitly learning novices how to program. The concept rests on the assumption that the novice learner is ignorant of what the machine (the computer) can do through the instructions of the programming language and how the instructions are actually carried out by the machine [13].

"*The notional machine is an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed.*" [13]

The notional machine can in this sense be understood as an abstraction of the execution of a computer program. It can be characterised as an "*idealized computer*" [12] which supports the novice learner in acquiring a certain programming language. From this will also follow that there may exist many different notional machines. Students may construct a notional machine for each programming language, based on the programming language's construction [13]. The notional machine is also seen as a façade for the inner workings of the real machine through a programming language controlling the actual machine [12]. In this way, the notional machine becomes a mental model of the real machine – an analogy based on the constructs present in a certain programming language. However, this does not always lead to a fruitful understanding of the machine, as it is based on an abstraction of its capabilities presented by the chosen programming

doi: 10.18178/ijiet.2018.8.10.1132

language – and programming languages differ in their design. Ideally, over time, as the novice builds more experience, the mental model or the notional machine gains in fidelity and a deeper and more solid and structured understanding of the real machine emerges. Computing education should, therefore, support this process in the mind of the novice learner [12]. This can be done down to the level of designing programming languages so that they make the working of the notional machine they are supposed to control more explicit. This may result in the language designer building smaller machines that allow the novice to control the main machine and build a mental model at the right level of detail [14].

B. Constructionism

Seymour Papert is commonly recognised as the creator of constructionism, a learning theory based on cognitive constructivism [15]. Cognitive constructivism is a concept to explain knowledge-construction through cognition in children's development [15] whereby students acquire new knowledge through the mechanisms of assimilation and accommodation of cognitive schemata [16]. Assimilation incorporates or interprets new information to fit in excising cognitive schemata. Accommodation alters the schema to fit new information [17]. In a constructivist approach to teaching, the teaching should support the construction of knowledge through the student's cognition. However, cognitive constructivism does not suggest any concrete method of teaching to achieve this. In Papert's constructionism, the cognitive process of the student's mind is externalised into activities that support the knowledge construction taking place. One point that Papert makes is the notion that the computer can aid and change the way children learn. The basic approach in constructionism is explained as "*learning by making*" [4], and the term bricolage is used to describe some of the processes associated with constructionist teaching. It underlines the tinkering and explorative approach encouraged when teaching [4]. The *oppositional principles* that Papert puts forward in the 1996 paper: An Exploration in the Space of Mathematics Educations [5], [10] is a good starting point for talking about computing education through this perspective. In the paper, Papert explores constructionist ways of teaching mathematics, and he puts forward a series of oppositional principles. The three main principles are: The power principle, the thingness principle and the dynamics before statics principle.

The power principle is concerned with what has the power in a learning situation: understanding or application. Papert notes that the natural mode of acquiring knowledge is through use which will progressively lead to a deepening of one's understanding. Out of this also comes the notion of "project before problem". He makes the example that problems arise during project work, and are then solved or dissolved through that process [5].

The thingness principle is related to the concept of reification which is concerned with making abstract ideas concrete through a meaningful representation. The thingness principle is concerned with "objects before operation" in the sense that a mathematical concept is to be understood as a thing first and then examined like a thing to understand the functionality [5].

Finally, the dynamics before statics principle is closely related to the medium used for teaching. Papert is critical of the approach where more often than not mathematical concepts are taught the other way around, that the dynamic nature of mathematics is not adequately captured by the static medium of pen and paper [5].

This notion of knowledge through use is very explicit in the way that Papert lets children explore geometry through the use of a programmable robot (the turtle) drawing shapes on large pieces of paper by moving around based on the programmed instruction. By programming the turtle, children are supported in discovering geometric principles through using them [3]. When children discover geometric principles through the use of the turtle, they engage in an exploration of the principles for them to produce certain shapes in a dynamic environment. This leads to a deeper understanding of the basic principles which can be put to use exploring more advanced subjects, to test assumptions and form conclusions. In a reflection on this practice, Papert mentions that the goal is to "*integrate computational thinking into everyday life*" [3].

C. Computational Thinking

It is outside the scope of this paper to present a comprehensive account of the past 11 years of the CT discourse, but a few key contributions have been selected. It is commonly attributed to Jeanette Wing is commonly attributed [18] to have popularised CT in a contemporary context in the year 2006. The first mention of the term itself seems to be by Seymour Papert in his book *Mindstorms* from 1980 [3], [9], [10]. CT is presented by Wing [18] as a set of problem-solving skills. The concept relies on the fundamental skills and abilities from CS, particularly abstraction and decomposition and applies them to other disciplines, e.g. biology [19]. CT is merely, according to Wing, to think like a computer scientist [18] when approaching a problem and in solving it. This definition has been criticised for being too broad, and attempts have been made to define it further [7]. Wing [20] has since put forward a more elaborate definition:

"*Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.*" [20]

CT has grown in popularity since 2006, and during the past decade, several K-12 institutions have in some way committed incorporating CT in their curricula, hoping it will help more children and teenagers to engage in STEM (Science, Technology, Engineering and Mathematics) activities [21], [22]. Also in higher education, there is a movement to introduce CT as a competence for both STEM and non-STEM students. In the particular case of design, media and informatics students it has been suggested that a CS0 [23] course might be appropriate for them to get a deeper understanding of their material of study.

Barr and Stephenson [7] are concerned with defining CT for use in K-12 education and to explore the role of the CS

community in this. They present an outline for how to incorporate what they have defined as "*core CT concepts and capabilities*" [7] in different subjects across a curriculum. They present these core concepts and capabilities as being: "*Data collection, Data analysis, Data representation, Problem Decomposition, Abstraction, Algorithms & procedures, Automation, Parallelization and Simulation*" [7].

III. A Dialectic Relationship between Implicit and Explicit Computational Thinking

CT has recently been a target for critique by both Barba [10] and Denning [9]. The two scholars each attack CT on their own terms but they reach more or less the same conclusion resulting in an emerging understanding of CT as a term having a different meaning pre- and post- 2006 [9], [10]. In this section, we suggest that instead of viewing CT as either traditional or new [9] a more pragmatic approach of viewing CT is as it having an explicit and implicit nature: this gives a more vibrant picture of what CT is, and also incorporates the other ways of talking about computing education. Spangsberg and Brynskov [11] put forward the idea of looking at CT through computer semiotics to reform the dualistic "traditional" vs "new" distinction into a dialectic relationship between an implicit and an explicit nature of CT. This section will first introduce the concept of the algorithmic sign as a basic framework for combining the three ways already presented and by doing so overcoming the concerns presented by Barba [10] and Denning [9]. Then the actual concerns for CT will be presented and lastly, the suggested solution for overcoming them.

A. The Algorithmic Sign

Spangsberg and Brynskov [11] are not the first to connect CT and semiotics to each other. Souza et al. [24] present a study for where they use a semiotic background for characterising students' perception of CT. We use semiotics differently. Here, we use the notion of the algorithmic sign to augment CT and to introduce a way to combine the three ways presented earlier into one framework. A key factor here is the double-sided interpretation taking place when concerned with the algorithmic sign as opposed to the traditional single-sided interpretation of ordinary signs

The cardinal concept to be understood to talk about the algorithmic sign is the concept of signs – the base for semiotic science. A sign in this context is to be understood as something to stand for something else. The focal domain of a sign is a matter of tradition. The three main traditions focus on signs as a social system (European), the individual use of a sign (American) or a philosophical inquiry where language is its sign type among others (Peircean) [25]. It has been suggested that software is to be understood as an algorithmic sign [26]. Because of signs in general stand for something else, signs need to be interpreted. What is unique about the algorithmic sign is that it not interpreted by humans alone, but also by the machine [26]. It is important to note that the word 'interpretation', when concerned with computers, does not share the nature of the human act of interpretation. The machine lacks the human freedom to choose different perspectives for interpreting. It can only interpret based on strict and formal parameters [27]. To operationalise the algorithmic sign as an analytical framework when it comes to programming. It is helpful to look at the works of Peter Bøgh Andersen [25], who worked together with Frieder Nake on producing a textbook on computer semiotics – including the algorithmic sign. The starting point is the Peircean framework of the semiotic triangle (in the following referred to as simply: "the semiotic triangle") as shown in Fig. 1.

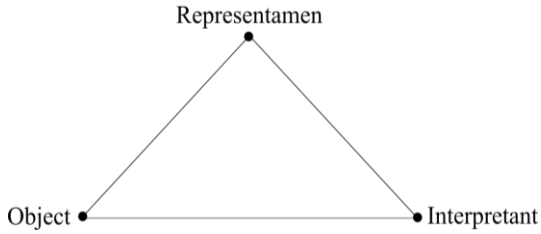

Fig. 1. The Peircean sign concept. Based on Andersen [25]. The *representamen* is concerned with representing the *object* in question and the *interpretant* is the human actor interpreting the sign through the representamen. As a whole, the relationship between the three corners is considered a sign [25].

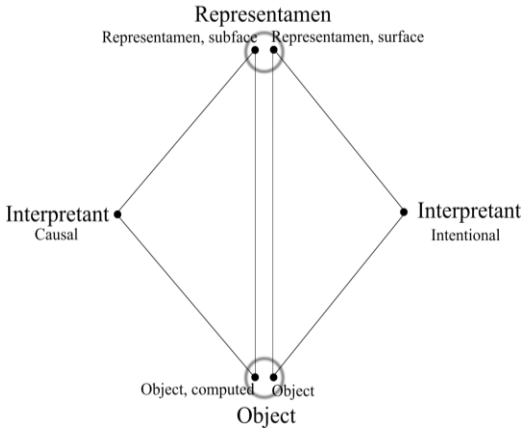

Fig. 2. The algorithmic sign. Based on Lomanto [28]. The triangular framework of Fig. 1 is transformed into a Quadrilateral model. The representamen gain a surface and subface perspective and the object is also enricged with a computed perspective.

The semiotic triangle shows the concept of a sign understood as the relationship between the representamen which represents an object which is interpreted by an interpretant. Consider a computer program – the object, is represented in the program code – the representamen. A programmer – the interpretant, interprets the program code while engaging with the object through the representamen The algorithmic sign takes into consideration both the machine and human side of the interpretation by introducing a *subface* and a *surface* attribute to the representami. Depending on the operating interpretant, the representamen will be interpreted from either perspective [27]. It is important to note that there are not two representamens in the framework, but the same representaem is present in both semiotic processes but with different attributes towards the interpretant. At the surface side, the human is operating as the interpretant, and at the subface side, the interpretant is the computer, with the restrictions noted earlier. Normally, the semiotic processes at the subface side are not accessible to the human interpreter [27]. However, when taking into account the role of program code, the human interpretant is

constantly working both with a human interpretation of input/output (I/O) signals and how the machine will interpret – or execute – said signals. The framework is summarised in Fig. 2.

The double triangle consists of two semiotic triangles joined by the subface and surface attributes of the representamen. The left side is concerned with the causal interpretation made by the machine, and the right side is concerned with the intentional interpretation made by the human. The object is also shared between the two sides but because of the different natures of the interpretamnts, the object on the subface side is considered the computed equivalent of the same object on the surface side. Againk, as with the representamen, they are not individual instances but perspectives of the same entity.

Consider a webpage viewed in a web browser – this will be the object. On the surface-side, the representamen will be what the user, the interpretant, is presented with. In this case that would be the graphical appearance of said webpage. Through interaction with the webpage, the user builds knowledge on how the system works on the surface side, for instance, that blue underlined text are links that take the user somewhere else. On the subface-side, the object is still same web page but its representamen will be the HTML-tags for the web browser, the interpretant. In this way, the same webpage is represented in two different ways, but it is still the same web page. If the user learns the language of the web browser (HTML), then the human user gains access to the subface-side of the webpage and the full extent of the representamen. When the human user sets out to build an HTML page, the subface-side of the interpretation needs to be taken into consideration for the web page function according to the anticipated surface representamen. For a link to function and have the right appearance, the correct HTML-tag will have to be placed in the right way according to the way the web browser interprets the finished web page. Having gained the knowledge of how to construct a web page and the language required to do so, a human user can begin to comprehend the subface representamen when looking at the surface version and vice versa.

B. A Dualistic of Computational Thinking

Most recently, in June of 2017, Denning [9] published a viewpoint in Communications of the ACM that suggests CT should be labelled either "traditional" or "new". Denning [9] draws up a series of collision points between traditional and new CT. The most relevant in the context of this paper is the role of programming skills as a result of CT (traditional) or the other way around (new), and whether the understanding of algorithms requires (traditional) or does not require (new) an underlying computational model. Denning is not alone with his critique of the seemingly opposite understandings of CT. In 2016, Barba [10] put forward a similar critique of Wing's [18] version of CT. Barba is particularly concerned with the origins of CT as expressed by Papert [3], [5] and highlights the oppositional principles of Papert [5] as a way to get to Papert's [3], [5] understanding of CT. Barba [10] arrives at the somewhat same conclusion as Denning [9] that CT has a different meaning today than traditional. Both scholars acknowledge Seymour Papert's work when it comes to an understanding of what CT is from a traditional perspective, that is, a constructionist approach to computing education where CT is developed through the engagement in traditional computing practices. In this sense, CT is a result of engaging in programming practice and is not detached from the computational model. This corresponds with what Denning [9] labels traditional CT. According to Jeanette Wing new CT is detached from traditional computing practice and is instead a set of problem-solving skills derived from that donain. However, a computational thinker is not required to engage in traditional computing practices to learn CT. In critique of this, Denning observes that computing skills follow from learning CT as he notes that new CT is "*a conceptual framework that enables programming*" [9]. In Wing's elaborated definition the machine is present as "(…) *information-processing agent*" [20]. However, the machine's primary purpose is to solve the problems and follow the algorithms put forward by the computational thinker. This is in contrast to the traditional understanding, where engagement with the machine helps to develop CT skills.

C. A Suggestion for Overcoming the Dualistic Understanding of Computational Thinking

By looking at CT through the lens of the algorithmic sign, the double interpretation opens the idea of viewing CT as one whole which can take on one of two natures depending on the context [11]. This idea can be linked to the notional machine in the sense that the user through exploring the surface-side of the representamen can achieve an understanding of how the machine works. This understanding can be enhanced by engaging in the subface-side as well. In an educational setting, the first establishment of the notional machine in a novice's mind may very well be given through the surface-side with a gradual transition into the subface-side of things at a later point. The point here being that with the algorithmic sign, the notional machine is not restricted in developing further detail at the subface-side even though the surface side is the starting point. In fact, we suggest that the goal is not reached just by arriving at the subface-side if the knowledge is not linked back to the surface-side. Taking the viewpoint of constructionism, the oppositional principles can be interpreted as transitions between the surface and the subface sides. The power principle describes the dialectic relationship between understanding and application. As an analogy, the subface-side of the algorithmic sign can represent the understanding and the surface-side the application of a geometric principle. Both sides are equally valuable in building knowledge about geometric principles. The thingness principle, in the same way, can be captured by this analogy. The surface-side is a fully functioning webpage which can be explored. Eventually, the subface-side can be reached when the building blocks of a webpage are learned. The dynamics before statics principle ensures that knowledge on web pages are taught through a medium that enables these transitions to take place.

The examples given above seek to show that there is a dialectic process taking place when treating the algorithmic sign as a framework for computing education. Applying this dialectic to CT results in the emergence of an implicit nature expressed in "traditional" CT and an explicit nature

expressed in "new" CT [11]. Instead of keeping traditional and new CT as a dualism where one is forced to make an either-or decision, the algorithmic sign opens up the possibility for one CT concept. This CT concept than can put on an implicit or an explicit nature depending on the context. In the case of forming the foundation for constructing a notional machine to aid the novice learner in picking up computing skills, an explicit nature of CT may be applied while still letting the learner work on a sub-face representamen of a computer program. An implicit nature of CT would then operate when the learner gains more experience in the subface-side of a computer program which refines the initial explicit concepts of CT. There would in this concept be a place for both the traditional and new versions of CT, but neither of them stand alone.

IV. Conclusion

In this paper, computational thinking has been explored through the perspective of semiotics. The algorithmic sign has been utilised to gather past articulations on computing education which can be seen as early articulations of computational thinking. Labelling computational thinking as either "traditional" or "new" can also be viewed as having an implicit or explicit nature. The lack of the implicit nature and the disconnect with traditional computing practices of "new" computational thinking causes opposite or ambiguous understandings of computational thinking as a whole concept. Through the use of a semiotics perspective, a computer program becomes a sign – an algorithmic sign for interpretation. This includes the surface-side of the human and the subface-side of the machine at the same time. This helps augment the understanding of computational thinking as a dialectic relationship between an implicit and explicit nature. This dialectic relationship is necessary to capture both understandings of computational thinking. It enables decision-makers in education to choose computational thinking as a single, rich concept.

Acknowledgment

The authors would like to thank their colleagues for valuable discussions of computational thinking, especially, Michael Caspersen Director of IT-vest, Clemens Klokmose and Peter Vahlstrup from Department of Digital Design and Information Studies at Aarhus University. Thomas Hvid Spangsberg thanks, prof. Sally Fincher from the Computing Education Group at University of Kent's School of Computing for valuable feedback and discussion of the topics of this paper.

References

[1] Computing at School. (2017). [Online]. Available: http://www.computingatschool.org.uk
[2] Computer Science for All. (2016). [Online]. Available: https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all
[3] S. Papert, *Mindstorms: Children, Computers and Powerfull Ideas*, 1st ed. 1980, New York: Basic Books, Inc.
[4] I. E. Harel and S. E. Papert, *Constructionism*, Ablex Publishing, 1991.
[5] S. Papert, "An exploration in the space of mathematics educations," *International Journal of Computers for Mathematical Learning*, 1996. vol. 1, no. 1.
[6] J. Monk *et al*., *The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices*, vol. 51, 1981, pp. 237-249.
[7] V. Barr and C. Stephenson, *Bringing Computational Thinking to K-12: What Is Involved and What Is the Role of the Computer Science Education Community?* ACM Inroads, 2011, vol. 2, no. 1.
[8] T. Li and T. Wang, *A Unified Approach to Teach Computational Thinking for First Year Non–CS Majors in an Introductory Course*, 2012, vol. 2, pp. 498-503.
[9] P. J. Denning, "Remaining trouble spots with computational thinking," *Commun*, ACM, 2017, vol. 60, no. 6, pp. 33-39.
[10] L. Barba. (2016). Computational Thinking- I do not think it means what you think it means. [Online]. Available: http://lorenabarba.com/blog/computational-thinking-i-do-not-think-it-means-what-you-think-it-means/
[11] T. H. Spangsberg and M. Brynskov, *Towards a Dialectic Relationship Between the Implicit and Explicit Nature of Computational Thinking – A Computer Semiotics Perspective*. in *Koli Calling 2017*, 2017.
[12] J. Sorva, "Notional machines and introductory programming education," *Trans. Comput. Educ*., 2013, vol. 13, no. 2, pp. 1-31.
[13] J. Monk *et al*., *The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices*, vol. 51, 1981, pp. 237-249.
[14] B. D. Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, 1986, vol. **2**, no. 1, pp. 57-73.
[15] E. Ackermann, "Piaget's Constructivism, Papert's Constructionism: What's the difference?" *Constructivism: Uses and Perspectives in Education*, 2001.
[16] T. H. Spangsberg and M. Brynskov, "Code-labelling: A teaching activity encouraging deep learning in a non-STEM introductory programming course," presented at ICCSE 2017, 2017, Houston, TX USA.
[17] J. Bryant and D. Miron, "Theory and research in mass communication," *Journal of Communication*, 2004, vol. 54, no. 4, pp. 662-704.
[18] J. M. Wing, "Computational thinking," *Communications of the ACM*, 2006, vol. 49, no. 3.
[19] J. M. Wing, "Computational thinking and thinking about computing," *Philos Trans A Math Phys Eng Sci*, 2008, vol. 366, no. 1881, pp. 3717-25.
[20] J. M. Wing, "Research Notebook: Computational Thinking-What and Why?" *The Magazine of the Carnegie Mellon University School of Computer Science*, 2017.
[21] M. Goldweber, "Programming should not be part of a CS course for non-majors," *ACM Inroads*, 2015, vol. 6, no. 1.
[22] P. Sengupta *et al*., "Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework," *Education and Information Technologies*, 2013, vol. 18, no. 2, pp. 351-380.
[23] A. Brady, P. Cutter, and K. Schultz, "Benefits of a Cs0 course in liberal arts colleges," *Consortium for Computing Sciences in Colleges*, 2004.
[24] C. S. D. Souza *et al*., "Semiotic traces of computational thinking acquisition," in *Proc. the Third International Conference on End-User Development*, 2011, Springer-Verlag: Torre Canne, Italy, pp. 155-170.
[25] P. B. Andersen, "Semiotic models of algorithmic signs," *Algorithmik – Kunst – Semiotik. Hommage für Frieder Nake*, 2003, Synchron Wissenschaftsverlag der Autoren: Heidelberg, pp. 165-210.
[26] F. Nake, "Computer art: A personal recollection," in *Proc. the 5th Conference on Creativity & Cognition*, 2005, ACM: London, United Kingdom, pp. 54-62.
[27] F. Nake, "Surface, interface, subface. Three cases of interaction and one concept," *Paradoxes of Interactivity. Perspectives for Media Theory, Human-Computer Interaction, and Artistic Investigations*, Berlin: Transcript, 2008, pp. 92-109.
[28] I. Lomanto, *Art after the Algorithmic Revolution - A Semiotic Approach to Digital Art*, 2011, University of Applied Sciences Bremerhaven: University of Applied Sciences Bremerhaven.

Thomas Hvid Spangsberg (male) was born in Esbjerg, Denmark on 7th of September 1983. He got the BSC in IT product design, 2010 at Aarhus University, Denmark; the MSC in digital design, 2012, Aarhus University Denmark; the PhD fellow in the field of information studies, 2015-2018 at Aarhus University.

He is a PhD-Fellow at the Department of Digital Design and Information Studies at Aarhus University and was a research assistant for two and a half years at the same department tasked with teaching programming to non-STEM (Science Technology, Engineering and Mathematics) students before beginning his doctoral studies. Before beginning his University studies, Spangsberg served in the Danish Army as a Sergeant and is still an active reservist. Spangsberg's publications

include: Spangsberg, T. H. and Brynskov, M. Code-labelling: A Teaching Activity Encouraging Deep Learning in a non-STEM Introductory Programming Course. In *Proceedings of the ICCSE 2017* (Houston, TX USA, 2017) and Spangsberg, T. H. and Brynskov, M. Towards a Dialectic Relationship Between the Implicit and Explicit Nature of Computational Thinking – A Computer Semiotics Perspective. *Proceedings of Koli Calling 2017* (Joensuu, Finland, 2017 - In print) Spangsberg's main research interest is teaching programming to non-STEM novices and computing education in general.

Martin Brynskov (male) was born in Aarhus, Denmark on 18th of November 1971. He got the MA in information studies and classical greek, Aarhus University. Thesis title: *Digital Habitats* (2003); Ph.D. in computer science (HCI), Dept. of Computer Science, Aarhus University. Dissertation title: *Tools for Social Construction* (2007).

He is associate professor in interaction technologies at Aarhus University, Dept. of Digital Design and Information Studies (2011-), previously associate professor (2008-2011). Recent works include Raetzsch, C., Brynskov, M. (forthcoming). "Challenging the Boundaries of Journalism through Communicative Objects: Berlin as a Bike-friendly City and #Radentscheid" in *Paragrafo*; Foth, M., Brynskov, M., Ojala, T. (Eds.) (2015). *Citizen's Right to the Digital City*, Springer; Foth, M. & Brynskov, M. (2015). "Participatory Action Research for Civic Engagement" in Gordon et al. (Eds.) *Handbook of Civic Technologies*, Cambridge, MA: MIT Press; and Brynskov, M., Halskov, K., Dalsgaard, P. (2015). "Media Architecture: Engaging Urban Experiences in Public Space", in *The Use of Art in Public Space*, Routledge. Current research focus is on smart cities and the internet of things, where he coordinates some of the largest projects in the field, including OrganiCity (7.2m€) and SynchroniCity (20m€).

Dr. Brynskov is chair of the global Open & Agile Smart Cities initiative (more than 100 cities in 23 countries), member of the Association for Computing Machinery Special Interest Group for Computer-Human Interaction, and vice-chair of the UN ITU-T Focus Group on Data Processing and Management to support IoT and Smart Cities & Communities.