Dufva, Tomi; Dufva, Mikko

Metaphors of code

Published in:
THINKING SKILLS AND CREATIVITY

DOI:
10.1016/j.tsc.2016.09.004

Published: 19/09/2016

Document Version
Publisher's PDF, also known as Version of record

Thinking Skills and Creativity 22 (2016) 97–110

Contents lists available at ScienceDirect

Thinking Skills and Creativity

journal homepage: http://www.elsevier.com/locate/tsc

Metaphors of code—Structuring and broadening the discussion on teaching children to code

Tomi Dufva[a,*], Mikko Dufva[b]

[a] Aalto-University, School of Arts, Design and Architecture, Finland
[b] VTT Technical Research Centre of Finland Ltd, Tekniikankatu 1, Tampere, P.O. Box 1300, 33101 Tampere, Finland

ARTICLE INFO

Article history:
Received 29 December 2015
Received in revised form 22 August 2016
Accepted 7 September 2016
Available online 15 September 2016

Keywords:
Code
Code literacy
Metaphors
Education
Programming
Teaching programming
Pedagogy
Media literacy

ABSTRACT

Digital technology has become embedded into our daily lives. Code is at the heart of this technology. The way code is perceived influences the way our everyday interaction with digital technologies is perceived: is it an objective exchange of ones and zeros, or a value-laden power struggle between white male programmers and those who think they are users, when they are, in fact, the product being sold. Understanding the nature of code thus enables the imagination and exploration of the present state and alternative future developments of digital technologies. A wider imagination is especially important for developing basic education so that it provides the capabilities for coping with these developments. Currently, the discussion has been mainly on the technical details of code. We study how to broaden this narrow view in order to support the design of more comprehensive and future-proof education around code and coding. We approach the concept of code through nine different metaphors from the existing literature on systems thinking and organisational studies. The metaphors we use are machine, organism, brain, flux and transformation, culture, political system, psychic prison, instrument of domination and carnival. We describe their epistemological backgrounds and give examples of how code is perceived through each of them. We then use the metaphors in order to suggest different complementary ways that ICT could be taught in schools. The metaphors illustrate different contexts and help to interpret the discussions related to developments in digital technologies such as free software movement, democratization of information and internet of things. They also help to identify the dominant views and the tensions between the views. We propose that the systematic use of metaphors described in this paper would be a useful tool for broadening and structuring the dialogue about teaching children to code.

1. Introduction

Digitality as a phenomenon defines our era. Digital technologies have secured their place in business and in social relations as well as in culture. Digital technologies affect society, but often these changes are taken as given, without broader discussion on the impacts and consequences (König et al., 1985). This is troubling, because digital technology functions in various positions in our society. For example, a high percentage of stock trading is done through trading algorithms with little human involvement. (Washington, 2015; Steiner, 2013). Modern cars carry so much digital technology they have been called "computers on wheels" (Foley Lardner LLP, 2014; Hirsch, 2015). Social media, essentially a digital phenomenon, has defined

* Corresponding author.
E-mail addresses: tomi.dufva@aalto.fi (T. Dufva), mikko.dufva@vtt.fi (M. Dufva).

http://dx.doi.org/10.1016/j.tsc.2016.09.004

new ways of interaction and has influenced culture. There is also evidence that digital technologies shape the way people think, by supporting, sharing and expanding people's cognitive processes(Barzilai and Zohar, 2006). By digital technologies, we mean technologies that are based on digital signal processing, which can be reduced to a flow of ones and zeroes, and which usually utilize information networks to function.Digital technologies allowed for the rampant innovation and growth that started around the 1940s and are defined as the digital age (Ceruzzi, 2012). Digital technologies include all the technologies from smartphones and computers to automated manufacturing and decentralized communication protocols. Digitalization presents new challenges, that, in essence, call for an understanding of digital technologies. The so-called digital divide, that formerly implied the distinction between those who have access to the internet and to those who do not (Mehra, Merkel, & Bishop, 2004) can now be seen as the divide between those who understand digital technologies and those who do not. (For a historical view on ICT in education, see Wilson, Scalise, & Gochyyev, 2015). Mark Warschauer points out that, in today's society, the ability to access, adapt and create knowledge using information and communication technologies is critical to social inclusion (Warschauer, 2004).

The access to digital resources, as well as the ease of use of those resources, has increased, but the understanding of the code has not kept the same pace. This can be seen, for example, within the digital natives discussion. Knowing how to use a tablet computer at the age of two does not mean that one understands the way the machine works or the code behind it. It does not even imply that one could learn to cope with the technology (Kupiainen, 2013). This can also be seen from Carita Kiili's dissertation (Kiili, 2012) where she states that many young adults have problems assessing and evaluating search results in the net. In essence, digital technologies are a source of inequality, which is problematic given their ubiquity in modern society.

Code is the heart of every digital technology and substantially shapes its behaviour. In this paper, we define code as a digital language with a set of assumptions about the users and the world. Code is used to create programs that control digital technologies, from automated factories to personal computers, and from connected home appliances to services providing social networking. Thus, code, in our working definition, refers to the principles and choices made, and is not restricted to any specific programming language. Coding is the act of writing code and building programs, which includes making implicit and explicit choices about the purpose, framing and scope of the program.

The key motivation for this paper is that, because digital technologies are always programmed and are thus based on code, understanding code and the assumptions inherent in it is necessary for full participation in modern society. The code in digital technologies is not value-free, rather it widely reflects both conscious and subliminal values of the programmer, a software company or society's understanding of good code. Digital technology's operating models are not immutable laws of nature, but rather flexible models that are designed and controlled by humans (Lessig, 1999, 2009). Code does not reflect objective truth about the world. Instead, it constructs laws in the digital realm. Without understanding how these laws are formed, we are not able to fully participate in the discourse of our digital life (Giroux, 2011; Lessig, 2009, Rushkoff, 2010). Technology does not impinge upon us from the outside of society, but interweaves into our society in the same way as the political or economic system does, and is also dependent on these other systems, which can alter the way, or speed, of technological progress (König et al., 1985). Without including technology as a coherent part of societal discussion the effects of technology and its relations to other systems stay ambiguous. Furthermore discussion around the ramifications of technologies are crucial as technology has the tendency to convert social, scientific, governmental and human problems into technical problems (Williamson, 2015).

We propose code literacy as a way to participate to the discussion around the effects of digital technologies on society. Code literacy does not directly allude to learning to program in the traditional sense, rather it implies the understanding of the code and its intentions and context. The notion of literacy illustrates the case: In the same way that not all literate individuals become authors, not all code-literate individuals become developers. Still, literate people have the necessary skills and the apprehension of reading and writing.

Understanding code does not emerge naturally from lived experience, but has to be taught. The code used to form the present digital world, be it an operating system, software or stock- trading algorithm, is distinctly different from the everyday analogue tools, such as hammer, pen or paintbrush, used to form the material world. One example of this is the binary system of two alternate states, often represented as 1 and 0. Code is binary and, therefore, can be reduced to "yes or no" decisions. However, as Rushkoff argues, human lives are not binary and thus trying to represent them using these binary systems is problematic (Rushkoff, 2010).

Learning to code and digital learning systems are deeply intertwined in political, societal and commercial structures (Williamson, 2015, 2016). We argue that current teaching about digital technologies, programming and code and the discussion around it does not take fully into account the societal and ethical dimensions of code. Thus, our goal in this paper is to broaden the discussion and propose a structure for understanding different views on code. To facilitate this, we describe nine metaphors of code based on four paradigms. Through the use of metaphors and their associated paradigms we wish to support a larger and more holistic view on code and digital technologies.

This paper is structured as follows. After this introduction, in Section 2 we describe nine general metaphors that cover four common paradigms of social theory as well as different assumptions about the complexity of the world and the relations between stakeholders. In Section 3, we apply these metaphors to structuring the discussion around code and illustrating various viewpoints expressed about what code is and how it influences society. In Section 4, we focus specifically on education around code and coding, and suggest different views on teaching code. Section 5 concludes the paper.

<table>
<tr><td></td><td></td><td colspan="3">Assumptions about the values and interests of stakeholders</td></tr>
<tr><td></td><td></td><td>Unitary</td><td>Pluralist</td><td>Coercive</td></tr>
<tr><td rowspan="2">Assumptions about the nature of the world</td><td>Simple</td><td>Machine
Organism</td><td rowspan="2">Culture
Political system</td><td>Psychic prison
Instrument of domination</td></tr>
<tr><td>Complex</td><td>Brain
Flux and transformation</td><td>Carnival</td></tr>
</table>

Fig. 1. Nine metaphors categorised by their assumption of the complexity of the context or "system", and the values and interests of stakeholders (Jackson, 2003).

2. Metaphors for structuring the discussion around code

The language around concepts such as technology has been analysed before through methods such as discourse analysis and critical discourse analysis (Fairclough, 1995; Weiss and Wodak, 2006). Our analysis is based on this stream of qualitative analysis of the concepts used to describe a phenomenon. However, in this paper we use metaphors as the tool for analysing and structuring the discussion. Metaphors are a mechanism for describing, understanding and comparing abstract concepts, and can be defined as mappings across conceptual domains (Lakoff, 2009). Through a metaphor, the entities in one domain are mapped onto entities in another domain. For example, a segment of code could be mapped to represent an organ in the human body. Metaphors can be powerful in influencing how an issue is approached or a problem is framed, but we are mostly unaware of their effect (Thibodeau and Boroditsky, 2011).

Metaphors have been used in a systematic fashion in management and organisational studies (Jackson, 2007; Morgan, 2006). We use the metaphors introduced by Morgan (Morgan, 2006) and developed further by Jackson (Jackson and Keys, 1984). These nine metaphors describe different views on the concept of code and include the metaphors of machine, organism, brain, flux and transformation, culture, political system, psychic prison, instrument of domination and carnival. The nine metaphors are based on four common research approaches or paradigms in social theory: the functionalist, interpretive, emancipatory and postmodern (Jackson and Keys, 1984; Jackson, 2007) based on (Louis, Burrell, & Morgan, 1983) and (Alvesson and Deetz, 1996).

Paradigm, in its original sense, means the set of ideas, assumptions and beliefs that shape and guide the scientific activity of a research community (Kuhn, 1970). The aim in the functionalist paradigm is to demonstrate law-like relations between objects. The emphasis is on function and efficiency. The functionalist paradigm is based on the assumption that an understanding can be gained through scientific method and empirical research. The interpretive paradigm, as the name suggests, is more interested in the interpretations people make of different issues and situations. These interpretations guide people's behaviour. Thus, the aim is to understand these interpretations and the underlying culture through methods such as hermeneutics and ethnography. The emancipatory paradigm focuses on the power relations in society. It is aimed at "emancipating", i.e. liberating and empowering people and unmasking domination through ideological and cultural critique. The postmodern paradigm is opposed to all three former paradigms, which it views as modernist. It critiques the attempt to form grand narratives and assuming rationality and direction. Its methods include deconstruction and genealogy.

The metaphors can be structured along two dimensions (Jackson, 2003). The first considers the assumptions made about the world. The world can be seen as relative simple, meaning that the key issues are knowable, causal relations between the issues are straightforward and known, and goals are achievable by following a detailed plan. On the other hand, the world can be seen to be a complex, interconnected "mess", where there are many surprises, unintended consequences, non-linear causal relations and, thus, the focus is more on adapting and "muddling through" than following a plan.

The second dimension covers three different perceptions of the values and interests of the stakeholders: unitary, pluralist and coercive. Stakeholder values and opinions can be assumed to be unitary, meaning that the stakeholders tend to agree on a common goal and share a similar worldview. A pluralist view criticises this as too simplistic, and assumes that there are multiple, competing goals and worldviews. A coercive view goes further and frames the stakeholder relations as a power struggle between those in power and those who are oppressed. Thus, there are multiple goals and worldviews, but not all are given voice.

The metaphors can be positioned to a matrix using these two dimensions. (Fig. 1, see also the system of system methodologies by Jackson & Keys (1984) (Jackson, 2003). While Jackson (2003) uses metaphors to describe organisations, we argue that they can be used also to shed light on more general issues. We will next briefly describe the metaphors and then, in Section 3, use them to illustrate various views of code.

The first four metaphors are based on the functionalist paradigm and view the values and interests of stakeholders, i.e. people who are influenced by code, as unitary and thus not problematic. The machine metaphor depicts issues as linear, mechanistic sequences from inputs to outputs and emphasises efficiency above all. The organism metaphor describes a

Table 1
Nine metaphors for understanding the nature and purpose of code.

Metaphor	Description of code	Purpose of code	Example
Machine	Code is a linear sequence of commands that is input to a machine	To control a machine	Algorithms, code listings
Organism	Code is a set of objects that represent different parts of a program	To create functionality, to interact	Object-oriented programming
Brain	Code is the intelligence of man-made systems	To create new information, to learn	Cloud computing, artificial intelligence
Flux and transformation	Code is the process that creates changes in man-made systems	To create change, to create structure	Software as life changer
Culture	Code is a way of thinking and understanding the world	To connect and create a community	Free software foundation, Hacker ethcis, Hacker Culture
Political system	Code is a statement and a tool to shape the world	To establish a new form of society	Code as political construct. Internet
Psychic prison	Code is a system which requires people to adapt to it	To shape people	Filter bubble
Instrument of domination	Code is a tool for domination	To control people	Data as a source of power
Carnival	Code is a tool for art and creativity	To challenge existing mindsets, to open up discussion	Creative coding.

non-linear interaction between different parts and highlights the functional differences and roles of the parts. The brain metaphor, stemming from cybernetics, puts emphasis on learning and adaptation in a hierarchical system, while the flux and transformation focuses on the processes and logics of change.

The culture and political system metaphors are based on the interpretive paradigm, which puts emphasis on the different interpretations that exist of an issue. The culture metaphor focuses on values, beliefs and worldviews, and thus highlights the community or communities around the issue. The political system metaphor also emphasises values and worldviews, but focuses more on the governance and decision-making around the issue. It thus highlights relevant institutions and political structures.

The psychic prison and instrument of domination metaphors are based on the emancipatory paradigm. Similar to the interpretive paradigm, the assumption is that there are multiple differing worldviews, beliefs and values. However, now the focus is on the power relations between the worldviews and on bringing ignored or suppressed aspects and questions to the surface. The psychic prison metaphor focuses on the structures, both intentional and unintentional, that suppress individual freedom and learning. The instrument of domination metaphor focuses more on the group level and highlights how the issue is used as a way to control others.

The final metaphor, carnival, is based on the postmodern paradigm, which seeks to question the way the issues are discussed and framed in general by deconstructing the main concepts. The carnival metaphor thus highlights the creative and chaotic side of an issue, in order to use the issue itself to question the way it is discussed. This may often result in a multi-faceted picture of the issue, which is not as coherent as in the other metaphors.

Our purpose in describing and applying these metaphors is not to argue that one is better than the other, or that a certain view to an issue should be followed. Rather, our purpose is to use the metaphors to structure the discussion around code. The nine different views help to understand the discussions and decisions around code. In addition to giving a more comprehensive view of what code is, the different metaphors also highlight what is missing from the discussions and which views conflict with each other. We will return to these questions in the discussion section, after we have applied the nine metaphors in the next section.

3. Understanding code through metaphors

In this chapter, we propose ways to define code through the different metaphors. We illustrate how code is defined and how it appears in the different metaphors. In Table 1, we provide a summary of these descriptions of code, views of the purpose of code as well as some examples. These results are elaborated below.

3.1. Functionalist paradigm

The functionalist paradigm introduces a mechanical and unitary view of code. It focuses on the straightforward advancement of code as a technical invention. Inside the paradigm, four different metaphors present different nuances. As a whole, the functionalist paradigm can be marked as a dominant view: It predominantly acts as a common and shared understanding of the meaning of code.

3.1.1. Machine: code as a mechanistic, linear sequence of commands

The machine metaphor represents the fundamental mechanical comprehension of code. Code is seen as a sequential set of instructions that are input into and processed by a machine: the computer. The results are then displayed to the user. In other words, the user expects that the computer as a machine will deliver her or him results based on a set of instructions – the code.

From a technical perspective, the machine metaphor demonstrates the fundamental physics of code. Paul E. Ceruzzi calls this the digital paradigm – that all code, computation and control are done in binary form. With binary form, he not only refers to a binary arithmetic – the number system that uses just two symbols, 1 and 0–but also to the use of binary logic that is used to control, encode and transmit the information (Ceruzzi, 2012). In essence, all digital information is based on the binary code.

In the machine metaphor, computers, the machines that are able to process digital information, are basically input and output machines. They take instructions, process those instructions and output information based on the instructions. Code represents the set of instructions in the languages that the computers can understand. Computer languages vary from lower level languages to higher level languages. Lower level languages are closer to the binary logic that computers use on the implementation level, while more complex, higher level languages are easier for humans to write and read. Whatever the language is, in the end all of these languages are compiled back to a binary form.

From the machine-metaphor view, the higher level languages can be seen as rational progression towards getting the intended process completed faster and easier. Even though the code in higher level languages is farther from the binary code, being closer to the language humans use increases efficiency through a manageable working environment and less friction in the process. Many modern compilers are generally more efficient in compacting the code to binary than are humans, resulting in a more robust code (Ceruzzi, 2012). Machine metaphor illustrates the straightforward process of digital technology – progress means creating ever more efficient machines to interpret increasingly complex code.

The machine metaphor represents a reductionist viewpoint and a hierarchical way of processing data. Tasks are broken into parts and processed in a strict order governed by the rules of the program – the code. This assumes that the context is simple and can be reduced to separate parts, and that a single common goal exists. Seeing code only through this metaphor results in an emphasis on the process without questioning the direction, which, furthermore, often results in advocacy of a single way of coding without embracing possible diversity of goals and processes.

In the context of planning education, this could mean a debate on which coding language should be taught, but not questioning what the purpose of teaching the coding language is in the first place. The underlying rationale behind such a debate is that coding is a skill for the job market and teaching coding – the right language and style – is thus good for ensuring the employability of future workforce.

3.1.2. Organism: code as a combination of objects

The organism metaphor sees the code as a construct of many individual parts that work together. This can be seen as a continuation of the machine metaphor, as it focuses further on increasing the efficiency of code by further breaking the code into more manageable parts, thus allowing programmers easier ways to reaching their goals (Petzold, 1999). The organism metaphor represents another common mechanical view of the code. It can also give us an idea of how modern code is created and how software problems are addressed – code is not seen as a simple set of instructions but as a structured sets of code, organs, that together create a working program, or a body.

On a technical level, the organism metaphor corresponds to object-oriented programming (Cox, 1985). Object oriented programming breaks the linear set of instructions to different objects that can be addressed when necessary. Most modern programming languages favour this approach as it allows for a more structured management of complex code that makes problem solving easier, thus increasing efficiency (Petzold, 1999).

Furthermore, the organism metaphor represents a structural approach, which allows the creation of more flexible code that can interact simultaneously to multiple inputs and outputs. Coding is still seen as a mechanic practice of giving instructions, but the linearity of the instructions is broken into interconnected parts. Object-oriented thinking and problem solving are at the heart of modern coding. Many commonly used higher level programming languages incorporate object-oriented thinking. As such, object-oriented thinking and problem solving break the traditional narrative and sequential ways of thinking and understanding (Manovich, 1999).

3.1.3. Brain: code is intelligence

In the brain, metaphor code is not only sets of organized instructions, but represents the intelligence of computers. Code is seen as the man-made brain: intelligence that not only structures information, but also creates and modifies it. Code is the central unit that processes and develops information in the system, be it software, computer or any other machine.

One example of seeing code through the brain metaphor is the notion of artificial intelligence. Artificial Intelligence (AI) is the study of how to build or program computers to enable them to do what minds can do (Boden, 1996). The idea of artificial intelligence has captivated many past and present thinkers long before digital technologies existed (McCorduck, 2004). Modern programmable computers can be seen as the manifestation of the idea of artificial intelligence – before computers, machines were built for a specific task and purpose (Ceruzzi, 2012). The idea of a general device, the purpose of which could be changed indefinitely by programming, was revolutionary. A similar idea of programming and reprogramming fuels the current developments in artificial intelligence – pattern recognition, computational learning theory and machine learning

stem from the idea that the code inside the computer can change, or, loaning a biological term, it can evolve (Chrisley and Begeer, 2000). The ultimate extreme in artificial intelligence is technological singularity in which artificial intelligence has progressed beyond human intelligence and becomes sentient through code (Kurzweil, 2005; Lanier, 2010). Through the brain metaphor, this development is seen as natural and desirable; the metaphor contains no problematization or critique. Code only actualizes the potential and predetermined ultimate goal of digitality. In technological singularity, code truly becomes the brains of the computer.

The brain metaphor is naturally not limited to the discussion of artificial intelligence. We can also look at other systems of code through the brain metaphor. It extends the functionalist paradigm further, from lists and objects to a system with a central controller who has the authority to control and modify the code. A good example is cloud computing, where the machines running the code become secondary. Even though the code is running on physical computers, the physical location is irrelevant. Code is seen to escape the hardware and have a life of its own in the cloud of digital computing power. In a similar way, modern digital voice-controlled assistants aim to create the illusion of an omniscient virtual entity and can thus be seen to represent code in its abstract form. They seem to exist beyond the machinery running them.

3.1.4. Flux and transformation: code will save the world

The Flux & Transformation metaphor is similar to the brain metaphor, as it also concentrates on the development of the code, but, rather than framing code as the intelligence of machines, it sees code as a transformative tool to continually change the world. It therefore broadens the focus from computers and code to their environment. It can bring into focus the aspiration many software companies share, at least in their public declarations, which is not just to create better products, but to make the world a better place. From Google's "Do no evil"-slogan to Facebook's CEO Mark Zuckerberg, who argues that his company's mission is to "make the world more open and connected" (Mark Zuckerberg, Sarah Lacy Interview Video, 2008), software companies are focusing on solving problems rather than creating products. As Jeff Jarvis has said, "Complexity is a solvable problem in the right hands" (Jarvis, 2012).

Code is seen as a medium that is both flexible and can be deployed rapidly and widely. It only takes one person and a few nights to come up with a solution that has the possibility to change or disrupt the way we see the world. The Flux and Transformation metaphor thus moves the focus from the advancement of efficient code to code's ability to advance our lives. The metaphor is firmly grounded in the functionalist paradigm, and focuses on how to create a change rather than on the question of why change is needed, what the direction should be and who gets to decide the direction. Thus, it does not problematize the act of making the world a better place. The problems are seen as simple, straightforward tasks that can be solved with code.

3.2. Interpretive paradigm

Whereas the functional paradigm and the last four metaphors saw the code as a fairly straightforward issue that mainly concerns technical aspects and implementations, the interpretive paradigm has greater interest in the different ways of seeing and understanding code. In contrast to the unitary perspective of functionalism, the interpretive paradigm takes into account the plurality of stakeholder values and opinions in the context in which the code is created and deployed.

3.2.1. Culture: code creating communities

The Culture metaphor focuses on the communal aspects of code, for example on what kind of communities and subcultures are formed around code and coding, and what kinds of values are projected to code. The popularisation of digital technology has led to a whole industry that has created its ways of working and communicating as well as its ethical rules, which are reflected in the way code is perceived and treated. The culture is not unambiguous; rather it consists of many sub-cultures and ideologies.

The Culture metaphor brings into focus the ways code affects how the surrounding environment – the world – is interpreted. One example of this is the free software movement. The movement has a long creation history dating back to the early phases of computers. Before personal computers, computers were mainly used in corporations, universities and research laboratories. Most of the operating systems were open. Anyone could read and modify the way operating systems worked. When the industry began to grow, especially into businesses and households, and the operating systems evolved, many manufacturers started closing their code, thus preventing collaboration and modification. For some, this development went against their basic rights and values as programmers. On this basis, Richard Stallman, then working for the Artificial Intelligence Lab at MIT (Stallman, Gay, & Lessig, 2009), created the GNU project (Fsf, 2015a), on which Linus Torvalds later built his free operating system, Linux. A few years after starting the GNU project, Stallman founded the Free Software Foundation (FSF) (Fsf, 2015b).

These projects can be seen as a wish to maintain the academic ethos and collaboration as well as the hacker culture alive in the developer culture (Stallman et al., 2009). The stated goal for these projects is societal change. FSF wants to change the way we use, distribute and think about code. At the core of FSF are four rights that, according to FSF, are essential in keeping the development and use of code democratic:

> The freedom to run the program as you wish, for any purpose (freedom 0).

> The freedom to study how the program works, and change it, so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
>
> The freedom to redistribute copies so you can help your neighbor (freedom 2).
>
> The freedom to distribute copies of your modified versions to others (freedom 3). By doing this, you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this (Fsf, 2015a).

These rights align with hacker culture, which, at the time Stallman founded the foundation had different connotations than the word "hacker" has now. Hacker was a positive concept rather than depicting a coder with a criminal aptitude. Hacker culture believes in free access, freedom of information, and improvement to the quality of life (by using digital technologies) (Levy, 2010). Even though the aims of FSF are political and ideological, it also reveals the richness in the developer culture, with its core beliefs, tradition and ethics. As Coleman (2012) says in her book Coding Freedom, The ethics and aesthetics of hacking (Gillen, 2013), there is a common pride and joy in offering your "handmade" code to others, as well as the genuine interest in learning from other developers' code.

As the examples above illustrate, inspecting coding from the perspective of the cultural metaphor reveals the rich and many-sided culture of code and reveals that coders sit simultaneously at the centre and at the margins of the liberal tradition (Gillen, 2013). Code both creates many sub-cultures and at the same time affects the general culture. Thus, code and coding is not only about giving instructions to a machine or solving problems, but also about influencing the culture.

3.2.2. *Political system: code structuring the society*

The other metaphor in the interpretive paradigm, political system, offers a somewhat different view from the culture metaphor. Whereas the culture metaphor sees the world from the individual and grassroots perspective, the political system metaphor takes a look at how code creates hierarchical systems that affect our everyday lives. Besides influencing its culture, code also affects society in a more systematic manner. The way our coded environments are built, as well as the way code itself is built, constructs the ways we act in the world. From operating systems and programs to protocols that hold the constructions together, the many ways we interact in the world are channelled through the code. *"Code is law"*, as Harvard lawyer and author of Code and other laws of cyberspace (Lessig, 1999, 2009) Lawrence Lessig puts it. In the political system, metaphor code is seen as not mere mechanical technology, but a malleable force that can be changed by the culture that developers live in, as well through governmental or any other institutional control. One example of this is the internet, as it offers us a multi-faceted view of how political systems affect the way code is structured. Born out of research projects in the US defence department, the internet spread to universities and from there to the public. In the beginning, the internet was seen as a revolutionary medium that allowed every participant to not only receive, but to send information (Lessig, 2009), thus enabling a 'real' democratic process. The internet was seen as free by its nature, offering equal opportunities to everyone (Fleischer et al., 2014, Lessig, 2009). A quote from MIT professor Dave Clark's 1992 speech at the IETF (Internet Engineering Task Force) conference depicts the ethos well: *"We reject: kings, presidents, and voting. We e believe in: rough consensus and running code."* (Borsook, 1995) But as Lessig wrote already in 1999, The internet has no nature per se, but is dependent on our choices:

> 'We can build, or architect, or code cyberspace to protect values that we believe are fundamental, or we can build, or architect or code cyberspace to allow those to disappear. . . . .There is no choice that does not include some kind of building. Code is never found; it is only ever made, and only ever made by us' (Lessig, 1999).

Sixteen years later, the structure of the internet has been changed considerably through the actions of several different sources. When Lessig was writing the first revision of the Code and other laws of cyberspace, the current topic was free mp3-downloads and the music industry's reaction to it, leading to digital rights management (DRM) and legislation. At about the same time, China was waking up to the threats that the internet, as a source of non- controlled information might impose to its governance, causing it to erect the "Great Firewall of China", a project that aims to manage all the net communication in and out of China (University of California-Davis, 2007). And a few years ago Edward Snowden revealed the widespread internet surveillance that governments were engaged in, thus displaying yet another layer of the internet and what has been made possible through code. As Mikael Brunila proposes, the internet has enabled panspectric control, which alludes to the way information can be gathered from the internet. In traditional panoptic control, information is gathered from the suspects after they actually become suspects; in panspectric control, everything is collected, all the time, and from everyone (Fleischer et al., 2014).

These kind of structural changes in societal architecture give us a glimpse of the reach code has. The internet is a multi-layered construction of code, which is inherently intertwined with political systems. Code is not free from these ties, but rather has a decisive role in creating the architectures we use every day. The questions of how to control code, who can control code and why would we control it are increasingly more relevant in our lives, as code permeates more and more of our everyday activities via the internet-based services, but also through increasingly "smart" gadgets.

3.3. Emancipatory paradigm

Many of the issues that arise in the interpretive paradigm can also be seen as issues in the emancipatory paradigm, and vice versa. The difference comes from the focus on power relations. In the interpretive paradigm, there are differing views on the purpose and goals related to code, but the differences between these views are assumed to be somewhat unproblematic. We can examine the different views that code offers to culture and politics. In contrast, the metaphors in the emancipatory paradigm focus more on what the power relationship is between these various views, and how these power relations are reflected or enacted through code. For example, does code enable or restrict emancipation both at the individual and societal level?

3.3.1. Psychic prison: code restricting human behaviour

The psychic prison metaphor takes a look at the power relations from the individual perspective. It brings into focus the code that underlies technological inventions from the emancipatory perspective. Is a code good for an individual? Does this code help an individual accomplish the things she wants to do? How does the architecture of code influence the life of an individual? One example of this is what Eli Pariser calls the filter bubble (Pariser, 2012), meaning the possible outcome that may result from using invisible automatic personalisation algorithms. The algorithms are invisible in the sense that an individual does not choose to use them, nor sees them. Rather, she has opted into them automatically when using certain services. One example Pariser gives is the difference in results people get by doing the same Google search. Using the same search words yields different results, based on dozens of different signals Google collects from the user. (Pariser, 2012) A quote from Mark Zuckerberg, CEO of Facebook illustrates the idea further:

> "A squirrel dying in your front yard may be more relevant to your interests right now than people dying in Africa."

As Parisier says

> "Your filter bubble is your own personal, unique universe of information that you live in online. And what's in your filter bubble depends on who you are, and it depends on what you do. But the thing is that you don't decide what gets in. And more importantly, you don't actually see what gets edited out. "

The idea of the filter bubble shows the possible problems caused by code that is selecting content from the internet unbeknown to the user. Having no control over this code creates an unequal situation between the user and the code. On what basis does the code select what content is shown and what is hidden? What are the bases of the code selecting the showable content? And what are the motivations of the developer who decided these rules embedded in the code? Are the rules decided with the user's assumed benefit in mind, or are they defined to benefit the business that the developer is in?

On a more abstract level, the psychic prison metaphor also focuses on the issue of how we might knowingly or unconsciously change ourselves because of code. For example, MIT professor Sherry Turkle talks about the ways we require digital devices to actualize our feelings. She gives an example of her study where she concluded that some teenagers require the passing of text messages to truly justify and experience their feelings, like falling in love or being scared (Turkle, 2011). Another point Turkle, along with many others such as Jaron Lanier (Lanier, 2010) and Douglas Rushkoff (Rushkoff, 2010, 2013) bring up, is the alienation that code allows us to feel. Turkle speaks about the feeling of "alone together" where we are physically in one place with other people, but mentally somewhere else (Turkle, 2011). Another example of this abstract level is obsessive gaming. How does the code in the games take into account the player and their needs? Is the code made in a responsible way or does it use tricks to hook the player into spending more time or money on the game?

The psychic prison metaphor highlights how the power relationship between individual and code is problematic. The ways code changes us may not always be for the good. As Jaron Lanier asks, do coded environments change people, or do people change themselves because of them? Lanier's point is that, in order to use, enjoy or respect code, humans can adjust to many levels of intelligence. Sometimes, code requires us to be less intelligent than we really are (Lanier, 2010). Self-control is required in order to break free from the psychic prison. Both Lanier and Turkle use the term dieting. In a similar vein, Parisier is concerned that the filter bubble might feed us too much of the information we enjoy and too little of the information we need, and uses the term "information junk food" (Pariser, 2012). Turkle asks for a digital diet: a reflective and introspective review of what and how we want to use our devices (Turkle, 2011). The psychic prison metaphor enables the exploration of the ways code might limit or shape the current and future potential of humans.

3.3.2. Instrument of domination: knowledge and control of code is power

The instrument of domination metaphor focuses on the power relations between societal and communal constructs and code. Code is seen as a force that is used intentionally in order to shape and control others. The metaphor concentrates on those aspects of code that may enable some group to dominate another group in ways that might not have been possible or feasible before. In other words, does the architecture of code have an aptitude to cause inequality? If that is the case, then those who understand and have access to code have more power than those that do not. Because of the widespread nature of code, these issues are not just marginal questions. Code is not just at the heart of computer screens or smart phones, but affects a wide variety of things from pacemakers to cars and manufacturing units, offering unforeseen access to the everyday lives of humans.

For example, if computer browsers can transfer so much information to Google that it can confidently personalise our search results, how much more does the mobile phone with its sensors and location data add to this information? Or, what about our payment data collected from credit card purchases and ewallets and the increasing popularisation of the internet of things? If all the data from house temperature and the efficiency of a person' habits of recycling to their history of payments are funnelled to one or a few institutions or corporations, does it not create the possibilities for domination? In a similar way, the invisibility of code in the filter bubble creates problematic situations, as does the invisible and closed collection of data to both individuals and to society as a whole (Morozov, 2013). Collection of data is problematic because of the lack of democratic availability of the data. Most of the information collection is done by large tech companies that keep the information to themselves or only sell it to other businesses (Fleischer et al., 2014).

The problematics of domination are not just limited between tech companies and users, but the relationship can be seen in several different scenarios. When more devices get both transformed into code and connected to networks, new opportunities arise for misuse. For example, modern cars can be thought to be computer servers on wheels (Vallance, 2015), and when they get connected to outside networks they can also be hacked and remotely controlled, as two new studies demonstrate (Checkoway et al., 2015; Vallance, 2015). Being able to take almost full control of any network-connected car from the comfort of your sofa, using just your computer and mobile phone exemplifies the significance of domination by code very well.

Other more well-known examples are the privacy breaches that Edward Snowden revealed. The widespread nature of how governments spy on citizens illustrates the reach that digital devices and code have in our lives. Without acknowledgment, we are giving up information about our lives that we did not even know about before. One important angle on the massive data collection is that it is impossible to collect or manage that amount of information without code, thus increasing the dependency we have on code. The increase is not just in the pure processing power, but even more in the capabilities of evaluation of the information. Also, this processing power is more reachable by those that have more assets and time, creating an imbalance that is further increased by the lock-in effects, common in digital technologies (Lanier, 2010) (Morozov, 2013) (Rushkoff, 2010). The imbalance is further increased by the prevalent proprietary nature of the code (Stallman et al., 2009; Vaden, 2005).

Yet, even if code allows for new kinds of domination, and may be biased towards those who have more assets, it does also enable rebelling against those currently in power. The construction of code allows for clever individuals to use it for their own purposes. For example, hackers in China or in the Arab world during the Arab spring or in other countries that suppress freedom of speech can benefit from code architecture by tunnelling messages securely to the outside world, passing governmental restriction and walls. In the instrument of domination metaphor, code can be seen as architecture that allows more multi-layered ways of domination, and is both the instrument and the product of power relations.

3.4. Postmodern paradigm

Functionalist, interpretive and emancipatory paradigms provide different views of what code is. The postmodern paradigm provides a "meta-view" and focuses on the mechanisms through which we create these views. Essential questions in this paradigm are how do we see code, what influences our perception of code and what other ways could there be? The emphasis is thus on deconstructing the process of giving meaning to what code is.

3.4.1. Carnival: understanding of code can be created through creative use of code

To illustrate how the concept of code can be approached in the postmodern paradigm, we employ the metaphor of a carnival. In the carnival metaphor, many perceptions can exist at the same time and playfulness, suspension of disbelief and multi-facetedness is embraced. The carnival metaphor focuses on the creative and artistic sides of code. It illustrates how code can inspire people and evoke various emotions. It also helps to explore the different reactions people have expressed towards code. However, the carnival metaphor does not fully reflect all the aspects of the postmodern paradigm and the endeavour to deconstruct the meaning and sense of code. Art and creativity can be seen as ways of deconstruction but they are not the only ways to do this, nor can we say that they are only views into the multiple nature of postmodern. Jackson (2007) uses also the metaphor of broken mirror to reflect the change from one solid picture into various differenting pictures of the whole. A good example of the understanding of code in the carnival metaphor is creative coding, which concentrates on the expressive rather than functional sides of code. Creative coding has its origins in the 1960s, when artists first began to experiment with computers. In recent decades, creative coding has seen an upheaval along with several tools aimed at the creative professionals.

> "Creative code may sound like an oxymoron, but as in many technical processes in the art studio, creativity may emerge once rules are learned and then broken (Knochel & Patton, 2015)."

Creative coding allows artists to question and critique code and, at the same time, express themselves through code. In a similar way that a brush or a pen is a tool for visual artist, code can be seen and used as an artistic instrument. Code, like any instrument has its own biases and ways of working, creating a medium that allows things to be expressed in unique ways. As Cox says in his book Speaking Code: "Code, like language in general, evokes complex processes by which multiple voices can be expressed, modified, and further developed" (Cox, 2013, p.6)

One example of creative coding is "Smile TV", a project by David Hedberg. "Smile TV" is a simple TV-set, but it only works when the viewer is smiling, thus creating a real working product using modern technologies and at the same time critiquing digital culture (Scholz, 2014). The works in creative code are diverse, where some focus on the visual effects or on visualisation of data, such as Jer Thorp's works (Thorp, 2009). And some use digital technologies to reveal hidden layers in these techniques, such as the Immaterials project that materialises the existence of GPS-signals (Arnall, 2014) and Wifi signals (Arnall, 2011).

As the examples indicate, creative coding comments on the views of code expressed within multiple paradigms and metaphors. Whereas some works can take a functionalist angle and use code in an almost similar way when developing "working" software, some may misuse and break the workings of code altogether. And still others may use code as a way to critique the power issues arising from the code. As such, the world around creative code is ambiguous and multi-faceted.

Creative coding illustrates how the carnival metaphor incorporates various views captured in other metaphors, joins them together and deconstructs them. Like many art works, the carnival metaphor focuses more on the experience than the theory. The art created does not justify its presence, but rather waits to be experienced. As such, it can show us those sides of code that may not be otherwise understood, or seen.

In this section, we have described different perceptions of code through the use of nine metaphors. In order to illustrate how these metaphors can be used to structure and inform a topical issue, we apply them to the ongoing discussion about teaching programming in schools.

4. Applying the metaphors of code to developing education around code and coding

Teaching programming has lately been a much discussed subject in education. Finland along with many countries, such as Estonia, the UK and the US have started or are starting to incorporate programming in the basic curriculum in schools (Halinen, 2014; Sterling, 2015). Our research is mainly focused on the discussion, decisions and development of teaching programming in Finland, although it can be seen to echo similar tendencies in other countries such as UK (For example see Williamson, 2015). When the teaching of programming moves from the level of higher education to the level of basic education, the understanding of programming becomes increasingly important: does the basic curriculum just prepare younger students for the digital industry as a possible workforce, or does it offer educational views on the complex issues around widespread digital technology? This problematic is cumulative, as teachers are often unclear of the intended aims and goals of teaching programming (Pollari, 2014). The discussion around code is often limited to methods of teaching programming, such as different platforms etc., and to which programming language would be best in programming. In some cases, code is also seen as part of art and craft, such as in Finland, where teaching programming is going to be divided between maths and craft lessons (Opetushallitus, 2014).

In general, the views around teaching code are fairly limited and mechanical. Even though critique towards technological determinism has been expressed, the idea that technology acts as independent and often objective force is still often taken as granted. (König et al., 1985). Understanding the way code structures our daily interaction with machines and how it mediates our interaction with fellow humans (through digital services) is rarely seen as an essential societal skill. Rather, the code underlying the interfaces and services we use is taken as given. This limits students' capability to identify and question the implicit assumptions about this code. From the stance of critical pedagogy, Paulo Freire asked even in the 1990s to find a policy on teaching technology (Freire, Freire, & De Oliveira, 2014). He acknowledged the increasing speed that technologies advance and how this creates life changes, and asks for "*the quality of getting or creating ability to answer to different challenges with the same speed that things change. This is one of the demands of contemporary education. We need to form and not to train.*" (Freire et al., 2014).

In the previous section we applied nine metaphors to illustrate different perceptions of code and highlight various issues related to these perceptions. We now apply these metaphors to structure and broaden the discussion around teaching programming at the level of basic education. The most prevalent question that arises from applying the metaphors is about the objectivity of code and programming. Is code seen as an objective exchange of ones and zeroes, or is it a value-laden power struggle between white male programmers and those who think they are users when they are, in fact, the product being sold?

The current dominant discussion emphasises more the objective, logical and mathematical sides of code as described by the functionalist paradigm and especially by machine and organism metaphors. Code is seen as an unproblematic language to be taught in order for the students to have a more secure employment. In the context of planning education, this could mean a debate on which coding language should be taught, but not questioning what the purpose of teaching the coding language is in the first place. The underlying rationale behind such a debate is that coding is a skill for the job market and teaching coding – the right language and style – is thus good for ensuring the employability of future workforce. The endeavour to improve education on learning to code can be seen as a large campaign where both political and economical actors lobby their interest through boundary organisations (Williamson, 2015).[1]

[1] Williamson's research is focused on the "learning to code" endeavour in the UK, but there are similarities with the developments taken towards including coding to the basic curriculum in Finland (Saariketo, 2015).

But if we assume that the world around us is more complex, this perception of code does not hold. The brain and the flux & transformation metaphors move the focus from the mechanical viewpoint and put emphasis on the intelligence of code. Code is not a simple language to be learned in order to ensure employment, but rather a complex man-made tool for shaping the world. In other words, code is seen as an instrument that creates and changes our everyday behaviour and practices. Artificial intelligence, as well as the solutionist attitude of many software firms, show the possibilities and reach code has. Code is everywhere in our lives. From this standpoint, merely choosing a programming language to be taught or creating basic logical understanding might not be enough.

When learning and teaching code is understood more broadly, code can be more easily connected to real life situations. Thus students can have a more direct experience of the implications of the code. This can enable discussion in the classroom about the role of code in our society – a crucial discussion but one where there are no right answers. Here Freire's idea of forming rather than training students becomes more clear. Freire sees that education has the responsibility to create digital minds. Training students to learn a programming language is not enough, as it does not form the students to understand the full reach of digital technologies, thus preventing them from creating knowledge themselves, i.e. possessing a critical mind (Freire et al., 2014).

The ubiquitous nature of code leads to the question of whether we agree on how good or beneficial code is today? And furthermore, what do we mean by good or beneficial? These questions are essentially intertwined with public education's aims to help students not only to live in society but to understand societal structures and ethics, and also to question them. The interpretive paradigm focuses on these questions and the way code influences society and culture. The culture metaphor affixes code to its cultural context, offering views on the different mindsets, ideologies and trends that influence the code. The culture metaphor explains the societal, cultural and subcultural contexts that affect the ways code is written, offering us ways to better experience the reasons why code exists the way it does. For example, understanding the ways free software, open source software and proprietary software differ from each other can offer ways to impact software development as well as to offer an understanding of the design choices in the software. Furthermore, the cultural metaphor can offer views of the historical context of code and digital technologies. Understanding the beginning of digitality, such as Babbage's machine, Leibniz's binary logic, or Ada Lovelace, the first computer programmer, might offer valuable connections that increase the student's personal understanding of code.

The metaphor of political system approaches much of the same area as the culture metaphor, but more from the societal standpoint. It addresses critical questions of the purposes and morals of code: What part does code play in the democratic system? The political system metaphor offers ways to approach subjects such as privacy, whistle-blowers, free software ideology or the structure and politics of the internet. It can also be expanded to the philosophies and history of technological invention, and to a discussion about technological determinism.Possible questions to be raised in this metaphor include how technology changes society, what are the relations between technology and society and does society or other aspects of society, such as political decisions or economical forces shape the way the code we use today is made? Ars Industrialis manifestos by French philosopher Bernard Stiegler might offer interesting starting points for classroom discussions about the role of code in society as they contrast technology's role starkly as *pharmacon:* both the drug and remedy (Stiegler, 2005, 2010). The metaphor of code as a political system also offers more reflective viewpoints on the future of code, which might offer interesting talking points when contrasted with brain or flux and transformation metaphors.

The interpretive paradigm emphasises the various perceptions about the background and the context for the code we use every day. This information can be beneficial for teachers as well as students to increase their understanding of the reach that code has. It can offer practical discussions on the reasons and implications of the software we use every day. It also offers the idea that code is not a fixed thing, but a malleable invention, which is affected by the coders, the culture around it as well as societal decisions and politics. This kind of critical understanding might be what Freire calls forming instead of training.

The emancipatory paradigm further increases the humanistic viewpoints on the code. Code is seen not only as mechanical or societal, but as a force that has the power to affect and influence our lives. It questions the intentions of the code as well as our position in the coded world: Do people have the power to decide, or are they being manipulated? Is code made to be truly helpful for users, or is it created for the benefit of the coder or the company? The psychic prison metaphor considers these questions from the individual standpoint and the instrument of domination metaphor deals with the power struggle from a broader context.

The psychic prison metaphor asks how people (students, teachers, parents) are influenced by the code and what are its ramifications. Do the coded environments change people, and if so, how? Or, as Jaron Lanier asks, Do we change ourselves because of them? (Lanier, 2010). How does the filter bubble affect learning or searching for information? How different can the coded environments be, for example, between teacher and students? How do we deal with the loss of common "neutral" media such as newspapers? Themes like obsessive gaming, social media usage, and critical, self-aware ways of using digital technologies are at the heart of this metaphor. These questions can also lead to self-discovery in the digital age through different challenges students can face, for example being without a smartphone for a day or projects such as the Bored and Brilliant project organised by the WNYC radio show Note to self (http://www.wnyc.org/series/bored-and-brilliant/). Wajcman has written about the paradox of loss of time when using digital technologies that save us time in more detail in her latest book (Wajcman, 2014).

While the culture and political system metaphors dealt with many cultural and societal issues from a general standpoint, the instrument of domination metaphor emphasises the power issues of the code. Code is a tool for building structures

Table 2
Different views to teaching code.

Metaphor	Meaning for education on code and programming.
Mechanic	Learning a programming language, or logic.
Organism	Understanding the structure of complex code.
Brain	Understanding the "intelligence" of code.
Flux & Transformation	How code can solve problems.
Culture	Placing coding in its cultural context.
Political System	Understanding the ways code affect society.
Psychic Prison	Understanding how code influences individual.
Instrument of domination	Seeing the power issues involved in code.
Carnival	Learning to use code as a way of self-expression and as a tool of understanding code.

and obtaining knowledge, and whoever has control over these structures and information has power over the users of the software or service. As Rushkoff points out, some of the issues created by code are inherent in the code itself, and some are created by the people developing code. An example of the former is the binary nature of the code that leads to a different mode of thinking that humans do. An example of the latter is the hijacking of the social connections that people form over the internet, meaning that the platforms that offer connections use those connections for their own purposes, such as harvesting data for market purposes, etc. (Rushkoff, 2010). Being aware of the power issues inherent in the code is crucial in forming a critical understanding of the code. Increased awareness of these issues and their origins on the level of code may help students to become more critical consumers, and it may also trigger changes in these platforms. When the students are able to detect controlling structures inherent in code, they are also empowered to challenge these structures, which may create a new power dynamic in the digital world.

The former examples have been mostly about gaining skills (learning a programming language), learning how the world works (the ubiquity and influence of code) and debating what is preferable. The postmodern paradigm and the carnival metaphor highlight the creativity, emotions and experience in education about code. The postmodern paradigm emphasises the deconstruction and reconstruction of the concept of code. The carnival metaphor uses the code itself to challenge the idea of the code. It can encompass all the other metaphors or views of code to create a statement of itself. The tool it uses for this is the code itself. It shows how important arts and craft is in the understanding of the code. Not only can creativity be used to invent something, but it can also be used as a tool to understand code, or to critique code and its usage. Creating something by hand is an important tool in knowledge acquirement (Kojonkoski-Rännäli, 1998), and creative use of the code could be argued to be part of the craft skills of 21st century.

The different viewpoints and suggestions for education around code and programming are summarised in Table 2. Our point is not to recommend that a particular metaphor should be followed and others ignored, or to suggest a ranking of the usefulness of the metaphors. Instead, we argue that all of the areas metaphors brings out should be included in the teaching of code and programming. As we proposed in the beginning it might be more fruitful to think about teaching programming in the basic curriculum to be more about improving code literacy, than about teaching coding as merely a mechanical skill. Code literacy includes both understanding the more ambiguous and multiplexed issues that exist around code, and the basic principles and logic of coding. The machine and organism metaphors in the functionalist paradigm set the basis for understanding code from the technical perspective. This helps to understand how code is used in more complex real world situations, as the brain and flux & transformation metaphors illustrated. The culture and political system metaphors help to broaden the scope towards societal issues, while the instrument of domination and psychic prison metaphors illustrate the coercive characteristics code can have. Finally, the postmodern paradigm and the carnival metaphor broaden the method of learning about code from thinking and discussing to experience and creativity.

These metaphors may be implemented in several ways as a part of ICT education. The metaphors and the issues may be divided between different disciplines and may thus be more evenly distributed in existing school subjects. Or they can be studied as a whole in a phenomenon-based learning project, which can combine different school subjects together to form a larger picture of the subject. Or programming could be its own subject, where it would not only include mechanical knowledge of programming, but it would incorporate all

the different issues we have brought forth in this article. Code could also be seen as a new subject: as a "digital survival skills for digital natives." In Finland, recent plans to focus more on phenomenon-based learning discloses many interesting opportunities in teaching code and creating a broader understanding around it – improving code literacy. (Halinen, 2014).

5. Discussion & conclusion

As coding and code literacy are gaining more popularity, what is meant by code becomes more important. However, the societal discussion around code is still fragmented and partly superficial, focusing only on a few points of view and more often on a mechanical understanding of the code. There is also traction between these different views. Our article illustrates ways of embracing the tensions, and also of raising the neglected aspects to the educational agenda. We propose that the aim should not be just on code and programming as a skill (coding), but also as a capability of better understanding the world and its structures. This understanding can be seen to become even more important in the future.

We propose the metaphors as a useful heuristic for illustrating different viewpoints on code. However, some limitations can also be identified. From the theoretical side, the key question is do the metaphors adapted from the organisation and systems science cover every important aspects of the code? This relates to another limitation, that of the lack of empirical evidence. While we do illustrate the metaphors with examples, we have not presented an empirical case study where all the metaphors would be used. We believe that such a case study would be a fruitful direction for further research and would help to refine the metaphors Furthermore an empirical case study would enable analyzing how different metaphors interact with each other, where are the main tensions, which metaphors are closely linked to each other etc. Further research could also focus on the social practices and historical backgrounds of these metaphors. These points are out of the scope of this article, as we have focused on describing the metaphors and using them as a lens to focus on various effects code has. Another strand of possible future research might be the focus on emancipatory paradigm and for example dissecting platform monopolies and the ways they govern the code. Related to this, interesting work regarding educational platforms has been done by Williamson. (For example see: https://codeactsineducation.wordpress.com).

Our approach illustrates that there are multiple views of what code is and how it influences our everyday lives. This understanding may help to better reflect the needs of future education. The metaphors we have described can be used as one way to support the planning of education around coding as well as to structure the discussion around code and coding. From a societal standpoint, the metaphors help to identify the dominant metaphor and thus to understand the current direction of code-based issues. Contrasting the dominant metaphor with the alternative views proposed by the metaphors presents us with alternative future directions. However, we do not propose that any singular view is sufficient by itself. Rather, the focus should be on opening the discussion, allowing plural views and helping to take different views systematically into account.

Acknowledgements

The research leading to these results has received funding from the Strategic Research Council at the Academy of Finland under grant agreement no 293446 – Platform Value Now: Value capturing in the fast emerging platform ecosystems.

References

Alvesson, M., & Deetz, S. A. (1996). Critical theory and postmodernism approaches to organizational studies. In *The SAGE handbook of organization studies*. pp. 255–283. SAGE Publications Ltd. http://dx.doi.org/10.4135/9781848608030.n8
Arnall, T. (2011). *Immaterials*,. Retrieved August 1, 2015, from. http://www.nearfield.org/2011/02/wifi-light-painting
Arnall, T. (2014). *Immaterials*,. Retrieved August 1, 2015, from. http://www.nearfield.org/2014/08/satellite-lamps
Barzilai, S., & Zohar, A. (2006). How does information technology shape thinking? *Thinking Skills and Creativity*, *1*(November (2)), 130–145. http://dx.doi.org/10.1016/j.tsc.2006.08.001
Boden, M. A. (1996). *Artificial intelligence*. Academic Press.
Borsook, P. (1995). How anarchy works. *Wired*,. Retrieved 1.8.2015 from:. http://www.wired.com/1995/10/ietf/
Ceruzzi, P. E. (2012). *Computing*. MIT Press.
Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., et al. (2015). *Comprehensive experimental analyses of automotive attack surfaces*. Center for Automotive Embedded Systems Security. http://www.autosec.org/pubs/cars-usenixsec2011.pdf
Chrisley, R., & Begeer, S. (2000). *Artificial intelligence*. Taylor & Francis.
Coleman, E. G. (2012). *Coding freedom – The ethics and aesthetics of hacking*. USA: Princeton University Press.
Cox, B. J. (1985). *Object oriented programming*. Addison-Wesley.
Cox, G. (2013). *Speaking code*. MIT Press.
Fairclough, N. (1995). *Critical discourse analysis*. Longman Group Limited.
Fleischer, R., Kullenberg, C., Brunila, M., Nikkanen, H., Purokuru, P., Wåg, M., et al. (2014). Verkko suljettu. In M. Brunila, & K. Kallio (Eds.),. Into Kustannus.
Foley Lardner LLP. (2014). Is it a car or a computer on wheels? *Dashboard Insights*,. Retrieved: 1.8.2015. From. http://www.autoindustrylawblog.com/2014/06/09/is-it-a-car-or-a-computer-on-wheels/
Freire, P., Freire, A., & De Oliveira, W. F. (2014). *Pedagogy of solidarity*. Left Coast Press Inc.
Fsf. http://www.fsf.org/about/
Fsf. https://www.gnu.org/philosophy/free-sw.html
Giroux, H. A. (2011). *On critical pedagogy*. USA: Bloomsbury Publishing.
Halinen, I. (2014). Presentation: Ops2016-koulu katsoo tulevaisuuteen. *ITK-conference*, 10.4.2014.
Hirsch, J. (2015). *Elon Musk: Model S not a car but a sophisticated computer on wheels*. (March 19 Retrieved September 20, 2015, from]. http://www.latimes.com/business/autos/la- fi-hy-musk-computer-on-wheels-20150319-story.html
Jackson, M. C., & Keys, P. (1984). Towards a system of systems methodologies. *Journal of the Operational Research Society*, *35*(6), 473–486. http://dx.doi.org/10.1057/jors.1984.101
Jackson, M. C. (2003). *Systems thinking*. SAGE Publications Ltd. http://dx.doi.org/10.4135/9781446263556
Jackson, M. (2007). *Systems approaches to management*. Springer Science & Business Media.
Jarvis, J. (2012). *Rewired youth?, BuzzMachine*. Retrieved July 1, 2015. from:. buzzmachine.com/2012/02/29/rewired-youth/
König, W., Winner, L., Hughes, T. P., Schwartz, R. C., Cockburn, C., Bloch, M., et al. (1985). The social shaping of technology. In D. MacKenzie, & J. Wajcman (Eds.),. Open Univeristy Press. http://dx.doi.org/10.5771/9783845269238-268
Kiili, C. (2012). *Online reading as an individual and social practice*. Jyväskylä Studies in Education.
Knochel, A. D., & Patton, R. M. (2015). If art education then critical digital making: Computational thinking and creative code. *Studies in Art Education*.
Kojonkoski-Rännäli, S. (1998). *Ajatus Käsissämme*. University of Turku, Rauman opettajankoulutuslaito.
Kuhn, T. S. (1970). *The structure of scientific revolutions*. Chicago: University of Chicago Press.
Kupiainen, R. (2013). *Diginatiivit Ja Käyttäjälähtöinen Kulttuuri*. Widescreen.fi. Retrieved April 16 2015 From:. http://widerscreen.fi/numerot/2013-1/diginatiivit/
Kurzweil, R. (2005). *The singularity is near*. Penguin.
Lakoff, G. (2009). The contemporary theory of metaphor. In A. Ortony (Ed.), *Metaphor and thought* (2nd ed., pp. 202–251). Cambridge: Cambridge University Press. http://dx.doi.org/10.1017/CBO9781139173865.013
Lanier, J. (2010). *You are not a Gadget*. Vintage.

Lessig, L. (1999). *Code and other laws of cyberspace*. Basic Books (AZ).
Lessig, L. (2009). *Code 2.0*. CreateSpace.
Levy, S. (2010). *Hackers*. O'Reilly Media, Inc.
Louis, M. R., Burrell, G., & Morgan, G. (1983). Sociological paradigms and organizational analysis. *Administrative Science Quarterly*, *28*(1), 153. http://dx.doi.org/10.2307/2392394
Manovich, L. (1999). Database as symbolic form. *Convergence: The International Journal of Research Into New Media Technologies*, *5*(2), 80–99. http://dx.doi.org/10.1177/135485659900500206
McCorduck, P. (2004). *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. Natick, Mass: AK Peters.
Mehra, B., Merkel, C., & Bishop, A. P. (2004). The internet for empowerment of minority and marginalized users. *New Media & Society*.
Morgan, G. (2006). *Images of organization*. SAGE Publications.
Morozov, E., 2013. To Save Everything. Click Here: The Folly of Technological Solutionism. PublicAffairs.
O'Neill, N. (2008). *Mark Zuckerberg, Sarah Lacy interview video*. Retrieved August 8, 2015. From:. adweek.com/socialtimes/mark-zuckerberg-sarah-lacy-interview- video/304287
Opetushallitus. (2014). *Perusopetuksen opetussuunnitelman perusteet*.. Retrieved July 21, 2015 From:. http://www.opinsys.fi/ohjelmointi-ja-perusopetuksen-opetussuunnitelman-perusteet
Pariser, E. (2012). *The filter bubble* (ibooks ed.). UK: Penguin.
Petzold, C. (1999). *Code: The hidden language of computer hardware and software* (1st ed.). Redmond: Microsoft Press.
Pollari, M. (2014). *Ohjelmointi Osaksi Opetusta, Mistä Ensiapua?* Luma.fi Retrieved September 9, 2014. From:. http://www.luma.fi/artikkelit/3183/ohjelmointi-osaksi-opetusta-mista-ensiapua
Rushkoff, D. (2010). *Program or Be programmed*. OR Books.
Rushkoff, D. (2013). *Present shock: When everything happens now*. Penguin.
Saariketo, M., 2015. Presentation: Koodauksesta kansalaistaito? Koodaustaidon julkinen määrittely yhteiskunnallisena kamppailuna at Toimijaksi kasva(tta)minen ohjelmoidussa yhteiskunnassa in Univeristy of Tampere 9.10.2015.
Scholz, A. (2014). *Smile TV*. Retrieved September 1, 2015, from:. http://www.creativeapplications.net/maxmsp/smile-tv-works-only-when-you-smile/
Stallman, R. M., Gay, J., & Lessig, L. (2009). *Free software, free society*. Hodder Christian Books.
Steiner, C. (2013). *Automate this: How algorithms took over our markets, our jobs, and the world*. Portfolio Trade.
Sterling, L. (2015). *An education for the 21st century means teaching coding in schools*.. Retrieved July 1, 2015, from:. http://theconversation.com/an-education-for-the- 21st-century-means-teaching-coding-in-schools-42046
Stiegler, B. (2005). *Manifesto*. Arsindustrialis.org. Retrieved April 1, 2015, from:. arsindustrialis.org/node/1472
Stiegler, B. (2010). *Manifesto*. Arsindustrialis.org. Retrieved April 1, 2015, from:. arsindustrialis.org/manifesto-2010
Thibodeau, P. H., & Boroditsky, L. (2011). Metaphors we think with: The role of metaphor in reasoning. *PLoS ONE*, *6*(2), 16782. http://dx.doi.org/10.1371/journal.pone.0016782
Thorp, J. (2009). *Just landed*. Retrieved August 1, 2015, from:. http://blog.blprnt.com/blog/blprnt/just-landed-processing-twitter-metacarta-hidden-data
Turkle, S. (2011). Alone together: Why we expect more from technology and less from each other. In *Basic books*.
University of California-Davis. (2007). China's eye on the internet. *ScienceDaily*,. Retrieved 21 july 2015, from:. www.sciencedaily.com/releases/2007/09/070911202441.htm
Vaden, T. (2005). Digital nominalism. Notes on the ethics of information society in view of the ontology of the digital. *Ethics and Information Technology*, *6*(4), 223–231. http://dx.doi.org/10.1007/s10676-005-0350-7
Vallance, C. (2015). *Car hack uses digital-radio broadcasts to seize control*.. Retrieved September 10, 2015, from. http://www.bbc.co.uk/news/technology-33622298
Wajcman, J. (2014). *Pressed for Time: The Acceleration of Life in Digital Capitalism*. University of Chicago Press.
Warschauer, M. (2004). *Technology and social inclusion*. MIT Press.
Washington, G. (2015). *84% of all stock trades are by high-frequency computers . . . only 16% are done by human traders*. Zero Hedge. Retrieved September 23, 2015, from. zerohedge.com/contributed/2012-17-26/84-all-stock-trades-are-high-frequency- computers-%E2%80%A6-only-16-are-done-human-tra
Weiss, G., & Wodak, R. (2006). Introduction. In G. Weiss, & R. Wodak (Eds.), *Critical discourse analysis: Theory and interdisciplinarity*. Palgrave Macmillan.
Williamson, B. (2015). Political computational thinking: Policy networks, digital governance and 'learning to code'. *Critical Policy Studies*, http://dx.doi.org/10.1080/19460171.2015.1052003
Williamson, B. (2016). Digital education governance: Data visualization, predictive analytics, and 'real-time' policy instruments. *Journal of Education Policy*, *31*(2), 123–141. http://dx.doi.org/10.1080/02680939.2015.1035758
Wilson, M., Scalise, K., & Gochyyev, P. (2015). Rethinking ICT literacy: From computer skills to social network settings. *Thinking Skills and Creativity*, http://dx.doi.org/10.1016/j.tsc.2015.05.001