Andreas Larsson is a Doctoral student in Technology education at the Department for Behavioural Science and Learning (IBL), Linköping University. His research mainly focusses on the relationship between teachers' speech, gestures, and knowledge in programming. Andreas is also the head of The Swedish Centre for Science and Technology Education – NATDID

ANDREAS LARSSON
Linköping University., Sweden
andreas.b.larsson@liu.se

Reading the Code Between the Lines – Exploring the Structure of metaphors in educational programming resources

Abstract

Recent work in computer science education shows that natural language plays a pivotal role in learners' understanding of programming concepts. This study explores metaphorical expressions in four computer programming textbooks and online resources in Swedish upper secondary education. The Metaphor Identification Procedure was applied to identify metaphoric language. The metaphors reveal how expressions such as ***the 'program asking' or the 'function building'*** *are structured in relation to embodied experiences. The results show that central concepts are structured in relation to metaphors such as* INANIMATE PHENOMENA ARE HUMAN AGENTS *and* **ORGANISATION IS PHYSICAL STRUCTURE***. Findings also demonstrate differences in the types of metaphors are present in each resource, with* EVENTS ARE ACTIONS *communicated most frequently. Lastly, the resources vary in how they describe the role of the programmer: as a 'constructor' or 'instructor'. This implies that the discovered metaphoric structure in textual resources might influence students' subsequent learning of programming concepts.*

INTRODUCTION

Because computer programming is an abstract activity, learning and teaching programming require using natural language that reflects how the program operates in a concrete way. Furthermore, the natural language related to computer programming must mirror the structure of the code being used in specific situations. Throughout the programming process, novices as well as expert programmers use metaphorical language in relation to both programming concepts (e.g., loop, function or structure) and the code itself (Colburn & Shute, 2008). Metaphorical language allows us to speak and reason about the abstract programming process in terms of the concrete. The interaction between the programmer, the computer and the code can therefore be considered a linguistic activity.

It is rare for students to reflect on their own programming activities in relation to programming concepts (Vieira, Magana, Roy, & Falk, 2019). Instead, programming within an educational context often becomes a game of trial-and-error or a pure 'thinking practice'. To understand how programming concepts are related to one another, students usually need to experiment with different ways of using programming language to communicate solutions to given problems. In that sense, novice programmers are akin to children learning to speak (Colburn & Shute, 2008). Furthermore, as syntactic conventions are generally English-based, it is likely that understanding and handling error-messages might become an obstacle for non-English speaking students' trying to develop novel code (Becker, 2019). Colburn and Shute (2008) argue that the metaphors present in the language surrounding computer programming education have three roles: (1) A pedagogical role in which, when encountering programming concepts such as *catch* or *throw*, students can structure programming in relation to bodily experiences and thereby obtain an initial understanding of a programming concept in question. (2) A design-oriented role allowing programmers to use words such as *jump*, *kill* or *run* as source domains for programming concepts that facilitate students' communication. (3) A scientific role in which metaphors can be used as analytical tools to study computer programming phenomena.

The last ten years has seen increasing research directed towards the use of metaphorical language within science education and other related areas (e.g., Amin, Jeppsson, & Haglund, 2015) such as metaphor use in relation to energy (e.g., Dreyfus et al., 2014), metaphors in problem-solving (Hidalgo-Cespedes, Marin-Raventos, Lara-Villagran, & Villalobos-Fernandez, 2018) and the metaphorical nature of the language used in textbooks (Bråting & Kilhamn, 2021; Hedberg, Haglund, & Jeppsson, 2015; Stavrum, Bungum, & Persson, 2020). In parallel, growing research concerning computer science education related to metaphors, computing concepts and programming languages is emerging (e.g., Gogolla & Stevens, 2018; Long, 2007; Robins, Rountree, & Rountree, 2003; Woollard, 2005). Research in computer science has a long tradition of metaphor-based studies in areas such as human-machine-interaction (Carroll & Mack, 1985; Hurtienne et al., 2010). The methodology and data in these studies vary from large-scale, text-based studies (Blackwell, 2006; Videla, 2017) to video observations focusing on the varying use of conceptual metaphors and gestures among both students and teachers (Manches, McKenna, Rajendran, & Robertson, 2020; Solomon, Bae, DiSalvo, & Guzdial, 2020). Furthermore, a number of studies have focused on student-generated computing metaphors (Agribas, 2018)) and teachers' use of metaphor while teaching (Larsson & Stolpe, 2022; Chibaya, 2019). Lacking, however, are empirical studies focusing on the metaphorical nature of the language found in computer programming textbooks and online resources.

As noted by Qian and Lehman (2017), at least some of students' programming difficulties can be related to natural language. At worst, teachers' use of misleading metaphorical language can result in limited, or formation of faulty mental models of programming concepts, and deficient problem-solving abilities (Forišek & Steinová, 2012; Kwon, 2017), or a lowered sense of being in control of the machine (Caporael, 1986). It is therefore essential that educators gain an understanding of how the natural language of humans relates to students' understanding of computer programming concepts and the syntax of the code (Qian & Lehman, 2017). Consequently, Qian and Lehman (2017) state that there is a need for research that goes beyond documenting the use of misleading metaphors. Speaking about computers in terms of people could allow us to speak about computers as if they are smart, powerful and user-friendly machines, or to understand the concept of a queue in programming (Peelle, 1983). Consequently, the metaphors provided in a textbook or used in a classroom will influence how learners structure, communicate and study computer programming (Colburn & Shute, 2008), and also how they might approach and solve programming challenges (Qian & Lehman, 2017).

AIM OF THE STUDY

The aim of this study is to explore the metaphoric language in four text resources related to programming in Swedish upper secondary technology education. More specifically, the study aims to identify how the metaphoric language of the resources contributes to computer programming in relation to embodied experiences. The following research questions will be posed and resolved:

1. How is the metaphoric language in the resources structured in relation to embodied experiences?
2. In what ways do the identified metaphors affect how the resources conceptualize computer programming?

THEORETICAL FRAMEWORK

Metaphoric language is closely tied to how we use embodied experiences to communicate and reason about aspects of abstract phenomena, i.e., our use of conceptual metaphors (Gibbs, 2007; Gibbs & Colston, 2012; Lakoff & Johnson, 1980, 1999). Conceptual metaphors arise from the reality that we exist in, forming experiential correlations between our perceptions of everyday experiences (e.g., motion, temperature or height) – termed the source domain – and perceptions of abstract phenomena (e.g., time, comfort or quantity) – termed the target domain (e.g. Johnson, 2007; Lakoff & Johnson, 1980, 1999). These are correlations that in turn structure metaphors such as Time is Motion (metaphors will henceforth be indicated using capital characters), Comfort is Warmth, and More is Up (Grady, 1997).

Within Cognitive Linguistics, conceptual metaphors are considered parts of our conceptual system rather than mere linguistic entities (Lakoff, 2008). In a computer programming context this means that in a metaphorical expression such as 'the ***value*** is ***placed inside***' (instances of words that are used as metaphors will henceforth be notated using bold italic characters), the abstract concept ***value*** takes on a 'physical' form and is structured in relation to embodied experiences of placing objects inside a container (Chibaya, 2019). Another example would be when 'the program then ***waits*** until the condition is fulfilled'. Here, the working of the code is structured in relation to embodied experiences of stopping. Moreover, the metaphor displays an example of a personification (see e.g., Lakoff & Johnson, 1980), a connection between a phenomenon (the execution of a computer program) and a human action. Lakoff and Johnson (1999) argue that the metaphors manifested in language reflect the cognitive processes that are involved in structuring a phenomenon. Hence, by studying the metaphorical nature of programming concepts, we will be able to gain an understanding of the experientiential basis of programming.

A particular category for analysing the experiential basis of programming concepts are primary metaphors. In contrast to conceptual metaphors, a primary metaphor is based on one single mapping between an embodied experience (a source domain) and a specific abstract phenomenon (a target domain) (Gibbs, 2017). Primary metaphors are reflected in expressions such as 'I am ***boiling*** with rage', in which our concept of rage is structured in relation to embodied experiences of perceiving increased body temperature while feeling angry. Grady (1997) terms this experience as the experiential motivation for the metaphor Anger is Heat. Grady (2005) argues that primary metaphors with similar experiential motivation may be combined, thus allowing for novel metaphors based on previous experiences. Hence, by attending to the connections between the sensory and the non-sensory elements of a metaphor (conceptual or primary), it is possible to predict a plausible scenario in which the source and target element of a programming metaphor would co-occur and become the experiential motivation for programming concepts, i.e., how computer programming is conceptualised in terms of embodied experiences.

METHODOLOGY

Data Selection

It is common for students to use online resources to quickly solve problems (Pierce & Aparício, 2010). This is something often encouraged by teachers. Furthermore, as observed by Stigberg and Stigberg (2020), teachers tend to use the internet as a source for acquiring programming knowledge. Consequently, both resources available online as well as in books are relevant to this study. Python (recommended as by the Swedish National Agency for Education as a suitable beginner's text-based language in schools) and C++ (used in both upper secondary and higher education) can be regarded as established programming languages in Swedish upper secondary education. Resources relating to these two languages were therefore selected to increase the ecological validity of the study. It is worth noting that two of the resources (Resource A and B) are authored and published in Swedish. Here, the metaphor candidates have been identified in Swedish, translated, and analysed in English.

The data for the present study comprises metaphoric units (henceforth indicated using bold characters) identified in the natural language present in four different textual educational resources on programming – two online tutorials and two textbooks. Examples of these would be 'the program will **be able to print**', 'the **power** of abstraction' or the 'computer **remembers** the dictionary **returned**'. As the study focuses on exploring the general structures of metaphoric language present in the textual resources rather than programming language-specific concepts as expressed in the programming syntax, online resources and book chapters (Table 1) that cover the basic structures and immediate features of the programming language have been included in the analysis. Consequently, examples of syntax have ben excluded from the corpus. Furthermore, as the study focuses on written natural language, resources containing rich descriptions of programming and the functions of a program have been selected. In cases where example of code have been included in a text (e.g., sumProblem or variable names), they have been considered instances of natural language.

Table 1. An overview of the four textual educational resources analysed in the study

	Title	**Author**	**Programming language**	**Resource Type**	**Language**
Resource A	C++ Direkt	Skansholm, J.	C++	Book	Swedish
Resource B	Programmering 1 med Python : lärobok	Sundström, J.	Python	Book	Swedish
Resource C	C++ Language Tutorial.	Soulié, J.	C++	Online	English
Resource D	Hands-On Python A Tutorial Introduction for Beginners.	Harrington, A. N.	Python	Online	English

Data Analysis

The Pragglejaz Group's Metaphor Identification Procedure (MIP) (see Gibbs & Colston, 2012; Hedberg et al., 2015; Pragglejaz Group, 2007) was adopted as the systematic method for identifying metaphoric language from the computer programming texts (table 1). Based on Cognitive Linguistics, such an analysis is a complex yet powerful procedure. Complex in the sense that metaphorical expressions can vary in meaning depending on context or situation (Gibbs & Cameron, 2008); powerful in the sense that when context is accounted for, analysing the conceptual structure of language can provide knowledge of how our understanding of programming concepts are grounded in embodied experiences, i.e., an experiential ground for understanding computer programming. Additionally, metaphorical language does not follow any standard grammatical or lexical form. As such, metaphorical language cannot be easily identified in a 'mechanical' way using ready-made algorithms (Stefanowitsch & Gries, 2007). Furthermore, even though the readers may have extensive prior knowledge of such areas as computer programming, they will not necessarily read a text in the way it was intended by the author; what was intended as a metaphor may still be interpreted literally by the reader (Gibbs & Colston, 2012).

MIP was developed by the Pragglejaz Group (2007) with the aim of crafting a systematic way of exploring "*metaphors in the wild*" (p. 1), By employing this approach to metaphor theory, the focus changes from considering metaphors as mere isolated expressions – often constructed for experimental purposes – to exploring how natural language occurs in 'real life' contexts (Gibbs, 2015; Jeppsson, Haglund, Amin, & Strömdahl, 2013). Furthermore, the method was developed to address the problem of researchers relying on their 'gut feelings' rather than using established criteria for what is and is not a metaphor. This is done by primarily regarding the meaning of a word in relation to context in which it is manifested.

The identification procedure was performed as follows:

1. Read the full text to establish a general understanding of the meaning
2. Determine the metaphor candidates in the texts
3. For each metaphor candidate:

    a. determine its meaning in the context
    b. determine if it has a more conventional/basic meaning in another context.
    c. If (b) is yes, mark as metaphoric unit

The data identification was performed manually by the author and rendered a total of 2 400 metaphor candidates (see example below) that were considered metaphoric across the four text-based resources (circa 600 units per resource).

To demonstrate the application of MIP in the current work, the following excerpt is used as an example. The excerpt that can be considered representative of the text-based resources.

> *A quick way to build a list is to format a list i.e., 'list comprehension'. One such formatting means that you create a new list in which every element is a result of some operation on the element of another iterable variable, perhaps as part of a sequence that satisfies a specific condition. (Resource B, p. 103)*

Identification of metaphoric units

All verbs, nouns, adjectives, prepositions, and adverbs were regarded as metaphor candidates (metaphor candidates are notated using **bold** font). As the general meaning of the sentence is part of a larger text concerning the construction of lists in a computer program, all metaphor candidates that have a more conventional meaning in other contexts are to be considered metaphoric units. For example, 'build' generally refers to the erection of a physical structure. Consequently, the metaphor candidate 'build' has a more conventional meaning in a construction context than in a programming context. Similarly, the metaphor candidate 'comprehension' – with a conventional meaning in relation to human cognition – should be considered as metaphoric. The metaphor candidate 'quick' however, does not have a more (or less) conventional meaning in other contexts than programming and is hence not considered as a metaphoric unit.

> *A quick way to* **build** *a* **list** *is to* **format** *a* **list** *i.e.,* **'list comprehension'**. *One such formatting means that you create a new list in which every* **element** *is a result of some* **operation** *on the* **element** *of another iterable variable, perhaps as part of a* **sequence** *that* **satisfies** *a specific condition.* (Resource B, p. 103)

Categorisation of metaphoric unit

The identified metaphoric units were then analysed to determine the metaphoric structure of the excerpt. This was done by employing a three-stage process based on an adaptation of Lakoff and Johnson's (1999) Integrated Theory of Primary Metaphor Larsson, Stafstedt & Schönborn (2019). Firstly, the identified metaphoric units were categorized in relation to the primary metaphors listed by Grady (1997) (here, notated using superscripted numbers, refering to the categories in (Table 2). In the case of the Swedish resources (see Table 1) the identification procedure was based on the original texts. The metaphoric units were later translated and categorised in relation to English primary metaphors.

> *A quick way to* **build**^{1} *a* **list**^{3} *is to* **format**^{1} *a* **list**^{3} *i.e.,* **'list comprehension**^{2}**'**. *One such formatting means that you create a new list in which every* **element**^{3} *is a result of some* **operation**^{4} *on the* **element**^{3} *of another iterable variable, perhaps as part of a* **sequence**^{5} *that* **satisfies**^{1} *a specific condition.* (Resource B, p. 103)

Secondly, similarities in experiential motivation between units were identified, indicated using underlined font (Table 2). Here, for example, both Nature of an Entity is Shape and Essential is Internal concern the properties of an object. Thirdly, the agent of each metaphor – the active part of the excerpts – was indicated and tabulated.

Table 2. Categorisation of metaphoric units of the above example in relation to primary metaphor and their relation to experiential motivation

	Primary Metaphor	**Experiential Motivation**	**Agent**	**Metaphoric Unit**
1	Events are Actions, Inanimate Phenomena are Human Agents	The correlation between observable events in our environment and the presence of human agents.	Programmer	build, satisfy, format
2	Contextual roles are locations	The tendency for people/objects filling certain functions to be found in particular locations.	Program	comprehension
3	Nature of an entity is shape	The tendency to draw inferences about an object from its shape. The correlation between an object's shape and its behaviour.	Program	element, list
4	Essential is Internal	The correlation between internal features of objects and their essential properties	Program	operation
5	Events are Transferred Objects	The correlation between receiving a transferred object and being affected in some way.	Program	sequence

Altogether, the process of identifying the metaphoric units and analysing their relation to primary metaphors and experiential motivation reveals the underlying structure of the metaphoric language. This works partly in terms of (1) the mappings between elements in one domain to similar elements in other domains (as seen in the distribution of the metaphoric language) and (2) partly in terms of the general structure of the metaphoric language of the resources (as seen in the metaphorical structure and contextualisation of the metaphoric language) (Gibbs, 2017).

RESULTS

Distribution of Metaphoric Language across the Resources

Figures 1-4 show how the metaphoric language in each resource is structured, i.e., the distribution of primary metaphors (e.g., Events are actions, Inanimate Phenomena are Human Agents, (Logical) Organisation is Physical Structure and Essential is Internal) in relation to the respective resource. Such examples would be where code is related to actions (e.g., 'the computer **printing**' or 'the program **returning** a value' or where the program '**builds structures**' by using "code-**elements**"). Other examples would be the primary metaphor Existence is Visibility underpinning for example, the 'declaration of a variable'. As shown in Figures 1 and 2, the two educational resources A and B display a relatively low diversity in how their language is structured in relation to primary metaphors, while resources C and D show diverse metaphoric language. This suggests that resources C and D will provide increased possibilities of structuring programming concepts in relation to different embodied experiences. In contrast, the metaphoric language identified in resources A and B enhances the possibilities of communicating programming concepts in a uniform manner.

Furthermore, the figures provide initial insight into how the programming content of each resource is structured in terms of agency i.e., who is doing what in the metaphor. For example, in Resources A, B and D, the programmer has no actual agency within the program itself. Rather it is the program – an inanimate phenomenon – that performs human actions (e.g., '**building**' or '**moving objects'**). Instead, the program is the builder (Inanimate Phenomena are Human Agents) and the code-elements are the building blocks for the program ((Logical) Organisation is Physical Structure). Resource C, however, displays an increased number of metaphors related to organisation and the explicit causes thereof (Figure 3). One such example would be when "[t]he argument supplies the actual data to be used in the function execution" (Resource C, p. 26), where the argument is the cause (and instance of the primary metaphor Effects are Objects which Emerge From Cause) and the function is a part of the organisation.

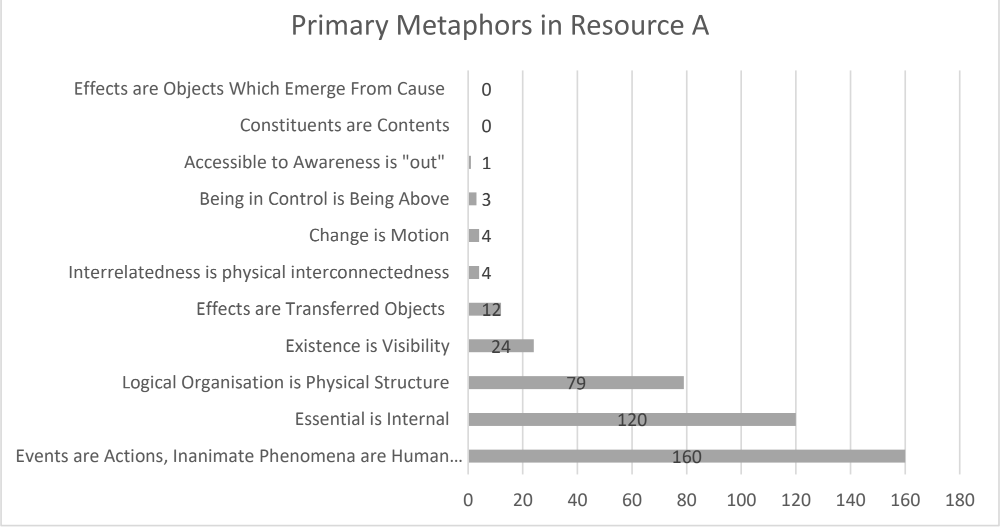

Figure 1. The most common primary metaphors identified in Resource A.

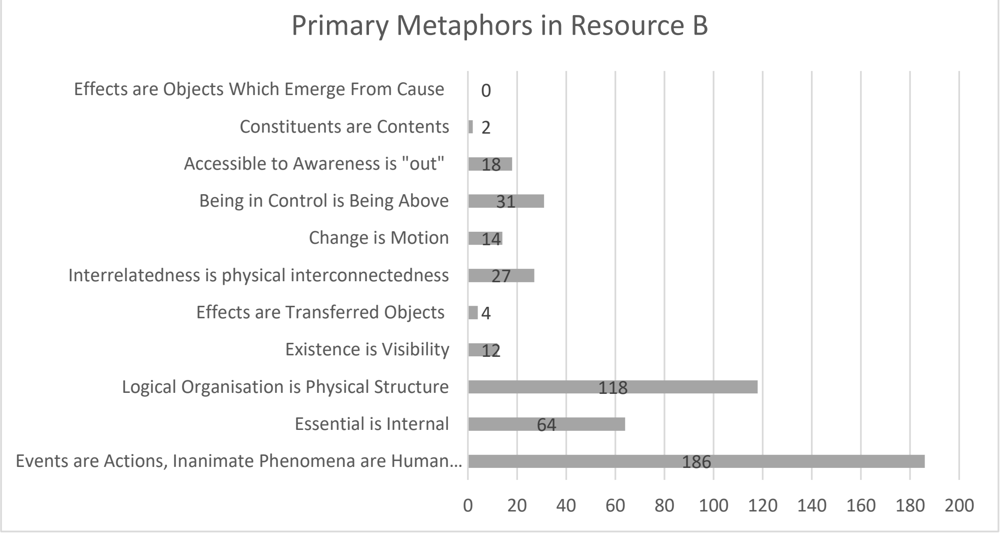

Figure 2. The most common primary metaphors identified in Resource B.

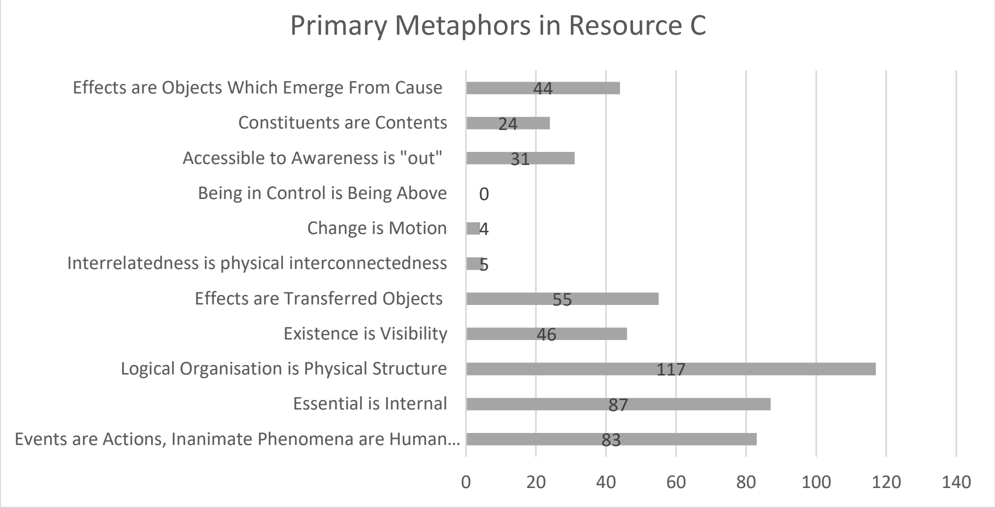

Figure 3. The most common primary metaphors identified in Resource C.

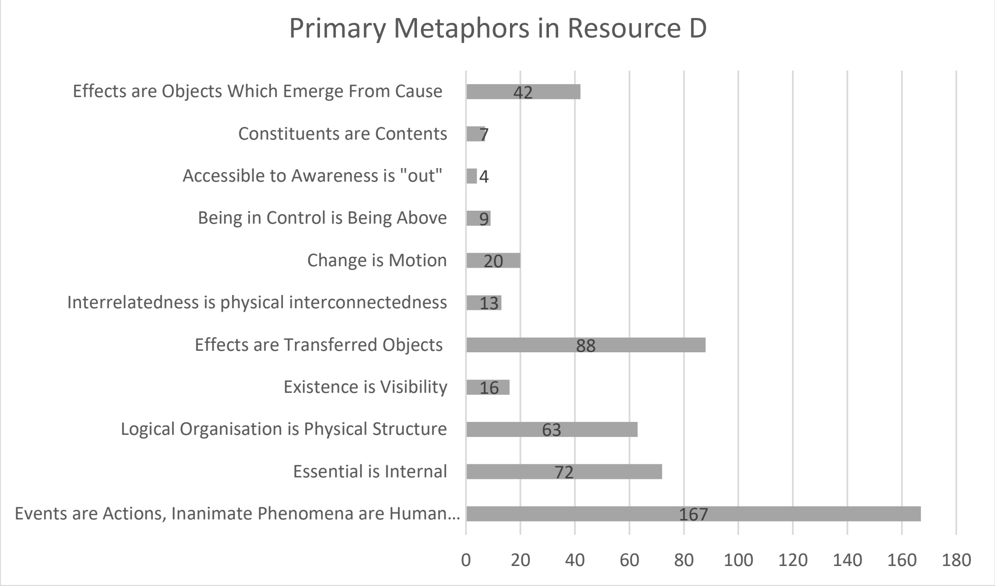

Figure 4. The most common primary metaphors identified in Resource D

The Structure of the Metaphoric Language in the Resources

When relating the metaphoric content of the four educational resources to its experiential motivation, clear differences emerge in how programming concepts are presented within the resources. All the identified metaphoric units are, however, in some way related to aspects of sensory experiences and human activities, instances of structural resemblance (e.g., buildings or the human body) or inherent features of structures (e.g., stable or blocks). The following findings present how each resource is structured in relation to primary metaphor and experiential motivation.

The Structure of Metaphoric Language in Resource As

Of the identified metaphorical language in Resource A, 29% is related to EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS (Figure 1). One such example is displayed below:

> "*such a program* **asks**[1] *the user to enter input data and then* **calculates**[1] *the output data that will be* **presented**[1] *[by the program] to the user*" (p. 41).

As seen in Table 3, two agents are present, one of whom is non-metaphorical (the end-user of the program), while the other one is metaphorical (the program). However, the computer is not present in this excerpt.

Table 3. Categorisation of metaphoric units in an excerpt from Resource A (p. 41) in relation to primary metaphor and experiential motivation

	Primary metaphor	Experiential motivation	Agent	Metaphoric Unit
1	EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS	The correlation between observable events in our environment and the presence of human agents.	the program, the user	asks, calculates, presented

Almost equally frequent, 27%, is the ESSENTIAL IS INTERNAL metaphor (Figure 1) – a metaphor based on the correlations between what is inside an object and its general properties. One such example is demonstrated in the following excerpt:

> "*The program* **reads**[1] *one number at a time to the variable* ***tal***. *The numbers are summed* **in**[2] ***sum*** *and* ***n*** *contains the number of read numbers*" (p. 40).

As seen in Table 4, this excerpt contains more than just simple, one-to-one mappings between sensory and non-sensory domains. The first metaphoric unit – "reads" – is motivated by experiences of watching a human-agent-performed action in our immediate environment (EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS), while the other – "in" – is related to the correlation between internal features of objects and their essential properties (ESSENTIAL IS INTERNAL). Consequently, the metaphoric language in this excerpt could be seen as a link to form a more complex metaphorical relationship in the form of a conceptual metaphor where THE COMPUTER IS A READER OF NUMBERS. Such a link provides for greater variance in how the metaphoric language might be interpreted and used to form new complex metaphors.

Table 4. Categorisation of metaphoric units in an excerpt from Resource A (p. 40) in relation primary metaphor and experiential motivation

	Primary metaphor	Experiential motivation	Agent	Metaphoric Unit
1	EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS	The correlation between observable events in our environment and the presence of human agents.	the program,	reads
2	ESSENTIAL IS INTERNAL	The correlation between internal features of objects and their essential properties	numbers	in

Moreover, as shown in the example below, there are also excerpts that can be related to physical objects and their parts. When read in isolation, these terms are mere building blocks of larger structures (Constituents are Contents). When put into context, however, these terms become the base for the (Logical) Organisation is Physical Structure metaphor.

> "*The text is* **stored internally**[1] *in a string variable in a* **field**[1] *of the type char. The* **components**[1] *in the* **field**[1] *are numbered from 0* **forward**[2]". (p. 67)

This is an example where text is structured as a physical object ("the variable"), which in turn is part of a greater object ("the field"). Furthermore, as indicated by "forward", this expression is structured in relation to movement (Change is Motion) (Table 5).

Table 5. Categorisation of metaphoric units in an excerpt from Resource A (p. 67) in relation to primary metaphor and experiential motivation

	Primary metaphor	**Experiential motivation**	**Agent**	**Metaphoric Unit**
1	(Logical) Organisation is Physical Structure	The correlation between observing the part-whole structure of objects and forming cognitive representation of the logical relationships within them.	the program,	Store, component
2	Change is Motion	the correlation between perceiving motion and being aware of a change in the world-state around us.	numbers	forward

The Structure of Metaphoric Language in Resource B

Of the metaphorical units in Resource B, 29% are based around instances of the primary metaphor Events are Actions, Inanimate Phenomena are Human Agents (Figure 2); expressions that relate to the correlation between the events taking place and the human agents responsible for them. Instances of these relations can be found in the two expressions below:

> "*it* **tells**[1] *the system which programs are supposed to* **handle**[1] *the file*"
> "is **seen**[1] *by Python as a commentary line*" (p. 48).

In contrast to resource A, where the agent in the excerpt corresponds to the program, Resource B bases these metaphoric expressions on mappings between the programming language (in this case Python) and a human agent (Table 6). Consequently, Python receives the role of an instructor within the computer.

Table 6. Categorisation of metaphoric units in an excerpt from Resource B (p. 48) in relation to primary metaphor and experiential motivation

	Primary metaphor	**Experiential motivation**	**Agent**	**Metaphoric Unit**
1	Events are Actions, Inanimate Phenomena are Human Agents	The correlation between observable events in our environment and the presence of human agents.	Programming language	tells, handle, seen

Another primary metaphor commonly employed in Resource B is (Logical) Organisation is Physical Structure (Figure 2), a metaphor that is motivated by the correlation between part/whole structures of objects and cognitive representations of the logic within them. One such example – "block of code" – is present in the excerpt below:

> "*Then,* ***i*** *is incremented, whereupon the program* **goes back**[1,3] *to the while instruction and* **checks** *whether the condition still holds true. If so, the entire block of* **code**[2] *will be iterated once more*" (p. 65).

As seen in Table 7, the above excerpt is based partly on instances of the Events are Actions, Inanimate Phenomena are Human Agents metaphor ("goes back" and "checks"). However, to provide these words with meaning in this programming context, these metaphors need to be structured in relation to an overarching metaphor A Computer Program is a Controller of Human Actions. Here, the use of the expression "block of code" (a part of a larger structure) implies an internal logic based around physical structures. Moreover, as the term "goes back" also implies movement, this overarching metaphor is based in three different source-domains, resulting in a complex text for the learner to comprehend. As seen in Figure 1, the logic organisation metaphor is also common in Resource A, but not, however, in relation to building activities.

Table 7. Categorisation of metaphoric units in an excerpt from Resource B (p. 65) in relation to primary metaphor and experiential motivation

	Primary metaphor	**Experiential motivation**	**Agent**	**Metaphoric Unit**
1	Events are Actions, Inanimate Phenomena are Human Agents	The correlation between <u>observable events</u> in our environment and the presence of human agents.	Program	goes (back), checks
2	(Logical) Organisation is Physical Structure	The correlation between <u>observing the part-whole</u> structure of objects and forming cognitive representation of the logical relationships holding within them.	Program	block of code
3	an Event is the Motion of an Object	The correlation between our location and how we feel. And/or the correlation between <u>perceiving motion</u> and being aware of a change in the world-state around us.	Program	(goes) back

The Structure of Metaphoric Language in Resource C

The metaphoric content present in Resource C is more evenly distributed than in the other text resources (Figure 3). Furthermore, this resource's distinguishing feature is its most common metaphor: (Logical) Structure is Physical Structure. Generally, these metaphorical expressions are related to instances of buildings, resulting in statements such as

> "*Since arrays are* **blocks**[1] *of non-dynamic memory whose size must be determined before execution…*" (p. 54).

Here, the "block" is a part of the larger structure, the "non-dynamic memory", resulting in a part/whole relation between programming elements and physical structures. Consequently, programming can be structured in relation to a building ((Logical) Organisation is Physical Structure) (Table

8) or in relation to the building of a structure (a combination of the (LOGICAL) ORGANISATION IS PHYSICAL STRUCTURE and the EVENTS ARE ACTION metaphors). In order to comprehend such a combination, one would have to include a mapping between an agent (e.g., a human or a program) into the context.

Table 8. Categorisation of metaphoric units in an excerpt from Resource C (p. 54) in relation to primary metaphor experiential motivation

	Primary metaphor	**Experiential motivation**	**Agent**	**Metaphoric Unit**
1	(LOGICAL) STRUCTURE IS PHYSICAL STRUCTURE	The correlation between observing the part-whole structure of objects and forming cognitive representation of the logical relationships holding within them.	none	blocks

The Structure of Metaphoric Language in Resource D

As in the case of Resource B, the Python online tutorial resource relies heavily on metaphors, with the source domain based on experiences of human agents performing actions (EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS) (see Figures 2 and 4). However, in relation to resources A and C, the metaphors in resource D typically concern the transfer of objects (e.g, CHANGE IS MOTION). In this regard, the texts are usually written with an imaginary reader in mind, as exemplified in the following extract:

> "*You introduced a statement to print what* ***sumProblem*** **returns**[1]. *Although* ***sumProblem*** **returns**[1] *nothing explicitly, Python does* **make**[1] *every function* **return**[1] *something*" (p. 30).

This expression provides a representative example of a situation in which the programmer ('you') is intended as the active recipient of the programming activity. Python, now with a human touch, becomes the entity which will return at least something that is to be written. As such, PYTHON IS A TOOL FOR THE HUMAN AGENT, and the returned object is the effect of Python's actions. This way of structuring the excerpt (Table 9) creates a situation where programming becomes an event where one tells the programming language (or the compiler of the language) to perform an action.

Table 9. Categorisation of metaphoric units in an excerpt from Resource D (p. 30) in relation to primary metaphor and experiential motivation

	Primary metaphor	**Experiential motivation**	**Agent**	**Metaphoric Unit**
1	EVENTS ARE ACTIONS, INANIMATE PHENOMENA ARE HUMAN AGENTS	The correlation between observable events in our environment and the presence of human agents.	Python	returns, make

Apart from the human aspects of the metaphoric language, multiple building-related expressions are present in Resource D, where, for example, "*built-in functions*" (p. 58) become important for communicating the programming. Here, excerpts such as "a new object of the specified type is **constructed**[1] *and* **returned**[2]" (p. 60) reveal a clear relation to (LOGICAL) STRUCTURE IS PHYSICAL STRUCTURE, although in the case of Resource D, the programming language is the builder itself (Table 10). Moreover, this excerpt is an example where the programming language also provides motion to the constructed object.

Table 10. Categorisation of metaphoric units in an excerpt from Resource D (p. 58, 60) in relation to primary metaphor and their relation to experiential motivation

	Primary metaphor	Experiential motivation	Agent	Metaphoric Unit
1	(LOGICAL) STRUCTURE IS PHYSICAL STRUCTURE	The correlation between observing the part-whole structure of objects and forming cognitive representation of the logical relationships holding within them	Programming language	constructed
2	EFFECTS ARE TRANSFERRED OBJECTS	The correlation between receiving a transferred object and being affected in some way.	Program	returned

Who has agency in the resources?

As stated by Grady (2005) primary metaphors with similar experiential motivation may integrate with conceptual metaphors, providing an increased degree of complex reasoning about computer programming (see for example Table 7). Consequently, it would be possible to suggest a set of conceptual metaphors that would reflect the general structure of computer programming in the four resources. Moreover, such a set of metaphors would provide knowledge about who is in control over which aspects of the program (e.g., the programmer, the computer, the code).

The combined results show salient differences between the resources. In Resources A and C, the programming process is generally structured in terms of human actions performed by the program and thus controlled by the programmer, while in Resources B and D, it is based around metaphorical language where it is the computer or programming language that has agency in the metaphors (Table 11). This is especially apparent in metaphorical expressions such as “built-in functions” (Resource D, p. 58) in which the programming language has “ownership” over the functions available for the programmer or “[t]he program reads one number at a time” (Resource A, p. 41). Here, the programmer can, at any time, take control over the actions in the computer. Hence, the programming activity in Resources A and C can be described as a non-metaphoric actions, while those in Resources B and D can be described as metaphoric actions. It follows that the choice of resource may affect how students structure and understand their role as programmers.

There is also variation in how the resources address the causes of different actions. This is reflected in the multiple use of the primary metaphor EFFECTS ARE OBJECTS WHICH EMERGE FROM CAUSE in Resources C and D, and the lack thereof in resources A and B (Figure 1-4). This is apparent in metaphoric expressions related to, for example when “Python checks the types and interprets the plus symbol based on the type” (Resource C, p. 14). Another difference between the resources is the emphasis of EFFECTS ARE TRANSFERRED OBJECTS in Resources C and D. When combined, these metaphors form a conceptual ground for structuring the processes in the computer (the effects of running a program) in relation to the inner structure of a program (Table 11).

Table 11. A description of suggested conceptual metaphors of the four resources, related to the role of the programmer, the program and the programming process.

	The Programmer is	The Program is	Computer Programming
Resource A	an Organiser	a human agent	is Organising human agents within a Physical structure
Resource B	a Builder	a Physical Structure, a Controller of Human Agents	is Building Physical Structures That Control Human Agents
Resource C	a Writer of Code	a Set of Instructions To Control (Human) Actions within The Computer	is Writing Instructions to Control Events within The Computer
Resource D	an Instructor	a Human Agent who Performs Events	is Instructing a Human agent to Perform events within the program

DISCUSSION

This study reveals both similarities and differences in how the identified metaphoric language of the resources is structured in relation to primary metaphor and experiential motivation, and in how the resources provide different conceptual metaphors concerning the role of the programmer. Consequently, the teachers' choice of resources affects students' understanding of computer programming concepts and the relationship between the programmer and the computer. This is reflected by a continuum of metaphors spanning from the concrete source domain of the Programmer is a Builder metaphor – where the programmer has agency in the 'construction' of a program – and the more abstract source domain of the Programmer is an Instructor metaphor, where the programmer 'tells' the program what to construct (Table 11). Nevertheless, independent of taking on the role of a constructor or instructor, the students will still need to be able to understand the events going on inside the computer. Based on the metaphorical language of the resources, this will generally be structured in relation to events (e.g., transferal of objects) in physical structures.

In line with findings from Manches et al. (2020) and Chibaya (2019), the results of this study reveal how some of the identified metaphors with similar experiential motivation (see table 7) can be integrated to form more complex metaphors. Here, primary metaphors operate as primitives to more complex metaphoric structures (Lakoff & Johnson, 1999). As closely related metaphors tend to integrate to new and more complex metaphors (Grady, 2005), there is a risk that students develop faulty or misleading metaphors that might hinder further development of their understanding of programming concepts and therefore, their programming skills (see e.g., Forišek & Steinová, 2012; Kohn, 2017). One such example could be found in Table 8, where it is unclear who is the agent in one of the metaphors, resulting in a text that is hard to interpret as a reader. As conceptual metaphors are a necessity for developing a nuanced understanding of complex phenomena (e.g, Lakoff & Johnson, 1999), it is pivotal that learners receive sufficient support in navigating the resources.

Moreover, the results of this study reveal that the resources tend to structure programming concepts around human action taking place *inside* the computer. This can be explained in relation to ongoing personification processes (see Lakoff & Johnson, 1980) resulting in opportunities to speak about the computer as being "smart", having a "memory" or being able to "calculate" and "compute"; terms normally used in relation to human cognition or cognitive abilities (Peelle, 1983). It is worth noting that these, and similar anthropomorphic metaphors, are often used by novice users (Caporael, 1986; Qian & Lehman, 2017) rather than expert users. Hence, an overuse of such metaphors might constrain an understanding of how the computer operates. For example, anthropomorphic metaphors can lure learners into believing that the computer mimics their own problem-solving methods (Kohn, 2017). At the same time, these types of metaphors can become a ground for an emergent understanding of computer programming. Moreover, as the use of metaphorical expressions in language will convey aspects of, for example, context and emotion (see e.g., Gibbs & Colston, 2012), there is a risk that an overuse of anthropomorphic metaphors will enhance attributing blame to the computer for mistakes that are a consequence of faulty programming.

CONCLUSIONS AND IMPLICATIONS

The results of this study demonstrate that there are salient patterns of metaphoric language present in different computer education textual resources that may affect how students learn and understand computer programming concepts, as well as their understanding of their roles as programmers. Allowing students to experience a variety of metaphorical language used to communicate the role of the programmer could therefore aid students in understanding the multi-faceted aspects of computer programming, and their roles as programmers, both in relation to programming language and natural language. Consequently, I argue that the code itself, at least in the context of programming education, should be thought of as a second language. Just as different programming languages have different purposes, they also entail a set of approaches that will affect the way we think about programming (Table 11). Based on the writings of, for example, Lakoff and Johnson (1999), the varying use of metaphors implies that the cognitive processes surrounding computer programming (i.e., the connections between embodied experience and metaphors) will differ in relation to the context in which the programming occurs. Hence, the results of this study may have implications for informing the design of educational resources (e.g., software, hardware, books).

Furthermore, the results of the study raise general questions about how programming is communicated in the classroom. As this study does not concern programming as an activity per se, I argue that the results of this study motivate further research on, for example, teachers' metaphor use in the classroom (e.g., Larsson, Stolpe & Johansson Falck, 2021) . However, while application of the MIP method reported here can identify metaphorical structures in textual resources, investigating how programming concepts are structured in a real educational setting, requires analytical methods that focus on human interaction (e.g. Cienki, 2016; Gibbs, 2019; Müller, 2019). Such approaches would include how gestures and speech contribute to communication (e.g. Cuccio & Fontana, 2017; McNeill, 2008) where the interpretation of metaphorical expressions are foregrounded. Furthermore, questions on students' perspective taking and how the affordances provided by our surroundings affects the way we interact with our peers (Gibson, 2014) will be of interest. Such methods are currently being developed and explored in an expansion of the research reported in this study.

REFERENCES

Agirbas, A. (2018). The Use of Metaphors as a Parametric Design Teaching Model. Design and Technology Education: An International Journal, 23(1), 40-54.

Amin, T. G., Jeppsson, F., & Haglund, J. (2015). Conceptual Metaphor and Embodied Cognition in Science Learning: Introduction to special issue. *International Journal of Science Education, 37*(5-6), 745-758. doi:10.1080/09500693.2015.1025245

Becker, B. A. (2019). Parlez-vous java? bonjour la monde!= hello world: Barriers to programming language acquisition for non-native english speakers. In 30th workshop of the psychology of program-ming interest group-PPIG'19. Retrieved from: www. brettbecker. com/publications.

Blackwell, A. F. (2006). Metaphors we program by: Space, action and society in Java. *Proceedings of PPIG*, 7-21.

Boot, I., & Pecher, D. (2010). Similarity is closeness: Metaphorical mapping in a conceptual task. *Q J Exp Psychol (Hove), 63*(5), 942-954. doi:10.1080/17470210903134351

Bråting, K., & Kilhamn, C. (2021). The Integration of Programming in Swedish School Mathematics: Investigating Elementary Mathematics Textbooks. *Scandinavian Journal of Educational Research*, 1-16.

Caporael, L. R. (1986). Anthropomorphism and mechanomorphism: Two faces of the human machine. *Computers in Human Behavior, 2*(3), 215-234. doi:https://doi.org/10.1016/0747-5632(86)90004-X

Carroll, J. M., & Mack, R. L. (1985). Metaphor, computing systems, and active learning. *International journal of man-machine studies, 22*(1), 39-57.

Chibaya, C. (2019). *A Metaphor-Based Approach for Introducing Programming Concepts*. Paper presented at the 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC).

Cienki, A. (2016). Analysing metaphor in gesture: A set of metaphor identification guidelines for gesture (MIG-G). In *The Routledge handbook of metaphor and language* (pp. 149-165): Routledge.

Colburn, T. R., & Shute, G. M. (2008). Metaphor in computer science. *Journal of applied logic, 6*(4), 526-533.

Cuccio, V., & Fontana, S. (2017). Embodied Simulation and metaphorical gestures*,* 77.

Dreyfus, B. W., Geller, B. D., Gouvea, J., Sawtelle, V., Turpen, C., & Redish, E. F. (2014). Ontological metaphors for negative energy in an interdisciplinary context. *Physical Review Special Topics-Physics Education Research, 10*(2), 020108.

Forišek, M., & Steinová, M. (2012). *Metaphors and analogies for teaching algorithms*. Paper presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, USA. https://doi.org/10.1145/2157136.2157147

Gibbs, R. W. (2007). Idioms and formulaic language. In *The Oxford handbook of cognitive linguistics*.

Gibbs, R. W. (2015). Counting metaphors: What does this reveal about language and thought? *Cognitive Semantics, 1*(2), 155-177.

Gibbs, R. W. (2017). *Metaphor wars*: Cambridge University Press.

Gibbs, R. W. (2019). Metaphor as Dynamical–Ecological Performance. *Metaphor and Symbol, 34*(1), 33-44. doi:10.1080/10926488.2019.1591713

Gibbs, R. W., & Cameron, L. (2008). The social-cognitive dynamics of metaphor performance. *Cognitive Systems Research, 9*(1-2), 64-75.

Gibbs, R. W., & Colston, H. L. (2012). *Interpreting figurative meaning*: Cambridge University Press.

Gibson, J. J. (2014). *The ecological approach to visual perception: classic edition*: Psychology Press.

Gogolla, M., & Stevens, P. (2018). Teaching modeling in computer science as an ecosystem: a provocative analogy. *Computer science education, 28*(1), 5-22.

Grady, J. (1997). *Foundations of Meaning: Primary Metaphors and Primary Scenes*: eScholarship, University of California.

Grady, J. (2005). Primary metaphors as inputs to conceptual integration. *Journal of Pragmatics, 37*(10), 1595-1614. doi:10.1016/j.pragma.2004.03.012

Harrington, A. N. Hands-On Python A Tutorial Introduction for Beginners.

Hedberg, D., Haglund, J., & Jeppsson, F. (2015). Metaforer och analogier inom termodynamik i kemiläroböcker för gymnasiet. *NorDiNa: Nordic Studies in Science Education, 11*(1), 102-117.

Hidalgo-Cespedes, J., Marin-Raventos, G., Lara-Villagran, V., & Villalobos-Fernandez, L. (2018). Effects of oral metaphors and allegories on programming problem solving. In (Vol. 26, pp. 852-871).

Hurtienne, J., Stößel, C., Sturm, C., Maus, A., Rötting, M., Langdon, P., & Clarkson, J. (2010). Physical gestures for abstract concepts: Inclusive design with primary metaphors. *Interacting with Computers, 22*, 475-484 doi:10.1016/j.intcom.2010.08.009

Jeppsson, F., Haglund, J., Amin, T. G., & Strömdahl, H. (2013). Exploring the Use of Conceptual Metaphors in Solving Problems on Entropy. *Journal of the learning sciences, 22*(1), 70-120. doi :10.1080/10508406.2012.691926

Johnson, M. (2007). *The meaning of the body: Aesthetics of human understanding*. Chicago, IL: University of Chicago Press.

Kohn, T. (2017). *Variable evaluation: An exploration of novice programmers' understanding and common misconceptions*. Paper presented at the Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.

Kwon, K. (2017). Novice programmer's misconception of programming reflected on problem-solving plans. *International Journal of Computer Science Education in Schools, 1*(4), 14-24.

Lakoff, G. (2008). The Neural Theory of Metaphor. In *The Cambridge handbook of metaphor and thought* (pp. 17).

Lakoff, G., & Johnson, M. (1980). *Metaphors we live by*: Chicago : Univ. of Chicago Press, 1980.

Lakoff, G., & Johnson, M. (1999). *Philosophy in the flesh : the embodied mind and its challenge to Western thought*: New York : Basic Books, 1999.

Larsson & Stolpe (2022). Hands on programming: Teachers' use of Metaphors in gesture and Speech make Abstract concepts tangible *International Journal of Technology and Design Education (2022)* https://link.springer.com/article/10.1007/s10798-022-09755-0

Larsson, Stafstedt & Schönborn (2019). Heat Angels and Paper Cups: Pupils' Use of Metaphoric Relations When Engaging Thermal Cameras to Investigate Heat. *Contributions from Science Education Research book series (CFSE,volume 6)* https://link.springer.com/chapter/10.1007/978-3-030-17219-0_5

Larsson, Stolpe & Johansson Falck, (2021). A Teacher's Hands on Programming: How orientations of gestures provide concrete dimensions to abstract thoughts. *14th Conference of the European Science Education Research Association (ESERA 2021), , Braga, Portugal*

Long, J. (2007). Just For Fun: Using Programming Games in Software Programming Training and Education. *6*, 279-290.

Manches, A., McKenna, P. E., Rajendran, G., & Robertson, J. (2020). Identifying embodied metaphors for computing education. *Computers in Human Behavior, 105*, 105859. doi:https://doi.org/10.1016/j.chb.2018.12.037

McNeill, D. (2008). *Gesture and thought*: University of Chicago press.

Müller, C. (2019). Metaphorizing as Embodied Interactivity: What Gesturing and Film Viewing Can Tell Us About an Ecological View on Metaphor. *Metaphor and Symbol, 34*(1), 61. doi:10.1080/10926488.2019.1591723

Peelle, H. A. (1983). Computer metaphors: approaches to computer literacy for educators. *Computers & Education, 7*(2), 91-99.

Pierce, R., & Aparício, M. (2010). *Resources to support computer programming learning and computer science problem solving*. Paper presented at the Proceedings of the Workshop on Open Source and Design of Communication, Lisbon, Portugal. https://doi.org/10.1145/1936755.1936766

Pragglejaz Group. (2007). MIP: A method for identifying metaphorically used words in discourse. *Metaphor and Symbol, 22*(1), 1-39.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education, 13*(2), 137-172.

Skansholm, J. (2011). *C++ direkt* (3., [omarb.] uppl. ed.): Studentlitteratur.

Solomon, A., Bae, M., DiSalvo, B., & Guzdial, M. (2020). Embodied Representations in Computing Education: How Gesture, Embodied Language, and Tool Use Support Teaching Recursion.

Soulié, J. (2008). *C++ Language Tutorial*.

Stavrum, L. R., Bungum, B., & Persson, J. R. (2020). "Never at rest": developing a conceptual framework for descriptions of 'force'in physics textbooks. *Nordic Studies in Science Education, 16*(2), 183-198.

Stefanowitsch, A., & Gries, S. T. (2007). *Corpus-based approaches to metaphor and metonymy* (Vol. 171): Walter de Gruyter.

Stigberg, H., & Stigberg, S. (2020). Teaching programming and mathematics in practice: A case study from a Swedish primary school. *Policy Futures in Education, 18*(4), 483-496.

Sundström, J. (2013). *Programmering 1 med Python : lärobok* (3. uppl. ed.): Thelin läromedel.

Videla, A. (2017). Metaphors We Compute By. *Communications of the ACM, 60*(10), 42-45. doi:10.1145/3106625

Vieira, C., Magana, A. J., Roy, A., & Falk, M. L. (2019). Student explanations in the context of computational science and engineering education. *Cognition and Instruction, 37*(2), 201-231.

Woollard, J. (2005). The implications of the pedagogic metaphor for teacher education in computing. *Technology, Pedagogy Education, 14*(2), 189-204.