Metaphor Models in Software Education: An Empirical Study

Evgeny Pyshkin
University of Aizu

Tsuruga, Ikki-Machi, Aizu-Wakamatsu, Fukushima, 965-8580, Japan
Email: pyshe@u-aizu.ac.jp

Abstract—**This research contributes to the literature on using metaphors in computing and software education. We examine the major theories of metaphors focusing on linguistic, cognitive and communicational aspects of contemporary discourse on metaphor and the applicability of these theories to the domain of computing, software engineering and education. We investigate the specific characteristics of metaphors used in computer science and software systems and introduce a number of use cases demonstrating how metaphors are used in programming classes while discussing such topics as code organization, code readability, code aesthetics, and software project workflow.**

***Keywords*–Metaphor; software engineering; education; empirical; readability.**

I. Introduction

Exploring metaphors and their use in education received significant attention in research works in various fields of knowledge from philosophy and linguistics on one side of the spectrum to technology and engineering on the other. Convergence of models and approaches used in different subject domains becomes a noticeable trend in present-day technology-related cross-disciplinary research. In the last decade, we can cite a number of efforts to put software engineering and computing discourse into the context of human-centric paradigm [1], humanities [2], social and cognitive sciences [3]. Investigations on crossings between natural, social and technology disciplines [4], centricity of computer science in contemporary liberal arts education [5], digital disruption challenges [6], relationsips between digital humanities, digital society and software study [7] are of much interest for both humanity and engineering researchers.

In linguistics, metaphors are language constructs referring to (or reasoning about) the concepts using words and phrases with the meanings appropriate to different kinds of concepts [8]. Consider the famous William Shakespeare's fragment from the play "As You Like It" [9]: *"All the world's **a stage**, And all the men and women merely **players**: They have their **exits** and their **entrances**."* We find here a direct metaphor: "world as theater stage" using the connected concepts "players", "actor's exit (from the stage)" and "actor's entrance (to the stage)". Persy Bysshe Shelly uses a metaphor of family to describe the cloud, which is itself a metaphor of his romantic hero in the poem "The cloud" [10]: *"I am the **daughter** of Earth and Water, And the **nursling** of the Sky."*

In poetry, the metaphors of empathy are very common; here is one more good example from Paul Verlaine's "L'heure exquise" (1820), where the lune has voice, the willow has a silhouette, and the wind (not a willow!) weeps [11] (Table I shows the original text together with the English direct translation).

TABLE I. VERLAINE'S METAPHORS

Original French Text	*English Direct Translation*
De chaque branche	From every branch
Part une voix	A **voice** goes
Sous la ramé…	To under the ridge
La silhouette	**The silhouette**
Du saule noir	Of the black **willow**
Où le vent pleure…	Where the **wind weeps**…

Metaphors are mental phenomena that could be manifested not only in language, but also in gesture or graphic form; thus, in every form connected to the metaphor's (and human being's) cognitive nature and the metaphor's socio-cultural dimensions [12]. Metaphors do not only assert object similarities; paradoxically, they point up the dissimilarities and contrasts between the objects, and this is equally important for understanding how metaphors work [13].

Apart from linguistics and philosophy, there are numerous metaphor connotations in technology and engineering. Carbonelli, Sánchez-Esguevillas, and Carro pointed out the role of metaphors in understanding the emerging technologies: *"Technologies are not only changing our world in a materialistic and pragmatic way but they are a primary factor in defining our conceptual models, influencing the way we understand and perceive our experience"* [14]. Being a new reality, the technologies are often based on new concepts requiring metaphors for their understanding, such as:

- **"Data as resources"** metaphor in *Data Science* (derived from the earlier resource-based metaphors for electricity, time, transportation systems, etc.);
- **"Software as a construction material"** in *Software-defined Anything* (connected to the earlier metaphor of software design as architecture);
- **"Home as device container"** in *Smart Home technology* (closely related to the IoT metaphors).

Kendall mentioned that successful user metaphors have an impact on the development of successful information systems and their interfaces: *"Invoking a metaphor means opening the door for a listener to use all previous associations in entering the subject in different way"* [15].

In software engineering, requirement elicitation and initial system design need good metaphors in order to facilitate establishing communication between the development team and project stakeholders, to allow both parties interpreting and understanding their languages (see Figure 1). We use the term "languages" in a broader sense, to designate the process of mapping the conceptual core connected to the subject domain to the model used on developer's side.

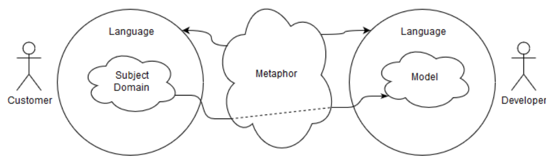

Figure 1. Mapping the customer's domain language to the development team's language.

The remaining text is organized as follows. In Section II, the existing theories of metaphor are introduced at a glance with respect to their mutual dependencies. Section III is focused on using the theories of metaphor in education, with particular emphasis on their applicability to technology disciplines. Section III presents our own experience on introducing metaphors to students attending a programming class.

II. Metaphor Paradigms in Historical Perspective

It is no exaggeration in saying that modern theories of metaphor appeal to a number of ancient views, where the works of Aristotle and Quintilian are the most discussed. Multiple hypotheses of contemporary linguistics are built on revealing the insights and finesse of the analysis of ancient authors [16]. Traditionally, the sources of so-called **Comparison Theory** are attributed to Aristotle, who introduced a metaphor as the *application (ἐπιφορά) of an alien name by transference either from genus ()γένος) to species (εἶδος), or from species to genus, or from species to species, or by analogy* [17]. In contrast to many modern studies of metaphor insisting that Aristotle undervalued metaphor and believed it to be a solely ornamental language feature, Mahon argued that Aristotle held a position on the *ubiquity of metaphor* which supports current views about the omnipresence of metaphors in everyday discourse [18]. Wood claimed that the Aristotle's definition contributes to the relationships between concepts and the processes of metaphor application (thus, from the perspective of software engineering, it is in direction of entity-relationship modeling) [19].

Quintilian, in turn, used a "process-oriented" approach, but emphasized the alteration, or mutation, rather than transferring [20]. Quintilian stated that the alterations arise from the words used metaphorically, and the involved changes *"concern not merely individual words, but also our thoughts and the structure of our sentences"* [21]. Thus, the Quintilian's approach may be considered as a precursor of **Cognitive Theory of Metaphor** by Lakoff and Johnson [22].

Richards introduced, and then Black developed the **Interaction Theory of Metaphor**. In contrast to ancient authors who worked with the transitional concepts of source and target, Richards introduced the technical concepts of **tenor** and **vehicle**, where the former is the thought being described in terms of another (metaphorically), while the latter is the thought, in terms of which the tenor is described [23]. Black described it in his conceptual book "Models and Metaphors": *"A memorable metaphor has the power to bring two separate domains into cognitive and emotional relation by using language directly appropriate to the one as a lens to seeing the other; the implications, suggestions, and supporting values entwined with the literal use of the metaphorical expression enable us to see a new subject matter in a new way. [...] the metaphor itself neither needs nor invites explanations and paraphrase. Metaphorical thought is a distinctive mode of achieving insight, not to be construed as an ornamental substitute for plain thought"* [24]. Thus, metaphors are linked to ontological models connected to the tenor and vehicle.

Lakoff created a **Contemporary Theory of Metaphor** focused on examination of metaphors as not solely language entities, but matters of thought and reason: *"the locus of metaphor is not in language at all, but in the way we conceptualize one mental domain in terms of another. The general theory of metaphor is given by characterizing such cross-domain mappings"* [25]. Such a conceptualization is delivered via finding and creating the ontological correspondences between the target and source domains. There are subject matters that cannot be comprehended, without using metaphors, even in everyday conversational language. Many conceptual metaphors are cross-language metaphors. The famous metaphor **"Love as Journey"** can be described in different languages without losing much of its metaphoric contents. This makes this example extremely successful. Vocabulary related to a journey serves as a frequent source for metaphors used in different knowledge areas. In project management, for example, we use **milestones** to designate the important project stages, **tickets** – to name the tasks assigned to engineers that should be completed by the deadline, project **roadmaps** – to name an overview of the project's goals and deliverable artifacts presented within a project **timeline**, etc.

Steen extended the preceding theories by adding a third, communicative, dimension. He pointed out that though Lakoff's cross-domain mappings may have been required in the history of language and its understanding, *"these mapping have become irrelevant to the thought processes of the contemporary language user, precisely because the metaphorical senses of the words have become equally conventional, and sometimes even more frequent that the non-metaphorical ones"* [26]. Specifically, in technology disciplines, there are frequent cases, when the professional language becomes non-metaphorical, even if many concepts were originally defined using the metaphor constructions, but which are not presently considered as a deliberate use of metaphors. There is also a kind of ontology deformation: when metaphors of **files**, **folders** and **directories** were used to name the interface elements in computer systems (first, command-line, and later – graphical), many people were able to understand the cross-domain mapping between the abstractions of computer storage organization and the concrete example of stationery items. Nowadays, for younger generations, these interface names are not abstract anymore; some have never seen physical files or folders. Thus, they use these words without thinking of their initial metaphorical connotations.

In terms of comparative theory, the source and the target may change roles. In the early years of Internet technology, the concept of electronic mail was explained by mapping the electronic message domain to a traditional letter mail. At that time, people had to explicitly emphasize the fact of sending an electronic message, not a traditional one: "e-mail me", "check your e-mail", etc. Nowadays, when most of communications have been transferred to the domain of computer (e.g., (still) electronic) systems, this "e" prefix became unnecessary and almost disappeared. By saying "mail me", we rather mean sending an electronic message, not a traditional mail. Colburn and

Shute give the following explanation of the above-mentioned phenomena: *"when the target domain becomes so dissimilar to the source through information enrichment that a metaphorical name for the target concept ceases to be metaphorical and becomes a historical artifact"* [27]. Because of technology, there could be even more unobvious cases, when at attempt to use a particular metaphor in its literal sense may generate a metaphorical connection "in opposite direction": nice example is the 2006 American romantic drama by Alejandro Agresti "Lake House", where the heroes used a physical mail box for operations shifted in time. Such operations are possible and even normal for electronic communications (nicely working for the heroes), but seems surrealistic while moved to the physical (neither electronic or virtual) reality.

III. Using Metaphors in Education

A. From Language Learning to Education in Broad Sense

In language learning, using metaphors is natural part of introducing new lexical material to learners. Metaphors provide a convenient approach to enhance and organize the learner's vocabulary, as well as to group together the words and synthetic concepts having a metaphorical meaning. Researching a metaphorical use of language constructions within a particular topic may enhance the vocabulary related to the mapped topics. In [28], Lazar gives a number of examples such as using body vocabulary to describe the locations (*in the heart of the city*, *on the foot of a mountain*) or weather vocabulary to describe the human relations (*a warm welcome*, *to freeze somebody out*).

Even the teaching process itself can be described metaphorically with respect to the teacher's roles and responsibilities. Clarken introduced 5 (perhaps, not exhaustive) teaching metaphors: *teachers as parents*, *teachers as gardeners*, *teachers as prophets*, *teachers as pearl oysters*, and *teacher as physicians* [29]. A teacher may operate differently by using different metaphors in different time; finding an appropriate teaching model is an important aspect of making the teaching process efficient and learner-friendly.

B. Metaphors in Computing and Software

Computer science and software metaphors are diverse and multi-faceted, they actively exploit different theories of metaphors (including linguistic, cognitive and communication approaches): they all may be concurrently used and cooperate.

Research on the particularities of technology language does not concern the corresponding technological or industrial applications only, but society at large. The aspects of technological and engineering literacy are important for improving educational practices in many areas of knowledge: *"informed citizens need an understanding of what technology is, how it works, how it is created, how it shapes society, and how society influences technological development"* [30]. The language of a particular technology-sensitive domain (such as software engineering) is not only for the domain professionals anymore: all members of contemporary society need a better understanding of this language and its metaphors. What makes software metaphors particularly complex is that they have connotations to abstract entities that we could not physically touch or point to. Johnson describes computer abstractions as *"based not on nature but rather on artificial world created by humans"* [31].

Software architecture exploits the **construction** metaphor with the list of relevant terms such as process **building**, **architectural pattern**, etc. In turn, an appropriate metaphor may improve the process of designing and describing software architectures. The architectures could not be designed only by a group of software engineers, they need more experts. That is why Smolander defined four metaphors referring to the different meanings of architecture, its description and stakeholder environment [32]:

- **Architecture as blueprint** describing the high-level implementation of the system;
- **Architecture as literature** describing the project documentation;
- **Architecture as language** describing the common understanding about the system structure and communication between different stakeholders;
- **Architecture as decision** describing the decisions about the system structure, the required resources and development strategies.

In education, metaphors help to introduce the unknown by using **concrete** examples explaining **abstract** things [29]. However, understanding of what is concrete and what is abstract differs between disciplines. Abstraction in computer science is not the same as in mathematics and linguistics [27]: computer scientists (rightfully) believe that an application control stack is a concrete entity, and its complexity can be explained better with using the inferential structures of abstract domains (like queuing). However, for others, the concept of memory stack seems to be a complete abstraction.

Software works with many abstract concepts, which are largely metaphorical. According to Boyd, software can be considered as a special case of fiction literature, that is why it is essentially metaphorical [33]. The approaches we use to describe the data entities and control structures, memory organization and program workflow, structural patterns and architecture designs actively exploit various metaphors. Some of such metaphors are listed in Table II.

Interestingly, introducing a metaphor to a software domain may lead to further extension of these metaphor within the bag of concepts specific for software design. Figure 2 provides an illustration: a design pattern, describing the object-oriented structure of instance creators, used a metaphor of factory borrowed from the domain of industrial technology.

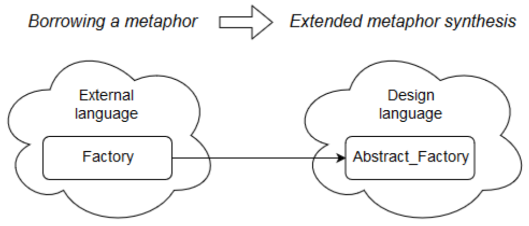

Figure 2. Synthesis of extended metaphor.

A **Factory Method** is for creation instances. It doesn't have compile-time dependencies on the object's type. In turn, for a given set of related interfaces, an **Abstract Factory** provides a way to create objects that implement those interfaces for a matched set of concrete classes (for example, while supporting changing platform's look and feel by selecting an alternative set of widgets as shown in Figure 3).

TABLE II. EXAMPLES OF SOFTWARE METAPHORS

Domain	*Examples*
Program objects	Scope Assignment Lifetime Location
Control structures and program workflow	Selection Loop Thread Lazy computation
Modular structure	Library Package
Interface design	Menu Palette Folder File
Storage containers	Array Map List Queue Tree
Structural and design patterns	Factory method Bridge Observer Visitor
Design process	Waterfall Evolution Agile
Software analysis	Bad smell Refactoring Test mutation
Project flow	Maturing Degradation Evolving

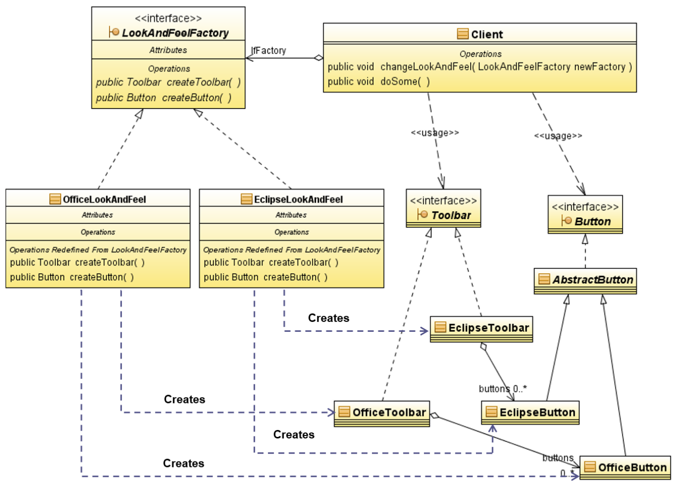

Figure 3. Abstract factory and "Look and Feel" Metaphor.

Indeed, "Abstract factory" could hardly be imagined in real world, however, in computer architectures, it provides a concrete (not abstract!) model of class structure representing the object creation subsystem of extensible and interchangeable sets of multiple object types that should function in a way that is independent on the specific types they are working with.

IV. Using Metaphors in the Programming Class: Study of Experience

Metaphors in education are helpful for many reasons. First, to link students' knowledge to newly introduced concepts and models [34] (experience-based metaphors). Second, to name new concepts in a way we can understand them by using similarity between the source and target domains (comparative metaphors). Finally, they can exploit the ontology mapping (ontological and interactive metaphors).

A. *Software Code Organization Metaphors*

Tomi and Mikko Difva introduced a number of metaphors used in the programming class for beginners that help to understand the different views on code structuring [35]. They defined nine metaphors: **machine**, **organism**, **brain**, **flux and transformation**, **culture**, **political system**, **psychic prison**, **instrument of domination**, and **carnival**. For example, the **Machine** metaphor is used to introduce a code as a sequence of commands, thus, referring an imperative programming paradigm; the **Organism** metaphor is used to introduce a code as a collection of interacting objects, thus, referring to object-oriented models, etc. Such metaphors may be very helpful in discussion on why the different views on system organization are required, and how a particular development process reflects a particular software development approach. Their suggestions are very interesting, but probably need further adjustments. For example, a sequential process probably needs another metaphor, not "Machine", since the contemporary understanding of this concept is more complex: Frank, Roerhrig and Pring define a machine as a **system of intelligence** combining software, hardware and user input. Such a machine is aimed not only at performing a series of control commands, but at improving on its own over time [36].

B. *Form and Contents as a Readability Metaphor*

In my programming class, I sometimes organize an exercise entitled *"The Form inside the Work"*, which is about discussing visual metaphors for introducing multi-faceted software concepts. In particular, we discuss how the code readability concept may be metaphorically expressed and analyzed using the famous artwork masterpieces (see the example of using Van Eyck's "Annunciation" for such an exercise in [37]).

As pointed out by Oosterman et al. in [38], artworks (compared to the photography or textual artifacts) provide less and often inconsistent visual information being an abstract, symbolic or allegorical (often metaphorical) interpretation of reality; therefore, their exact reading and annotation is a challenging problem. Nonetheless, the artworks are still readable, though such readings might naturally give various interpretations. Similarly to literary works, manner and matter are mutually dependent in visual works [39]: structures used by a creator in an artwork (the form) provide the ways making possible the reproduction of creator's intentions, metaphors and messages (the contents) in the beholder's mind. This reproduction implements the artist's program approximately, thus, giving space for many interpretations that can be considered as co-creation acts [40].

Let me illustrate this idea by using the iconic Rembrandt's chez-d-oeuvre "The Night Watch". Rembrandt Harmenszoon van Rijn's "The Night Watch" ("Militia Company of District II under the Command of Captain Frans Banninck Cocq", 1642, Amsterdam Museum on permanent loan to the Rijksmuseum, Amsterdam, Netherlands) is an exceptional example of huge multi-layer composition portraying a military group. Full of metaphoric symbolism, this masterpiece provides an excellent case to learn "painting reading". Figure 4 graphically demonstrates a possible interpretation.

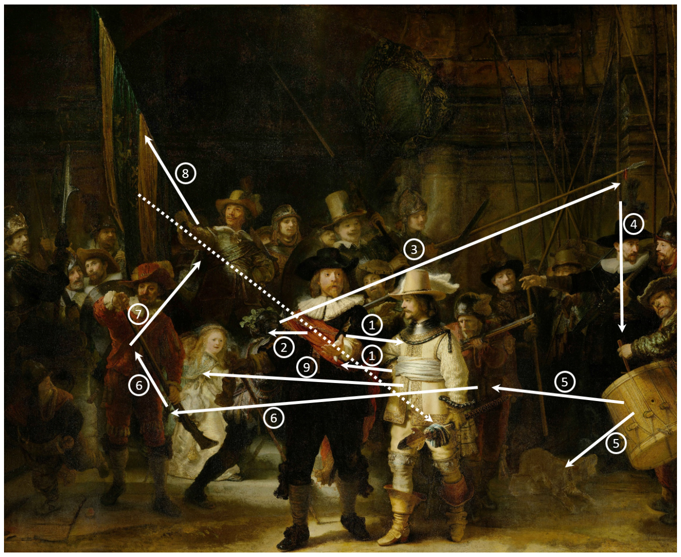

Figure 4. Possible reading links inside "The Night Watch" by Rembrandt.

As noted by Oliveira in [41], the canvas has visible multilayer structure. "Departing" from the two central characters (representing a cooperation between Protestant and Catholic parties), an eye may follow different ways, but the most likely directions are implicitly embedded inside the picture: a trained soldier close to the captain in the second layer (Figure 4, link 2) demonstrating his shooting skills by cutting the strip of a spear (link 3). The strip virtually points to the drummer (link 4) calling of arms. Close to the drummer, just behind the Catholic lieutenant, we see an aged volunteer and a distressed dog (links 4). From the "drummer's group" the eye moves to the left part of the composition (links 6) with an experienced soldier, making a compositional balance with the less experienced volunteer on the right side. Moving up to the back stage layer, a beholder's attention is caught by the figure of the man (link 7) holding the national flag (link 8). The flag colors call up the similar colors of the military baton held by the Catholic lieutenant (dashed line). In turn, the light-colored figure of the lieutenant is symbolically linked to the bright woman's figure (link 9), being one of the most discussed character of this painting work (Oliveira suggests she is a symbolic interpretation of the motherland).

Of course, the above presented rendition is not the only possible way to rediscover rich symbolism of Rembrandt's masterpiece, which has much more symbolic allusions and enigmatic elements (their detailed analysis is naturally out of scope of this paper). Nevertheless, it clearly shows that the possibilities are somehow "programmed" by the author, though the exact links are not represented. This consideration needs to go back to Aristotle's *Poetics*, where the cognitive meaningfulness of metaphor was emphasized: metaphors require an act of recognition and interpretation from the recipient [17].

In the case of software programs, it is commonly agreed that the code graphics, organization and legibility can be considered as essential aesthetic properties [42]. Meanwhile, the aesthetic values are connected to the quality properties, as it is nicely pointed out by Edmonds: *"if the resulting code is like spaghetti [. . .], it is not highly rated even if it performs its functions perfectly"* [43]. That is why the exercise described here can be considered as a small effort to compensate for relatively minor attention to the problems of understanding programming style and readability in software engineering curricula. Learning parallels between software engineering and art education give interesting insights to improving developers' culture (where, by the way, "culture" can be understood both literally and metaphorically).

In the research presented in [44], the authors describe the empirical code annotation study. They come to a conclusion, which is in partial contradiction to common practices in programming teaching: the source code comments (being an explicit way to explain the meaning of commented solution) affect the notion of code readability only moderately. Although the comments provide the very direct way of communication between the software writer and its reader, the code readability may be increased mostly because of improving the code organization and the used models (i. e., the code properties which do not provide a direct communication intent), and not because of increasing the number of detailed comments.

V. Conclusion

On the basis of analysis of linguistics and cognitive science original sources, this study examines the diversity and multifacetedness of the metaphor concept and its important role in technology and engineering areas. Since the particular focus of this research is on software education, we address a number of practical cases of using metaphors in the programming class by including a brief review of architectural software metaphors, metaphors of code organization, code readability and code aesthetics metaphors.

The above-mentioned cases do not exhaust the rich possibilities of using and exploring metaphors in class-time teaching scenarios. Hence, I expect that further extension of this study in cooperation with the experts from both liberal arts and technology domains might be of significant interest for teaching and research communities. Based on the wide contemporary discourse, we need more systematic analysis of metaphors used in software engineering, in order to classify the published approaches and to make them better visible and shareable among the members of academic community. Learning metaphors is connected to the development of soft skills, which are nowadays considered an important aspect of software engineering education. However, the evaluation whether the use of metaphors significantly improve the learning process still remains an open issue; the empirical analysis of benefits and potential results isn't trivial.

There are many important aspects, which, being out of scope of this paper, requires much attention. These aspects include software visualization and visual metaphors, metaphors of software architectures and design pattern metaphors, software code transformation and restructuring metaphors (such as code smells and refactoring), cooperation and mutual dependencies of different theories of metaphor in their application to technology domains, and transition of metaphors introduced in technology domains back to the non-technology areas.

Acknowledgement

I would like to thank Professor John Blake from the Center for Language Research of the University of Aizu who kindly read the earlier versions of this study and made very valuable comments.

Many thanks to the anonymous IARIA reviewers for their very constructive suggestions on making more focused and compact concretization of basic ideas described in the initial submission.

The work is supported by the University of Aizu Research Funding.

References

[1] M. Niemelä et al., "Human-driven design: a human-driven approach to the design of technology," in IFIP International Conference on Human Choice and Computers. Springer, 2014, pp. 78–91.

[2] G. C. Murphy, "Human-centric software engineering," in Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 251–254.

[3] E. Pyshkin, "Liberal arts in a digitally transformed world: Revisiting a case of software development education," in Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia, ser. CEE-SECR '17. New York, NY, USA: ACM, 2017, pp. 23:1–23:7.

[4] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in Guide to advanced empirical software engineering. Springer, 2008, pp. 285–311.

[5] D. Baldwin and A. Brady, "Guest editors' introduction: Computer science in the liberal arts," Trans. Comput. Educ., vol. 10, no. 1, Mar. 2010, pp. 1:1–1:5.

[6] V. Davidovski, "Exponential innovation through digital transformation," in Proceedings of the 3rd International Conference on Applications in Information Technology, ser. ICAIT'2018. New York, NY, USA: ACM, 2018, pp. 3–5.

[7] F. Frabetti, "Have the humanities always been digital? For an understanding of the digital humanities in the context of originary technicity," in Understanding digital humanities. Springer, 2012, pp. 161–171.

[8] J. H. Martin and D. Jurafsky, Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Pearson/Prentice Hall Upper Saddle River, 2009.

[9] W. Shakespeare, As You Like It. Edward Blount and William and Isaac Jaggard, London, 1623.

[10] P. B. Shelley, The Cloud. Charles and James Ollier, London, 1820.

[11] P. Verlaine, L'heure exquise. Creuzevault, 1936.

[12] L. Cameron, "Operationalising 'metaphor' for applied linguistic research," Researching and applying metaphor, 1999, pp. 3–28.

[13] R. M. Weaver and R. S. Beal, A rhetoric and handbook. Holt, Rinehart and Winston, 1967.

[14] J. Carbonell, A. Sánchez-Esguevillas, and B. Carro, "The role of metaphors in the development of technologies. The case of the artificial intelligence," Futures, vol. 84, 2016, pp. 145–153.

[15] J. E. Kendall and K. E. Kendall, "Metaphors and their meaning for information systems development," European Journal of Information Systems, vol. 3, no. 1, 1994, pp. 37–47.

[16] M. Armisen-Marchetti, "Histoire des notions rhétoriques de métaphore et de comparaison, des origines à quintilien," Bulletin de l'association Guillaume Budé, vol. 49, no. 4, 1990, pp. 333–344.

[17] S. H. Butcher, The poetics of Aristotle edited with Critical Notes and a Translation. Macmillan, 1902.

[18] J. E. Mahon, "Getting your sources right," Researching and applying metaphor, 1999, pp. 69–80.

[19] M. S. Wood, "Aristotle and the question of metaphor," Ph.D. dissertation, Université d'Ottawa/University of Ottawa, 2015.

[20] A. Novokhatko, "The linguistic treatment of metaphor in quintilian," Pallas, vol. 103, 2017, pp. 311–318.

[21] H. E. Butler et al., The Institutio Oratoria of Quintilian. Harvard University Press, 1922, vol. 4.

[22] G. Lakoff and M. Johnson, "Metaphors we live by," Chicago, IL: University of, 1980.

[23] I. A. Richards and J. Constable, The philosophy of rhetoric. Oxford University Press New York, 1965, vol. 94.

[24] M. Black, "Models and metaphors: Studies in language and philosophy." 1962.

[25] G. Lakoff, "The contemporary theory of metaphor," 1993, retrieved: Sep, 2019. [Online]. Available: https://escholarship.org/uc/item/54g7j6zh

[26] G. J. Steen, "The contemporary theory of metaphor – now new and improved!" Review of Cognitive Linguistics. Published under the auspices of the Spanish Cognitive Linguistics Association, vol. 9, no. 1, 2011, pp. 26–64.

[27] T. R. Colburn and G. M. Shute, "Metaphor in computer science," Journal of Applied Logic, vol. 6, no. 4, 2008, pp. 526–533.

[28] G. Lazar, "Exploring metaphors in the classroom," Teaching English, 2006.

[29] R. H. Clarken, "Five metaphors for educators," 1997.

[30] J. Krupczak et al., "Defining engineering and technological literacy," Philosophical and Educational Perspectives in Engineering and Technological Literacy 3, 2012, p. 8.

[31] G. J. Johnson, "Of metaphor and difficulty of computer discourse," Communications of the ACM, vol. 37, no. 12, 1994, pp. 97–103.

[32] K. Smolander, "Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice," in Proceedings International Symposium on Empirical Software Engineering. IEEE, 2002, pp. 211–221.

[33] N. Boyd, "Software metaphors," 2003, retrieved: Sep, 2019. [Online]. Available: https://pdfs.semanticscholar.org/deee/512ab8b7a3753fda248fe99780e3470e6881.pdf

[34] I. N. Umar and T. H. Hui, "Learning style, metaphor and pair programming: Do they influence performance?" Procedia-Social and Behavioral Sciences, vol. 46, 2012, pp. 5603–5609.

[35] T. Dufva and M. Dufva, "Metaphors of codestructuring and broadening the discussion on teaching children to code," Thinking Skills and Creativity, vol. 22, 2016, pp. 97–110.

[36] M. Frank, P. Roehrig, and B. Pring, What to do when machines do everything: How to get ahead in a world of AI, algorithms, bots, and Big Data. John Wiley & Sons, 2017.

[37] E. Pyshkin, "Designing human-centric applications: Transdisciplinary connections with examples," in Cybernetics (CYBCONF), 2017 3rd IEEE International Conference on. IEEE, 2017, pp. 1–6.

[38] J. Oosterman, J. Yang, A. Bozzon, L. Aroyo, and G.-J. Houben, "On the impact of knowledge extraction and aggregation on crowdsourced annotation of visual artworks," Computer Networks, vol. 90, 2015, pp. 133–149.

[39] J. G. McElroy, "Matter and manner in literary composition." Modern Language Notes, 1888, pp. 29–33.

[40] D. Likhachev, "Neskolko mysley o netochnosti iskusstva i stilistich-eskikh napravleniyakh," in Philologica. Issledovaniya po yazyku i literature, 1973, pp. 394–401, (Some ideas about uncertainty of arts and stylistic trends – In Russian).

[41] P. M. Oliveira, "The Dutch company," retrieved: Aug, 2019. [Online]. Available: https://www.academia.edu/8579003/_Eng_The_Dutch_Company

[42] S. Gruner, "Problems for a philosophy of software engineering," Minds and Machines, vol. 21, no. 2, 2011, pp. 275–299.

[43] E. Edmonds, "The art of programming or programs as art," Frontiers in Artifical Intelligence and Applications, vol. 161, 2007, p. 119.

[44] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," IEEE Transactions on Software Engineering, vol. 36, no. 4, 2009, pp. 546–558.