DOI: 10.18721/JHSS.11305
УДК 81-004

A METAPHORIC BRIDGE: UNDERSTANDING SOFTWARE ENGINEERING EDUCATION THROUGH LITERATURE AND FINE ARTS

E. Pyshkin, J. Blake

University of Aizu,
Aizu-Wakamatsu, Japan

This research contributes to the literature on understanding software engineering education from the perspective of liberal arts through the discussion on the pedagogic application of metaphors to convey complex concepts by drawing on the attributes of concrete known concepts (often – from different knowledge domains). Metaphors are defined and examples from everyday life and literature are used to contextualize their functionalities and possible application mechanisms. A chronology of the major theories of metaphors focusing on linguistic, cognitive and communicative aspects of contemporary discourse sets the theoretical background. To understand how metaphors can help educators, their major functions are clarified using examples from software engineering and computing. The plethora of technical metaphoric expressions is evidence of their importance in informatics and computer technology education. We describe several use cases of harnessing visual metaphors from the fine arts to teach programming and data management classes to computer science majors. These demonstrate how metaphors can be used while discussing such topics as code organization, code readability and modifiability, code aesthetics, software versions, and software project workflow. There appears to be a trend to using metaphors in the sciences and new technology domains, and given their ability to convey meaning and bridge terminology gaps, we argue this should be encouraged and further work carried out.

Keywords: metaphor, software engineering, education, programming, literature, liberal arts, fine arts.

Citation: E. Pyshkin, J. Blake, A metaphoric bridge: Understanding software engineering education through literature and fine arts, Society. Communication. Education, 11 (3) (2020) 59–77. DOI: 10.18721/JHSS.11305

СОЕДИНЯЯ МЕТАФОРЫ: ИНТЕРПРЕТАЦИЯ ПОДХОДОВ К ОБРАЗОВАНИЮ В ОБЛАСТИ ИНЖЕНЕРИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЧЕРЕЗ ПРИЗМУ ЛИТЕРАТУРЫ И ИЗОБРАЗИТЕЛЬНОГО ИСКУССТВА

Е.В. Пышкин, Дж. Блейк

University of Aizu,
Aizu-Wakamatsu, Japan

Статья развивает исследования об интерпретации образовательных технологий в инженерии программного обеспечения в контексте гуманитарных наук и, в частности, о роли метафоры в передаче сложных концепций посредством переноса атрибутов известных понятий (часто заимствуемых и переносимых из других областей знания). С помощью примеров метафор как из повседневной жизни, так и из художественной и научной литературы, предлагается подход к изучению их функциональных характеристик и возможных механизмов применения. Для представления контекста работы, приводится краткий исторический обзор развития основных теорий метафоры, при этом основное внимание уделяется лингвистическому, когнитивному и коммуникативному аспектам современного дискурса в данной области. Формулируются ос-

новные функции метафоры, важные для образовательной практики; эти функции иллюстрируются примерами из областей, связанных с инженерией программного обеспечения и компьютерных технологий в более широком смысле. Огромное количество технических понятий, основанных на метафоре, является очевидным подтверждением важности изучения механизмов применения метафоры в области информатики и компьютерных технологий. Несколько сценариев, используемых одним из авторов при подготовке курсов программирования и управления данными, служат иллюстрациями возможностей переноса отдельных визуальных метафор изобразительного искусства для демонстрации и объяснения студентам основных концепций, связанных с организацией программного кода, возможностью его восприятия читателем, эстетическими характеристиками компьютерных программ, модифицируемостью кода, управлением версиями и организацией процессов управления проектами. Дальнейшая работа по изучению способности метафор к передаче значения новых понятий и устранению терминологических пробелов может рассматриваться как перспективное направление исследований в рамках существующих тенденций анализа использования метафор в науке и новых технологических областях.

Ключевые слова: метафора, инженерия программного обеспечения, образование, программирование, литература, гуманитарная наука, изобразительное искусство.

Ссылка при цитировании: Пышкин Е.В., Блейк Дж. Соединяя метафоры: интерпретация подходов к образованию в области инженерии программного обеспечения через призму литературы и изобразительного искусства // Общество. Коммуникация. Образование. 2020. Т. 11. № 3. С. 59–77. DOI: 10.18721/JHSS.11305

I. Introduction

This article is an extended and significantly revised study based on the conference paper [1]. Metaphors have received significant attention in the general education literature [2, 3] and within disciplinary-specific education ranging from linguistics and philosophy in the humanities [4] through to cognitive sciences [5] and software engineering [6–10].

This paper shows how metaphors can be harnessed in computing and programming courses to convey complex meanings to a lay audience in a time-efficient manner and introduce fine arts into the curriculum, expanding the cultural horizons of computer science majors. It also adds to the body of literature that seeks to understand, explore and explain both the function and application of metaphors in educational contexts. The ubiquity of metaphors is discussed and illustrated drawing on examples from a range of domains within computing and software engineering. The pedagogic functions of metaphors are elucidated and exemplified.

Metaphors are figurative, helping people to describe the world in a non-literal way. Stating that "Andy is **a fox**" does not mean Andy is not human, but ascribes some quality of a fox to Andy. Foxes are known for being sly, and so Andy is portrayed as being sly. Linguists describe metaphors as language constructs referring to, or reasoning about, concepts using words and phrases with the meanings appropriate to different kinds of concepts [11]. In short, stating that "A is B" assigns the one or more features of B to A. For example, we can assign people the characteristics of various animals.

*Bianca is **a fish**. Charlie is **a dinosaur**.*

These metaphors enable us to work out who is a talented swimmer and who is a technophobe. Metaphors can be used for inanimate or abstract nouns:

*Diana has opened **Pandora's box** by mentioning the crisis.*

*The crisis is a real **can of worms**.*

Pandora's box is a mythical artifact that when opened releases unexpected problems. For those less familiar with classical studies, a can of worms has the same meaning.

Literature and the fine arts are rich hunting grounds for metaphors. Consider the famous fragment from William Shakespeare's play "As You Like It" [12]:

*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" [13]:

*I am the **daughter** of Earth and Water, And the **nursling** of the Sky.*

In poetry, metaphors of empathy are very common. Paul Verlaine's "L'heure exquise" (1870) is an excellent example, where the lune has voice, the willow has a silhouette, and the wind (not a willow!) is weeping [14]. As you can see in Table 1, the original metaphors used in the original French text and English translation are replaced by new ones in the Bryusov's interpretation in Russian.

Metaphors are mental phenomena connected to cognition and socio-cultural dimensions [16], and manifest not only verbally but visually. Metaphors usually compare similarities but, paradoxically, can be used to contrast differences, which is equally important for understanding how metaphors work [17]. Metaphors can fulfil numerous pedagogic functions from categorizing elements, conveying complex concepts, or as *memoria technica* [18, 19].

Table 1. Verlaine's metaphors

Original French (Paul Verlaine)	English Translation by Richard Stokes [15]	Russian Version by Valery Bryusov
La lune blanche Luit dans le bois ; De chaque branche Part une voix Sous la ramée…	The white moon Gleams in the woods; From every branch There comes a **voice** Beneath the boughs…	И месяц белый В лесу горит, И зов несмелый С ветвей летит Нас достигая…
L'étang reflète Profond miroir La silhouette Du saule noir Où le vent pleure…	The pool reflects, Deep mirror, **The silhouette** Of the black **willow** Where the **wind is weeping**…	Там пруд сверкает Зеркальность вод Он отражает Весь хоровод Кустов прибрежных…

The remaining text is organized as follows. In Section II, ancient and modern theories of metaphor are introduced. The functionalities of metaphors are analyzed in Section III. The role of metaphors as vehicles to convey complex concepts and bridge language gaps is explained and exemplified. Section IV describes the omnipresence of metaphors and lists illustrative examples from various domains in computing and software engineering. Section V presents a case study on the use of visual metaphors from the fine arts in a programming class. In the conclusion, we summarize the arguments for drawing on metaphors in the classroom, reflect on the case study and call for more research on the use of metaphors.

II. Theoretical perspective

A. Ancient theories

Metaphors have been the object of scholarly discussion for over two millennia [20]. 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* (ἰδος), *or from*

species to genus, or from species to species, or by analogy [21]. Aristotle described metaphors as *"giving the thing a name that belongs to something else"* [21] and defined metaphors as a transference process. Thus, metaphors have two senses: the first concrete sense is the concept in its transferred form, while the second abstract sense is the process of transferring [22]. In contrast to many modern studies of metaphor insisting that Aristotle undervalued metaphor and believed it to be a solely ornamental language feature, Wood claimed that the Aristotle's definition contributes to the relationships between concepts and the processes of metaphor application [22].

Quintilian used a "process-oriented" approach, but emphasized the alteration, or mutation, rather than transferring [23], *"a metaphor is the commonest and by far the most beautiful artistic alteration of a word, thus, distinguishing the language in which it is embedded"* [24]. Quintilian stated that the alterations arise from words used metaphorically, and the involved changes *"concern not merely individual words, but also our thoughts and the structure of our sentences"* [24]. Thus, Quintilian's approach may be considered as a precursor of **Cognitive Theory of Metaphor** by Lakoff and Johnson [25].

B. Modern theories

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 [26]. To quote Black, *"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"* [27]. Thus, despite some differences, both Richards and Black talked about metaphors that cannot be explained on the basis of similarity principles [28]: for example, "round wine" is not similar to a geometric shape in any manner, but probably exploits a virtual property of round shape's perfectness. Thus, metaphors are linked to ontological models connected to tenor and vehicle. In [29], there is example of using the term "hedging" (which commonly refers to risk management in economics) in its application to linguistics, to designate a communicative strategy aimed to express speaker's uncertainty and suggest a possibility of refutation [29].

Lakoff created the **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"* [30]. Such a conceptualization is delivered via finding and creating the ontological correspondences between the target and source domains. Some subject matter can only be explained using metaphors, even in everyday conversational language.

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"* [31].

Specifically, in technological disciplines, there are frequent cases of professional language becoming non-metaphorical. 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 knowledge of their initial metaphorical connotations. 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", which became "mail me". As email is now the default, the "e" prefix is no longer needed.

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"* [32].

III. Four functions of metaphors

Metaphors are used to get across difficult ideas using known examples. This enables those who are more experienced or expert users to explain abstract ideas to a lay audience or novice users. Educators frequently make use of these functionalities.

A. Mapping domains

As has already been elucidated, metaphors provide a way to map attributes from one element to another. In project management, for example, we draw on the terminology of journeys and 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.

B. Reifying abstract ideas

Complex ideas are axiomatically more difficult to understand than simple ideas. In general, **abstract** ideas are more complex than **concrete** items. Reification is the transformation of an abstract idea into a concrete tangible idea. Metaphors can act as concrete examples of the abstract concepts [33], facilitating the transformation and increasing the likelihood of comprehension. However, the 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 [32]: computer scientists (rightfully) believe that an application control stack is a concrete entity, and its complexity can be explained better by using the inferential structures of abstract domains (like queuing). However, for others, the concept of memory stack seems to be a complete abstraction.

C. Conveying complex concepts

The ability of metaphors to reify abstract idea no doubt explains how metaphors convey complexity with ease. New technologies are often coupled with new concepts and vocabulary, making metaphors an ideal vehicle to convey these complex new meanings. 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"* [34], In turn, Keen argued: *"Metaphors promote active communication, interpretation and understanding, and so encourage a rich discourse"* [8]. Examples of metaphors used to get across difficult ideas simply include:

- **"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"** and **"Home as communication environment"** in *Smart Home technology* (closely related to the IoT metaphors).

Carbonell, Sánchez-Esguevillas, and Carro point 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"* [35].

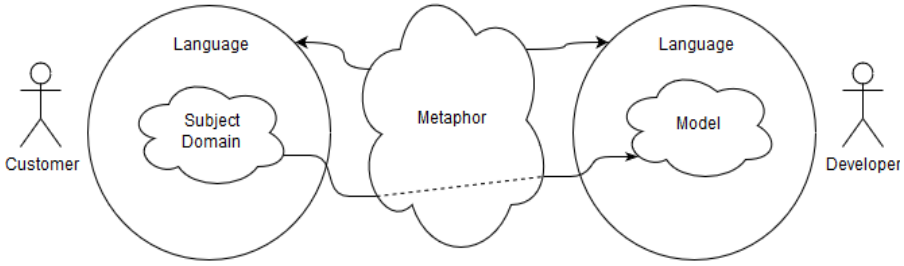

Fig. 1. Bridging language gaps by mapping the customer's domain to the development team's

D. Bridging communities

Specialists in professional contexts and educators in academic contexts face the same difficulty of conveying complex concepts to a lay audience who may lack either or both subject knowledge and technical terminology. In software engineering, metaphors help establish and facilitate communication during requirement elicitation and initial system design between the development team and project stakeholders. Metaphors provide both parties with a shared language that helps bridge the linguistic gulf between the participants (see Fig. 1).

Most present day human-centric projects require significant cross-disciplinary efforts necessitating communication between different groups of stakeholders. A key difficulty is for specialists to explain technical concepts to a lay audience. This is where metaphors come into play to provide models onto which concepts can be attributed and conveyed without the need for extended explanations. Using metaphors can convey complex concepts quickly by accessing known concepts and mapping them to new elements.

IV. Metaphors in computing and software engineering education

When learning new technical terms, using metaphors is a natural part of introducing new lexical material to learners (Fig. 2). 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 [36], many famous metaphors used in literature (such as **"Love as Journey"** [25]) or in technology areas (such as **"Development as War"** or **"System as Organism"** [37]) can be harnessed or adapted, giving the domain-specific examples of language in use. Metaphors lend themselves to creative adaption through the substitution of one or more elements as the example below shows.

"Education is the bridge between the present and the future".

"University is the bridge between school and success".

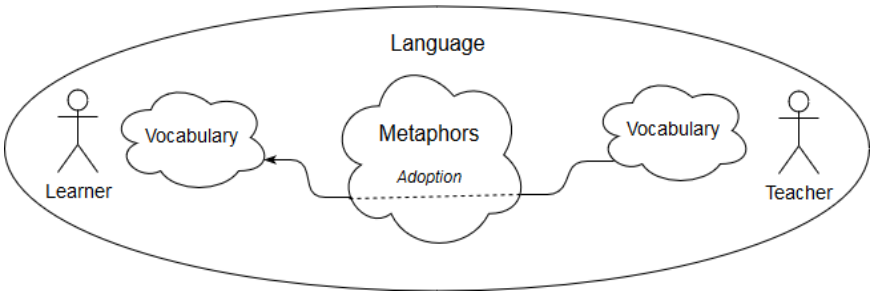

Fig. 2. Metaphor are helpful in enhancing learner's vocabulary

Even the teaching process itself can be described metaphorically with respect to the teacher's roles and responsibilities. Clarken introduced five (perhaps, not exhaustive) teaching metaphors: *teachers as parents, teachers as gardeners, teachers as prophets, teachers as pearl oysters, and teacher as physicians* [33]. Finding an appropriate teaching model is an important aspect of making the teaching process efficient and learner-friendly. There are three important outcomes of this work:

1) A teacher may operate differently by using different metaphors in different times: it could depend on the topic, on the type of class activity, or on particular students' skills (including soft skills) and their personal characteristics (such as responsiveness, discipline or ability to cooperate).

2) A teacher may feel uncomfortable if she has to operate according to metaphors, which do not correspond well to their personal views on educational process.

3) By understanding the teaching metaphors, teachers understand better the relationships between teachers and learners and the difference between teacher's and learner's perspectives.

A. Ubiquity of metaphors in software engineering and computing

Metaphors are ubiquitous in everyday discourse and are, at least, as common in software engineering. Their omnipresence was also a position supported by Aristotle according to arguments by Mahon [38]. Metaphors in software engineering and computing are diverse, multi-faceted and equally pervasive; and draw on different theories of metaphors. To describe the eclectic use of different theories both concurrently and consecutively of metaphors in technical disciplines, the term pluralistic methodology [39] was coined.

Research on technological language does not only concern technological or industrial applications, but also 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"* [40]. Software technology is one of the pervasive technologies changing everyday life and lifestyles. The language of a particular technology-sensitive domain (such as software engineering) is no longer only for domain professionals: all members of contemporary society need a better understanding of this language and its metaphors. What makes software metaphors particularly complex is their connotations to abstract entities. Johnson describes computer abstractions as *"based not on nature but rather on artificial world created by humans"* [41].

Within a scientific discourse, "model" itself is a systematic metaphor, conceived to be simpler and more abstract compared to the original concepts [27]. 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 [10]:

- **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.

B. Examples of metaphors in software engineering

Computer and software discourse relies on a large number of metaphorical expressions. Software works with many abstract concepts, which are largely metaphorical. According to Boyd, software can be considered as a special case of fiction, that is why it is essentially metaphorical [6]. As noted by Johnson, *"we should provide context by including technical details that help make the reasons for our metaphors clear"* [41]. Metaphors are actively exploited when we describe data entities and control structures, memory organization and program workflow, structural patterns and architecture designs. Examples of such metaphors are

listed in Table 2. In *Comments*, we use italics to designate the connections of metaphorically designed terms to their literal meaning sources according to Merriam-Webster dictionaries [42].

Interestingly, introducing a metaphor to a software domain may lead to further extension of the metaphor within the bag of concepts specific for software design. Fig. 3 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.

A **Factory Method** is for creation instances. It does not 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 Fig. 4).

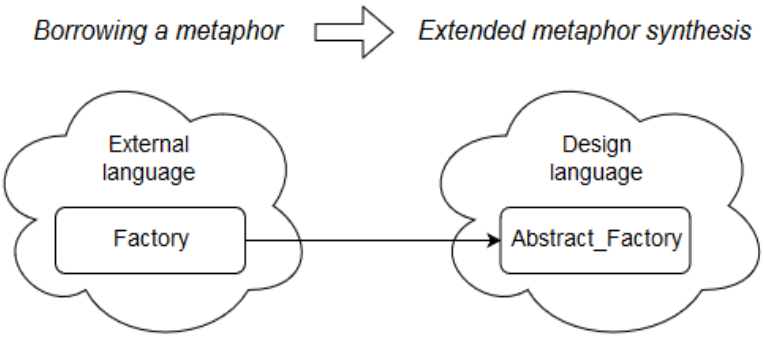

Fig. 3. Synthesis of extended metaphor

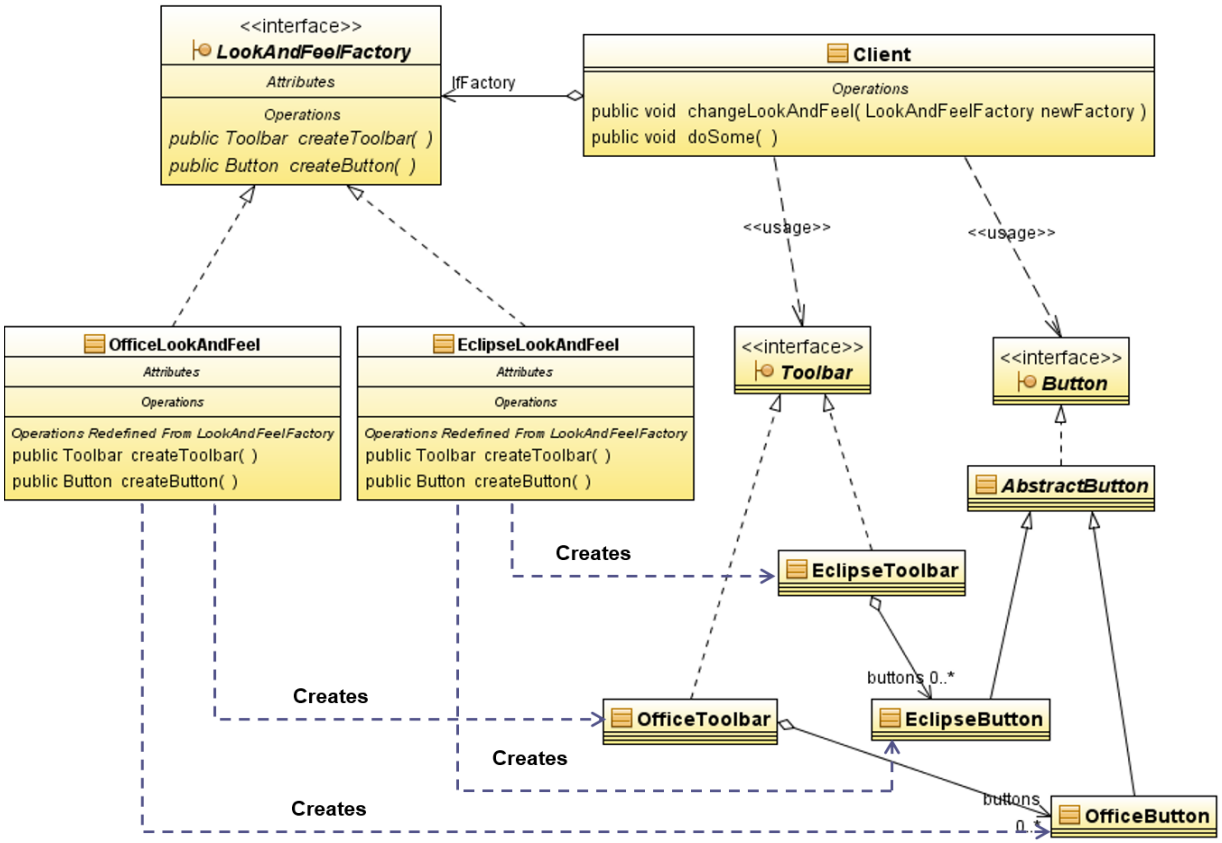

Fig. 4. Abstract factory and "Look and Feel" metaphor

Table 2. Examples of software metaphors

Domains	*Examples*	*Comments*
Program objects	Scope	Region of computer program, where particular name (referring to a variable, function, type, etc.) may be used to refer to an entity, thus defining a *space for activity* with this entity.)
	Assignment	A variable is assigned generally means that its value is set or re-set. In a sense, after assignment, the variable start working, thus, connecting to literal understanding of assignment as *a piece of work to be done*.
	Lifetime	Lifetime defines how long the object remains existing between the moments of object creation and its destruction. It may be close to execution time of the whole program or limited in accordance to the variable local scope or to explicit object creation/destruction processes. It can be naturally referred to the literal meaning *the duration of the existence* (on computer memory)
Control structures	Decision	Decision (also called selection) is one of three major control structures in structured programming. It refers to selecting a possible action depending on checked condition, thus an action *determination arrives at after consideration of condition* (usually in form of logic expression).
	Loop	Loop represents another important control structure in structured programming and refers to a sequence of *continually repeated* instructions performed until a certain condition is reached
Program workflow	Thread	In concurrent computing, thread is a sequence of instructions that can be executed independently and concurrently with other processes, which may share resources between each other. According to Saltzer [43], the term "thread", which is suggestive of the abstract concept embodied in the term "process", was originally introduced by Victor Vysotsky. The possible literal connotations of this term (such as *a group of filaments, a stream, a line of reasoning*) nicely refer to concurrency nature of its usage scope.
	Lazy computation	Lazy evaluation (also referred as call-by-need [44] is an evaluation strategy, where an expression (or its part) is not evaluated unless its value is needed, this one *avoids proceeding with an activity* until the activity is really needed
Modular structure	Library Package	Similarly to non-computer libraries, in software design, libraries, are collections of resources used by computer programs. Packages (also called namespaces) are containers that help to organize the code so that to put *related items*, or program entitles, (such as classes and interfaces) into a common name space, to facilitate their internal interaction and their export to external modules, and to avoid name collisions in complex projects
Interface design	Menu	Menu is standard component of user interface colors representing the commands used for accessing different program features. Computer menu is metaphor of the *assortment of offerings*, where the offerings are considered as available program functions that may be selected similarly to dish selection in a restaurant.
	Palette	Palette (in computer graphics applications) as set of available colors to be used in graphics design exploit the concept of painter's palette almost straightforwardly, even without much metaphoric contents.
	Folder and File	Similarly to a stationary folder (used *to hold or file papers*, computer folder is a storage (e. g., on computer hard drive) for placing and organizing different resources, or files
Design patterns	Factory method	This design pattern (in object-oriented design) defines an instance creation interface leaving subclasses to decide which object to construct. In turn, Factory class is used to *produce* (create) other classes (we illustrate this example in our explanations to Fig. 4 in more detail).
	Delegation	In object-oriented design, the literal meaning as *an act of empowering to act for another* is interpreted here as *delegation of the responsibility* to other objects and used in situations, when an object needs to be a different subclass of a class at different times.
	Future	Often used interchangeably with other names (Promise, Delay, Deferred), this pattern is about describing an organization, when an object is used to encapsulate the result of a computation in a way that hides from its clients whether the computation is synchronous or asynchronous. Thus, such "future" object acts as a proxy for a result that is initially unknown, usually because the computation of its value is not yet complete. The results of computation may *exist or at a later time*

Domains	*Examples*	*Comments*
Software analysis	Bad smell	In [45], Beck and Fowler defined a bad code smell as code properties (such as structural defects) possibly *affecting* deeper code problems. Code smells are indicators of the possibility for refactoring. Interestingly, some of names used to characterize common smells, are clear metaphors: shotgun surgery, lazy class, inappropriate intimacy, etc.
	Refactoring	According to [46], software code refactoring is changing the software system so that to improve its internal structure, without changing its external behavior. Though, etymology of this term (code re-production, or re-creation) seems to be quite obvious, the term itself could not usually be found in Merriam-Webster common language dictionaries, thus, currently its use is mostly limited by software design domain of computer science.
	Code mutant	In mutation testing, code mutant is an intentionally changed version of software code, so that its behavior is different from the original version. It can be used to evaluate the tests quality (in particular, by checking, whether the existing test suite could detect the difference between the original version and the mutant)

Indeed, *"Abstract factory"* could hardly be imagined in the 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.

V. Application of visual metaphors from fine arts to programming teaching

This section introduces a case study on using metaphors in credit-bearing programming classes taught to computer science majors by the first author within the context of academic courses of programming, software engineering and data management in the University of Aizu, Japan.

Metaphors in education are helpful for many reasons. First, to link students' knowledge to newly introduced concepts and models [47] (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). Metaphors work through domain mapping, not because of preexisting similarity. The combination of their ability to convey complex concepts and bridge the language gaps between experts and novice makes them a valuable teaching resource.

A. Use case 1: 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 [48]. 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 discussions 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 [49].

B. Use case 2: Black Square, or Persistence Metaphor

According to Demaine (MIT) [50], there are two senses in time travel (or temporal data structures). The first is **persistence** (*"Branching universe time travel model"*) – if we make a change in the past we create a new universe, we never destroy old universes. The second one is **retroactivity** (*"Back to the Future"*) – we

go back, make a change, return to the present and see what happened. The latter concept is normally more complicated compared to the former one [51].

In our data management course, we introduce a persistent data structure as a data structure that always preserves the previous versions of itself. Thus, the general idea is to keep all the versions of data structures. Data structure operations are all relative to a specific version of data structure. Specifically, in software development, persistent structures provide conceptual foundation for version control systems.

In turn, in the programming course, discussing persistent structures provides an opportunity to introduce the concept of immutability. Persistent structures are effectively immutable, as their operations do not (visibly) update the structure in-place, but instead always yield a new updated structure. Object immutability prevents possible improper coordination between the objects sharing the reference to another object like in concurrent threads. The example from stackoverflow.com illustrates the problem [52]:

```
Date d = new Date();
Scheduler.scheduleTask(task1, d);
d.setTime(d.getTime() + ONE_DAY);
Scheduler.scheduleTask(task2, d);
```

Assume we have a mutable `Date` object used to schedule task execution. Since two tasks share the same `Date` object, changing its value might affect not only the second task (as one could expect), but the first task as well. Thus, assuring correct object coordination requires careful programming. There are error risks. Of course, with two independent variables representing the dates (instead of one date reference), the problem would not exist (this is exactly what we mean by saying "careful programming" on the client side). However, a more reliable possible solution suggests organizing a library class (`Date`, in our case) in such a way that the state information of its instances never changes after they are constructed. No method, other than a constructor, should modify the values of a class instance variables. Change operations are disallowed. In operations that might change object new instance is created instead. Java class `java.util.Date` uses the same approach.

Going back to fine arts, we can use paint layers on canvas as a visual metaphor of partially persistent structures. In some art works, a new image sometimes covers the earlier painted layers. Nowadays, such hidden layers can be discovered with the use of X-ray examination based techniques and image processing technology. However, we can only virtually access the layers below the surface. For example, Kazimir Malevitch's post-suprematic "Black Square" (1915, Tretyakov Gallery, Moscow) serves as a prime example of partially persistent structure: we can virtually access the versions, but we are unable to update them (Fig. 5). We can theoretically update the upper layer, but fortunately we may not, unlike the Russian painter Ilya Repin (1834–1930), who attempted to correct his own finished works which had already been purchased by Russian collector Pavel Tretyakov (1832–1898) in Tretyakov's own house. This is why Tretyakov insisted that Repin never enter his house holding his brush and palette [53].

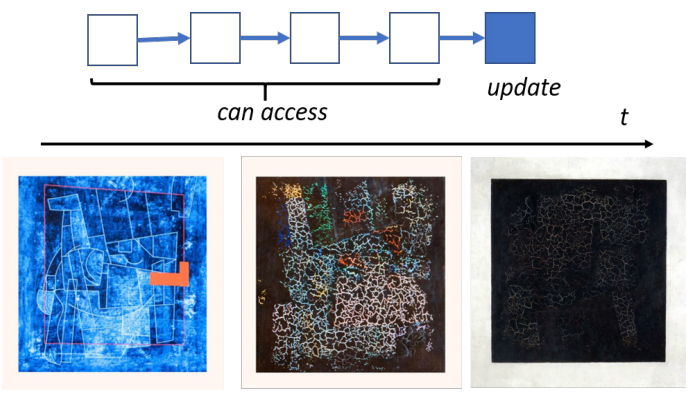

Fig. 5. Malevich's "Black Square" canvas layers as a metaphor of partial persistence

C. Use case 3: Form and Contents as a Readability Metaphor

In the programming class taught by the first author, there is an exercise entitled *"The Form inside the Work"*, which involves discussing visual metaphors as a vehicle to introduce multi-faceted software concepts. In particular, we discuss with the students how the concept of code readability may be expressed metaphorically and analyzed using masterpieces (see the example of using Van Eyck's "Annunciation" for such an exercise in [54]).

As pointed out by Oosterman et al. in [55], 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. In a similar vein to literary works, manner and matter are mutually dependent in visual works [56]: structures used by a creator in an artwork (the form) provide the way to reproduce the 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 multiple interpretations that can be considered as co-creation acts [57]. Paul Klee metaphorically described it as *an eye following the directions already existing inside the artwork* (as quoted in "La Vie mode d'emploi" by George Perec [58]). Even music (as an example of non-visual art) still works with some visual artifacts (music scores), and an experienced musician can often judge the value of a certain composition because of the beauty of its music score graphics [59], thus, mentally mapping the abstract graphics to the concrete sound.

Let us 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". Fig. 6 graphically demonstrates a possible interpretation.

As noted by Oliveira in [60], the canvas has a visible multilayer structure. This particularity was emphasized in the remarkable project of Taratynov and Dronov who implemented the sculpture version of "The Night Watch" [61, 62] (Fig. 7). "Departing" from the two central characters (representing a cooperation between Protestant and Catholic parties), the eye may follow different paths, but the most likely directions are implicitly embedded inside the picture: a trained soldier close to the captain in the second layer (Fig. 6, 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 characters of this painting (suggested by Oliveira to be a symbolic interpretation of the motherland).

Of course, this rendition is not the only possible way to rediscover rich symbolism of Rembrandt's masterpiece, which has many more symbolic allusions and enigmatic elements (their detailed analysis is naturally beyond the 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 [21]. As noted by Novokhatko, the creator *"speculates based on certain patterns of expectations in his audience"* [23].

In the case of software programs, it is commonly agreed that the code graphics, organization and legibility can be considered as essential aesthetic properties [63]. Meanwhile, the aesthetic values are connect-

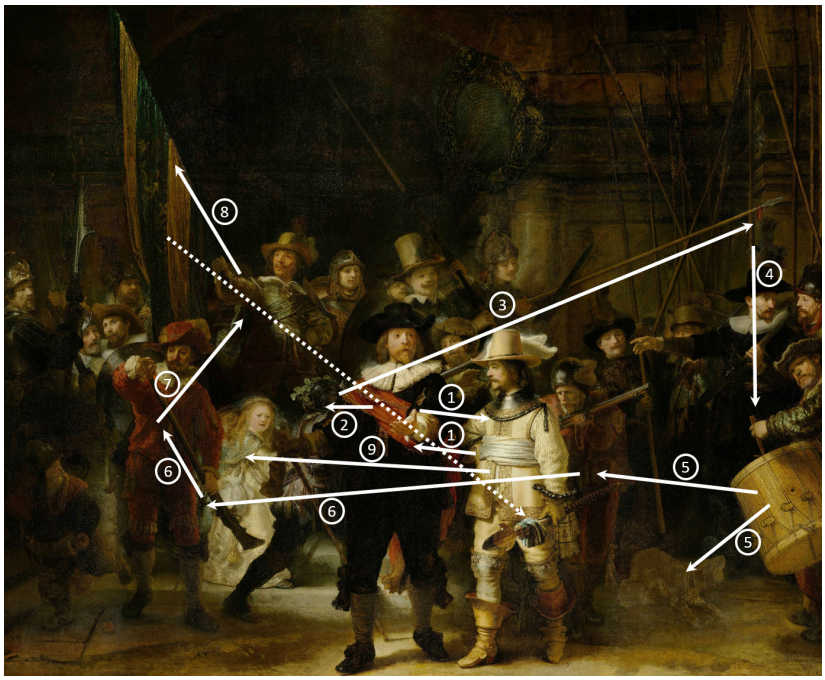

Fig. 6. Possible reading links inside "The Night Watch" by Rembrandt

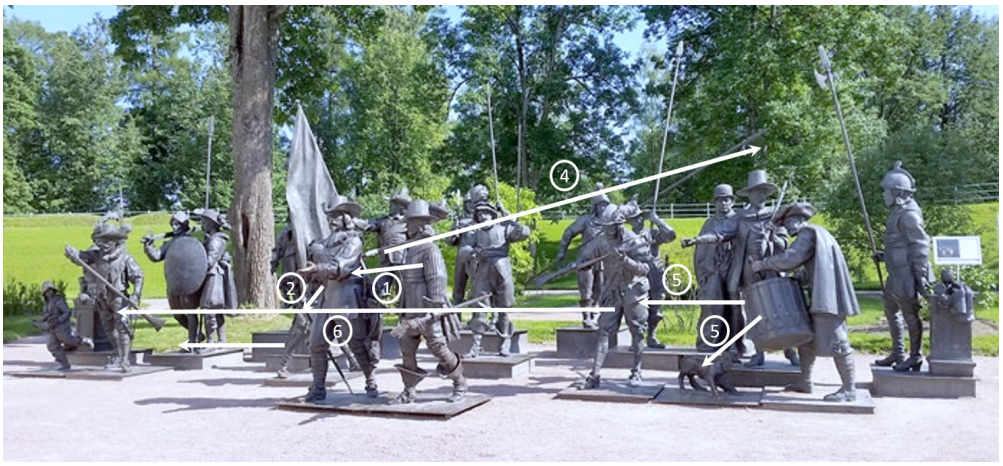

Fig. 7. Sculpture version of "The Night Watch" (Taratynov and Dronov's Paragon Project)

ed to the quality properties, as Edmonds candidly points out: *"if the resulting code is like spaghetti [...], it is not highly rated even if it performs its functions perfectly"* [64]. 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 gives interesting insights to improving developers' culture (where, by the way, "culture" can be understood both literally and metaphorically).

In the research presented in [65], based on the empirical code annotation study, the authors conclude that explicit source code comments only moderately affect the notion of code readability, which is in partial contradiction to common practices in programming teaching. Although the comments provide the very direct way of communication between the software writer and its reader, code readability may be increased more by improving the code organization and the models used (i. e., the code properties which do not provide a direct communication intent).

VI. Conclusion

Metaphors have long been harnessed in the humanities as vehicles to explain complex concepts. There appears to be a trend to using metaphors in the sciences, and given their ability to convey meaning and bridge terminology gaps, this should be encouraged. The results of our case study were promising. Incorporating metaphors from the fine arts into the curriculum and delivering the course content through metaphors was enjoyable and rewarding. Likewise, the students enjoyed their classes and not only learnt the technical aspects but also had the opportunity to analyze some masterpieces in detail. The feedback from students on the use of visual metaphors was very positive with a generous sprinkling of superlative adjectives like "best".

Drawing on original sources from linguistics and cognitive science, this study has examined the diversity and multi-facetedness of the concept of metaphor and its important role in technological and engineering disciplines. Since this research focuses on software education, we address a number of practical cases of using metaphors in programming classes by including a brief review of architectural software metaphors, metaphors of code organization, code readability and code aesthetics metaphors, as well as the metaphors of data persistence.

Learning metaphors is also connected to the development of soft skills, which are nowadays considered an important aspect of software engineering education [66]. However, the evaluation whether the use of metaphors significantly improve the learning process remains an open issue; empirical analysis of benefits and potential results is non-trivial.

A systematic analysis of metaphors used in software engineering is called for in order to classify the published approaches and to promote them to the members of academic community. Such an analysis may be considered as a component of learning and modeling language mechanisms, the latter being one of challenging problems of contemporary linguistics [67]. The possibilities of using and exploring metaphors in educational setting are rich and multifarious. A collaborative approach with experts from both liberal arts and technology domains could be adopted to extend this study of the pedagogic power of metaphors. There are many other important aspects that deserve attention, including software visualization and visual metaphors, metaphors of software architectures and design pattern metaphors, software code transformation and restructuring metaphors, cooperation and mutual dependencies of different theories of metaphor in their application to technology domains, and transition of metaphors introduced in technological domains back to the non-technological areas. We expect that this study may provoke further discussion on how using metaphors enhances the educational process and increases learning efficiency in various disciplines, from humanities to natural and information science.

Acknowledgement

The work is supported by research funding from the University of Aizu.

REFERENCES

[1] **E. Pyshkin,** Metaphor models in software education: An empirical study, in The 14th International Conference on Software Engineering Advances (ISCEA 2019). IARIA, 2019, pp. 30–35.

[2] **L. Mason,** Introduction: Bridging the cognitive and sociocultural approaches in research on conceptual change: Is it feasible? Educational psychologist, 42 (1) (2007) 1–7.

[3] **A. Mouraz, A.V. Pereira, R. Monteiro,** The use of metaphors in the processes of teaching and learning in higher education, International Online Journal of Educational Sciences, 5 (1) (2013) 99–110.

[4] **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.

[5] **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.

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

[7] **C. Chibaya,** A metaphor-based approach for introducing programming concepts, in 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC). IEEE, 2019, pp. 1–8.

[8] **C. Keen,** Treatment of metaphors in software engineering education, in Proceedings 1996 International Conference Software Engineering: Education and Practice. IEEE, 1996, pp. 329–335.

[9] **B.J. Oates, H. Gavin,** Metaphors in software engineering, in Engineering Psychology and Cognitive Ergonomics. Routledge, 2017, pp. 387–393.

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

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

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

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

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

[15] Songs. l'heure exquise (1890) by Paul Verlaine. English translation by Richard Stokes. 2020, retrieved: Jun, 2020. [Online]. Available: https://www.oxfordlieder.co.uk/song/2140.

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

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

[18] **L. Cameron,** Metaphor in educational discourse. London: Continuum, 2003.

[19] **G. Low, J. Littlemore, A. Koester,** Metaphor use in three UK university lectures, Applied linguistics, 29 (3) (2008) 428–455.

[20] **M. Armisen-Marchetti,** Histoire des notions rhétoriques de métaphore et de comparaison, des origines à Quintilien, Bulletin de l'association Guillaume Budé, 49 (4) (1990) 333–344.

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

[22] **M.S. Wood,** Aristotle and the question of metaphor, Ph.D. dissertation, University of Ottawa, 2015.

[23] **A. Novokhatko,** The linguistic treatment of metaphor in Quintilian, Pallas, 103 (2017) 311–318.

[24] **H.E. Butler, et al.,** The Institutio Oratoria of Quintilian. Harvard University Press, 4 (1922).

[25] **G. Lakoff, M. Johnson,** Metaphors we live by, Chicago, IL: University of, 1980.

[26] **I.A. Richards, J. Constable,** The philosophy of rhetoric. Oxford University Press New York, 94 (1965).

[27] **M. Black,** Models and metaphors: Studies in language and philosophy. Cornell University Press, 1962.

[28] **M. Black,** More about metaphor, Dialectica, 1977, pp. 431–457.

[29] **E. Safronenkova,** Hedging vs tolerance in presenting the scientific result in research articles (based on English research articles of the humanities field), St. Petersburg State Polytechnical University Journal. Humanities and Social Sciences, 10 (3) 2019 51–57. DOI: 10.18721/JHSS.10305

[30] **G. Lakoff,** The contemporary theory of metaphor, in Metaphor and Thought. UC Berkeley, 1993, retrieved: Sep, 2019. [Online]. Available: https://escholarship.org/uc/item/54g7j6zh

[31] **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, 9 (1) (2011) 26–64.

[32] **T.R. Colburn, G.M. Shute,** Metaphor in computer science, Journal of Applied Logic, 6 (4) (2008) 526–533.

[33] **R.H. Clarken,** Five metaphors for educators, 1997.

[34] **J.E. Kendall, K.E. Kendall,** Metaphors and their meaning for information systems development, European Journal of Information Systems, 3 (1) (1994) 37–47.

[35] **J. Carbonell, A. Sánchez-Esguevillas, B. Carro,** The role of metaphors in the development of technologies. The case of the artificial intelligence, Futures, 84 (2016) 145–153.

[36] **G. Lazar,** Exploring metaphors in the classroom, Teaching English, 2006.

[37] **J.E. Kendall, K.E. Kendall,** Metaphors and methodologies: Living beyond the systems machine, MIS quarterly, 1993, pp. 149–171.

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

[39] **P. Feyerabend,** Against method. Verso, 1993.

[40] **J. Krupczak, et al.,** Defining engineering and technological literacy, Philosophical and Educational Perspectives in Engineering and Technological Literacy, 3 (2012) 8.

[41] **G.J. Johnson,** Of metaphor and difficulty of computer discourse, Communications of the ACM, 37 (12) (1994) 97–103.

[42] Merriam-Webster, 2020, retrieved: Jun, 2020. [Online]. Available: https://www.merriam-webster.com/

[43] **J.H. Saltzer,** Traffic control in a multiplexed computer system. Ph.D. dissertation, Massachusetts Institute of Technology, 1966.

[44] **D.A. Watt,** Programming language design concepts. John Wiley & Sons, 2004.

[45] **K. Beck, M. Fowler, G. Beck,** Bad smells in code, Refactoring: Improving the design of existing code, 1 (1999) 75–88.

[46] **M. Fowler,** Refactoring: improving the design of existing code. Addison-Wesley Professional, 2018.

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

[48] **T. Dufva, M. Dufva,** Metaphors of code structuring and broadening the discussion on teaching children to code, Thinking Skills and Creativity, 22 (2016) 97–110.

[49] **M. Frank, P. Roehrig, 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.

[50] **E.D. Demaine,** 6.851 Advanced data structures. Spring 2012. Massachusetts Institute of Technology: MIT Open courseware, MIT, 2012, retrieved: June, 2019. [Online]. Available: https: //ocw.mit.edu/courses/electrical-engineering-and-computer-science/ 6-851-advanced-data-structures-spring-2012/

[51] **E.D.Demaine, J. Iacono, S. Langerman,** Retroactive data structures, ACM Trans. Algorithms, 3 (2), May 2007. [Online]. Available: http://doi.acm.org/10.1145/1240233.1240236.

[52] Mutable Date class fix, 2017, retrieved: Oct, 2019. [Online]. Available: https://stackoverflow.com/questions/43780276/mutable-date-class-fix?noredirect=1&lq=1

[53] **N.A. Mudrogel,** 58 let v Tretyakovskoy galeree. Vospominanya [58 years in the Tretyakov Gallery], 1962. (In Russian)

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

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

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

[57] **D. Likhachev,** Neskolko mysley o netochnosti iskusstva i stilisticheskikh napravleniyakh [Some ideas about uncertainty of arts and stylistic trends], Philologiya. Issledovaniya po yazyku i literature [Philology. Studies in language and literature], 1973, pp. 394–401. (In Russian)

[58] **G. Perec,** La Vie mode d'emploi. Hachette, Paris, 1978.

[59] **P. Florensky,** Analiz prostranstvennosti v khudozhestvennykh proizvedeniyakh [Spatiality Analysis in Works of Fine Arts], Stat'i i issledovaniya po istorii i filosofii iskusstva i arkheologii [Articles and research on the history and philosophy of art and archeology]. Mysl, Moscow, 2000, pp. 79–421, (In Russian)

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

[61] The biggest sculpture in the Netherlands: NIGHTWATCH 3D, 2004, retrieved: Sep, 2019. [Online]. Available: http://nightwatch3d.com/information.htm

[62] **D.J. Jansen, A. Taratynov, M. Dronov,** Paragone [: Sculptures After Paintings by Alexander Taratynov and Mikhail Dronov]. Ekega˚rd Galerie-Atelier, 2010. [Online]. Available: https://books.google.co.jp/ books?id=Il\ PjgEACAAJ.

[63] **S. Gruner,** Problems for a philosophy of software engineering, Minds and Machines, 21 (2) 2011 275–299.

[64] **E. Edmonds,** The art of programming or programs as art, Frontiers in Artificial Intelligence and Applications, 161 (2007) 119.

[65] **R.P. Buse, W.R. Weimer,** Learning a metric for code readability, IEEE Transactions on Software Engineering, 36 (4) (2009) 546–558.

[66] **J. Blake,** Real-world simulation: Software development, in Applied Degree Education and the Future of Work – Education 4.0, ser. Lecture Notes in Educational Technology, C. Hong and W.W.K. Ma, Eds. Cham, Switzerland: Springer, 2020, pp. 303–317.

[67] **L.N. Belyaeva, V.E. Chernyavskaya,** Scientific and technical texts in the framework of information 4.0: content analysis and text synthesis, St. Petersburg State Polytechnical University Journal. Humanities and Social Sciences, 10 (2) (2019) 53–63. (In Russian). DOI: 10.18721/JHSS.10205

Received 22.06.2020.

СПИСОК ЛИТЕРАТУРЫ

1. **Pyshkin E.** Metaphor models in software education: An empirical study // The 14th Internat. Conf. on Software Engineering Advances (ISCEA 2019). IARIA, 2019. Pp. 30–35.

2. **Mason L.** Introduction: Bridging the cognitive and sociocultural approaches in research on conceptual change: Is it feasible? // Educational psychologist. 2007. Vol. 42. No. 1. Pp. 1–7.

3. **Mouraz A., Pereira A.V., Monteiro R.** The use of metaphors in the processes of teaching and learning in higher education // Internat. Online J. of Educational Sciences. 2013. Vol. 5(1). Pp. 99–110.

4. **Murphy G.C.** Human-centric software engineering // Proc. of the FSE/SDP Workshop on Future of Software Engineering Research, ser. FoSER '10. New York, NY, USA: ACM, 2010. Pp. 251–254.

5. **Pyshkin E.** Liberal arts in a digitally transformed world: Revisiting a case of software development education // Proc. of the 13th Central & Eastern European Software Engineering Conf. in Russia, ser. CEE-SECR '17. New York, NY, USA: ACM, 2017. Pp. 23:1–23:7.

6. **Boyd N.** Software metaphors. 2003, retrieved: Sep. 2019 [Online] // URL: https://pdfs.semanticscholar.org/deee/ 512ab8b7a3753fda248fe99780e3470e6881.pdf.

7. **Chibaya C.** A metaphor-based approach for introducing programming concepts // 2019 Internat. Multidisciplinary Information Technology and Engineering Conf. (IMITEC). IEEE, 2019. Pp. 1–8.

8. **Keen C.** Treatment of metaphors in software engineering education // Proc. of Internat. Conf. Software Engineering: Education and Practice. IEEE, 1996. Pp. 329–335.

9. **Oates B.J., Gavin H.** Metaphors in software engineering // Engineering Psychology and Cognitive Ergonomics. Routledge. 2017. Pp. 387–393.

10. **Smolander K.** Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice // Proc. of Internat. Symp. on Empirical Software Engineering. IEEE, 2002. Pp. 211–221.

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

12. **Shakespeare W.** As You Like It. Edward Blount and William and Isaac Jaggard, London, 1623.

13. **Shelley P.B.** The Cloud. Charles and James Ollier, London, 1820.

14. **Verlaine P.** L'heure exquise. Creuzevault, 1936.

15. Songs. l'heure exquise (1890) by Paul Verlaine. English translation by Richard Stokes. Retrieved: Jun, 2020 [Online] // URL: https://www.oxfordlieder.co.uk/song/2140.

16. **Cameron L.** Operationalising 'metaphor' for applied linguistic research // Researching and applying metaphor. 1999. Pp. 3–28.

17. **Weaver R.M., Beal R.S.** A rhetoric and handbook. Holt, Rinehart and Winston, 1967.

18. **Cameron L.** Metaphor in educational discourse. London: Continuum, 2003.

19. **Low G., Littlemore J., Koester A.** Metaphor use in three UK university lectures // Applied linguistics. 2008. Vol. 29. No. 3. Pp. 428–455.

20. **Armisen-Marchetti M.** Histoire des notions rhétoriques de métaphore et de comparaison, des origines à Quintilien // Bulletin de l'association Guillaume Budé.1990. Vol. 49. No. 4. Pp. 333–344.

21. **Butcher S.H.** The poetics of Aristotle edited with Critical Notes and a Translation. Macmillan, 1902.

22. **Wood M.S.** Aristotle and the question of metaphor: Ph.D. dissertation. University of Ottawa, 2015.

23. **Novokhatko A.** The linguistic treatment of metaphor in Quintilian // Pallas. 2017. Vol. 103. Pp. 311–318.

24. **Butler H.E., et al.** The Institutio Oratoria of Quintilian. Harvard University Press, 1922, Vol. 4.

25. **Lakoff G., Johnson M.** Metaphors we live by. Chicago, IL: University of, 1980.

26. **Richards I.A., Constable J.** The philosophy of rhetoric. Oxford University Press New York, 1965. Vol. 94.

27. **Black M.** Models and metaphors: Studies in language and philosophy. Cornell University Press, 1962.

28. **Black M.** More about metaphor // Dialectica, 1977. Pp. 431–457.

29. **Safronenkova E.** Hedging vs tolerance in presenting the scientific result in research articles (based on English research articles of the humanities field) // St. Petersburg State Polytechnical University Journal. Humanities and Social Sciences. 2019. Vol. 10. No. 3. Pp. 51–57.

30. **Lakoff G.** The contemporary theory of metaphor // Metaphor and Thought. UC Berkeley, 1993, retrieved: Sep. 2019 [Online] // URL: https://escholarship.org/uc/item/54g7j6zh.

31. **Steen G.J.** The contemporary theory of metaphor – now new and improved! // Review of Cognitive Linguistics. Published under the auspices of the Spanish Cognitive Linguistics Association. 2011. Vol. 9. No. 1, Pp. 26–64.

32. **Colburn T.R., Shute G.M.** Metaphor in computer science // J. of Applied Logic. 2008. Vol. 6. No. 4. Pp. 526–533.

33. **Clarken R.H.** Five metaphors for educators, 1997.

34. **Kendall J.E., Kendall K.E.** Metaphors and their meaning for information systems development // European J. of Information Systems. 1994. Vol. 3. No. 1. Pp. 37–47.

35. **Carbonell J., Sánchez-Esguevillas A., Carro B.** The role of metaphors in the development of technologies. The case of the artificial intelligence // Futures. 2016. Vol. 84. Pp. 145–153.

36. **Lazar G.** Exploring metaphors in the classroom. Teaching English, 2006.

37. **Kendall J.E., Kendall K.E.** Metaphors and methodologies: Living beyond the systems machine // MIS quarterly. 1993. Pp. 149–171.

38. **Mahon J.E.** Getting your sources right // Researching and applying metaphor. 1999. Pp. 69–80.

39. **Feyerabend P.** Against method. Verso, 1993.

40. **Krupczak J., et al.** Defining engineering and technological literacy // Philosophical and Educational Perspectives in Engineering and Technological Literacy. 2012. No. 3. P. 8.

41. **Johnson G.J.** Of metaphor and difficulty of computer discourse // Communications of the ACM. 1994. Vol. 37. No. 12. Pp. 97–103.

42. Merriam-Webster. 2020, retrieved: Jun, 2020 [Online] // URL: https://www.merriam-webster.com/

43. **Saltzer J.H.** Traffic control in a multiplexed computer system: Ph.D. dissertation. Massachusetts Institute of Technology, 1966.

44. **Watt D.A.** Programming language design concepts. John Wiley & Sons, 2004.

45. **Beck K., Fowler M., Beck G.** Bad smells in code // Refactoring: Improving the Design of Existing Code. 1999. Vol. 1. Pp. 75–88.

46. **Fowler M.** Refactoring: improving the design of existing code. Addison-Wesley Professional, 2018.

47. **Umar I.N., Hui T.H.** Learning style, metaphor and pair programming: Do they influence performance? // Procedia-Social and Behavioral Sciences. 2012. Vol. 46. Pp. 5603–5609.

48. **Dufva T., Dufva M.** Metaphors of code structuring and broadening the discussion on teaching children to code // Thinking Skills and Creativity. 2016. Vol. 22. Pp. 97–110.

49. **Frank M., Roehrig P., Pring B.** 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.

50. **Demaine E.D.** 6.851 Advanced data structures. Spring 2012. Massachusetts Institute of Technology: MIT Open courseware. MIT, 2012, retrieved: June, 2019 [Online] // URL: https: //ocw.mit.edu/courses/electrical-engineering-and-computer-science/ 6-851-advanced-data-structures-spring-2012/

51. **Demaine E.D., Iacono J., Langerman S.** Retroactive data structures. ACM Trans. Algorithms. May 2007. Vol. 3. No. 2, [Online] // URL: http://doi.acm.org/10.1145/1240233.1240236.

52. Mutable Date class fix. 2017, retrieved: Oct, 2019 [Online] // URL: https://stackoverflow.com/questions/43780276/mutable-date-class-fix?noredirect=1&lq=1.

53. **Мудрогель Н.А.** 58 лет в Третьяковской галерее. Воспоминания. Л.: Художник РСФСР, 1962. 206 с.

54. **Pyshkin E.** Designing human-centric applications: Transdisciplinary connections with examples // 2017 3rd IEEE International Conference on Cybernetics. IEEE, 2017. Pp. 1–6.

55. **Oosterman J., Yang J., Bozzon A., Aroyo L., Houben G.-J.** On the impact of knowledge extraction and aggregation on crowdsourced annotation of visual artworks // Computer Networks. 2015. Vol. 90. Pp. 133–149.

56. **McElroy J.G.** Matter and manner in literary composition // Modern Language Notes. 1888. Pp. 29–33.

57. **Лихачев Д.** Несколько мыслей о неточности искусства и стилистических направлениях // Филология. Исследования по языку и литературе. 1973. С. 394–401.

58. **Perec G.** La Vie mode d'emploi. Hachette, Paris, 1978.

59. **Флоренский П.** Анализ пространственности в художественных произведениях // Статьи и исследования по истории и философии искусства и археологии. М.: Мысль, 2000. С. 79–421.

60. **Oliveira P.M.** The Dutch company. Retrieved: Aug. 2019 [Online] // URL: https://www.academia.edu/8579003/_Eng_The_Dutch_Company.

61. The biggest sculpture in the Netherlands: NIGHTWATCH 3D. 2004. Retrieved: Sep. 2019 [Online] // URL: http://nightwatch3d.com/information.htm

62. **Jansen D.J., Taratynov A., Dronov M.** Paragone [Sculptures After Paintings by Alexander Taratynov and Mikhail Dronov]. Ekega˚rd Galerie-Atelier, 2010 [Online] // URL: https://books.google.co.jp/books?id=II\ PjgEACAAJ.

63. **Gruner S.** Problems for a philosophy of software engineering // Minds and Machines. 2011. Vol. 21. No. 2. Pp. 275–299.

64. **Edmonds E.** The art of programming or programs as art // Frontiers in Artificial Intelligence and Applications. 2007. Vol. 161. P. 119.

65. **Buse R.P., Weimer W.R.** Learning a metric for code readability // IEEE Transactions on Software Engineering. 2009. Vol. 36. No. 4. Pp. 546–558.

66. **Blake J.** Real-world simulation: Software development // Applied Degree Education and the Future of Work – Education 4.0, ser. Lecture Notes in Educational Technology. Cham, Switzerland: Springer, 2020. Pp. 303–317.

67. **Беляева Л.Н., Чернявская В.Е.** Научный и технический текст и Информация 4.0: ключевые задачи при создании структурированного контента // Научно-технические ведомости СПбГПУ. Гуманитарные и общественные науки. 2019. Т. 10. № 2. С. 53–63. DOI: 10.18721/JHSS.10205

Статья поступила в редакцию 22.06.2020.

THE AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Pyshkin Evgeny
Пышкин Евгений Валерьевич
E-mail: pyshe@u-aizu.ac.jp

Blake John
Блейк Джон
E-mail: jblake@u-aizu.ac.jp