ISBN: 978-989-8704-43-6 © 2022

DEVELOPING AN ONTOLOGY OF MULTIPLE PROGRAMMING LANGUAGES FROM THE PERSPECTIVE OF COMPUTATIONAL THINKING EDUCATION

Lalita Na Nongkhai[1], Jingyun Wang[2] and Takahiko Mendori[1]
[1]*Graduate School of Engineering, Kochi University of Technology, Japan*
[2]*Department of Computer Science, Durham University, UK*

ABSTRACT

This paper proposes the design of an ontology of multiple programming languages and give three examples to show the methodology. Our ontology aims to summarize the core of computational thinking logic by elaborating the concepts of three object-oriented programming languages in the industry: Python, Java, and C#. Therefore, the construction of the ontology lies not only on these three programming languages but also on their common concepts. This kind of ontology design facilitates the ontology extension and merging when concepts of other programming languages are added in the future. This ontology could be used to not only provide a guideline for any research work focusing on computational thinking education but also describe the common concept of visual programming tools and existing programming languages. Based on this ontology, an adaptive tutoring system intended to provide learners with personalized programming exercises, is under development.

KEYWORDS

Computational Thinking, Ontology, Programming Education

1. INTRODUCTION

Computational Thinking (CT) is a fundamental skill of a problem-solving approach in various skills such as English, Science, Mathematics, etc. (Jeannette, 2006). There are many approaches to teach computational thinking model. Learning with visual programming (VP) tools is suggested to be one of suitable method for children between 8 and 12 years old (or primary and secondary students) (Mercier et al, 2021). The existing VPs normally use gamification and animation elements to make the learning environment more engaging (Rose, Simon P. et al., 2017). Furthermore, in order to support CT skills learning, the basic programming concepts (such as variables, repetition, and conditional statements), to some extension, are abstracted by VP languages with the more advanced concepts (such as recursion, traversal, or indexing). However, few evidence supports that programming language can help those students learn CT skill easier. Also, those VP tools do not cover the same concepts. For example, Springin (2016) enables learners to make their drawing as a character and define the character ability, and relations, etc., by tapping the icons. VISCUIT (Watanabe et al., 2018), defines a glasses-shape item; after adding drawings in both glasses, users can create sequences and conditions to activate image in left turning to the one in right. Both of those two tools focus on a sequential logic in order to support learners easily understand conditional and repetition. Another popular type is the block-based VP tool, such as Scratch (Maloney et al., 2010), Blocky (Fraser et al., 2012) and MOONBlock (Mukai and mLAB, 2016), which enable learners to make programs by blocking-code. Some of them (such as Blocky) have a feature to convert from blocking-code to programming languages (Javascript, Python, etc). This feature not only supports learners to compare VP with PL, but also makes those tools involve more programming concepts (variables, conditions, repetitions, function) than other types of VP tools.

In higher education, explaining problem solving examples while teaching a specific programming language (PL) is another methodology for computational thinking logic education. Obviously, a PL covers more content than any existing VP and its education is more complicated. The trend of PL has changed quickly, as new VP tools are continually developing. This makes it more difficult for teachers to find a suitable teaching strategy. Because VP and PL have many programming concepts and not all concepts are included in every VP tool or PL. To address this issue, the summary of the common concept of any programming course, whatever it is VP, or any existing PL, can help any programming teacher find a strategy to design material or exercise. Also, it can help researchers/developers to design new programs or expand existing e-learning platforms or tools to support CT education.

Ontology can be used to represent any domain knowledge concept and relationship in any interested fields. In the education field, the e-learning systems adopting ontology as a semantic network to enhance their functions have been proven with the ability to support learners and instructors to search the information quickly and accurately (Wang, J. et al, 2014; Wang, J. et al, 2020). The object properties of ontology can be used to describe the relationship between knowledge concepts and its data properties can be used to elaborate information and examples of knowledge concepts. Furthermore, these ontologies can be easily reused in other research or systems.

In this work, we propose a COmputatioNal ThinkIng oNtology mUltiple prOgramming langUageS (CONTINUOUS, can be accessed by https://github.com/lalita-nk/CONTINUOUS.git) to describe common concepts of any programming course (no matter VP or PL). Our research questions are: can CONTINUOUS benefit (1) novice learners in understanding and determining their learning orders, (2) programming lectures in making teaching plan and preparing materials/exercises, and (3) the researcher working on the designs new VP/PL or the development of the existing VP/PL? For the education of programming languages, most of the previous systems incorporate only one ontology of an existing programming language. The ontology designed in our research involves the domain knowledge of three programming languages: Python, Java, and C# (C sharp), which are the most popular programming languages for beginner learners. CONTINUOUS is intended be the foundation of any intelligent tutoring systems which need to record the metadata of programming knowledge.

2. LITERATURE REVIEW

Most previous ontologies implemented in programming language fields focus on the procedural programming language C or object-oriented programming language Java. Sosnovsky and Gavrilova (2006) proposed a step-designing approach to create an educational ontology for C programming to explain knowledge concepts and their relationship in C. The top-level class of this ontology is “C programming”, following with the second-level classes “SYNTAX”, “PROGRAMMING TECHNIQUES”, and “PLATFORMS”. They claimed that this ontology shows the important concept of C programming from the researchers’ vision. However, their ontology only includes “has part” relation to show the hierarchical structure. Ming-Che Lee et al. (2005) proposed the ontology of a Java programming language, one of the object-oriented programming languages. The objective of this research is to guide instructors to design learning strategies for teaching. The relationship between concepts can help instructors to build more suitable material. This research also proposed a methodology to build this ontology so that any researchers interested in the Java programming language can refer to this ontology in their research.

Pierrakeas et al. (2012) proposed to organize Learning Object (LOs) based on two ontologies of two programming languages: C and Java. Their ontologies included the basic concept of two languages as metadata of the corresponding Learning Objects (LO) to help instructors make clearly the domain concepts. The instructors can follow the information in the ontology for the creation of learning material and teaching strategies. However, their ontology only describes the content of C and Java separately, no common content is summarized.

Our research proposed and implemented an ontology (called "CONTINUOUS") which combines the concepts of three programming languages. We designed a top-level class “common concepts” which summarizes the basic concepts shared by all the three programming languages and designed the concepts in each language in a separate class. "CONTINUOUS" may benefit instructors, learners, and others to apply this basic programming language concept in their work or research.

ISBN: 978-989-8704-43-6 © 2022

3. THE DESIGN OF A COMPUTATIONAL THINKING ONTOLOGY BASED ON MULTIPLE PROGRAMMING LANGUAGES (CONTINUOUS)

3.1 The Design of Top-level & Second-level Classes

The construction of the programming language ontology in this research covering the concepts of Python, Java, and C#, is consistent with the knowledge emerging in our Question Bank (include 20 basic concepts) (Na Nongkhai et al., 2021), which are design based on opinion of instructors who have several years of programming teaching experience, and existing programming tutorials. CONTINUOUS was built by Protégé (Musen and Protégé Team, 2015), a popular and open-source ontology editor.

According to CONTINUOUS design which includes more than one programming language. The top-level concepts consist of 4 classes: common concepts, C sharp, Java, and Python, as shown in Figure 1. For the top-level, our design first focuses on the "common concepts" which covers all basic concepts of the three programming languages such as built-in function, data structure, conditionals (if, if-else, etc.), repetition (for-loop, etc.), method, etc. CONTINUOUS defines the basic programming concepts as a "class" (Top-level and Second-level as shown in Figure 1) and the syntax of a specific programming language is located on the leaf as an "instance".

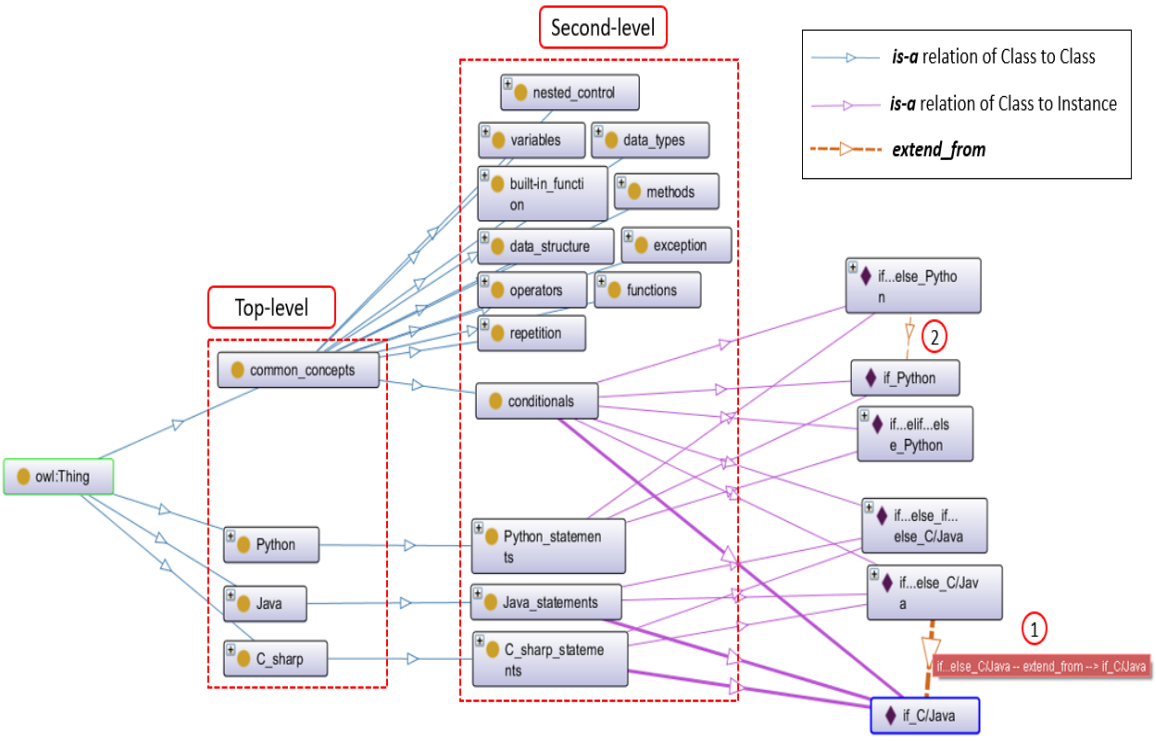

Figure 1. An example of instance in class "conditionals"

For example, "if_C/Java" is designed as one instance of "conditionals" because the "if statement" has the same programming syntax in both C# and Java while "if_Python" as another instance due to its different syntax. On the other hand, "if_C/Java" is also an instance of the second-level concept "Java_statements" and "C_sharp_statements" while "if_Python" is also an instance of "Python_statements". We also designed a relation named "extend from" between "if…else_C/Java" and "if_C/Java" (① in Figure 1) because "if-else statement" is the syntax that extends from the "if statement", this relation (② in Figure 1) also exists between

"if_Python" and "if…else_Python" for the same reason. In summary, the design of the instances and their relations is based on the similarity and the difference of syntax of the three programming languages. Therefore, as shown in Figure 2, the object property of "if…else_C/Java" in CONTINUOUS includes "extend from" connecting to "if_C/Java", which corresponds to ① in Figure 1. And the data property is designed to include "definition" (③ in Figure 2), "example" (② in Figure 2, which shows a code snippet), and "grammar specification" (① in Figure.2, which describes the syntax) to explain the usage of "if-else statement".

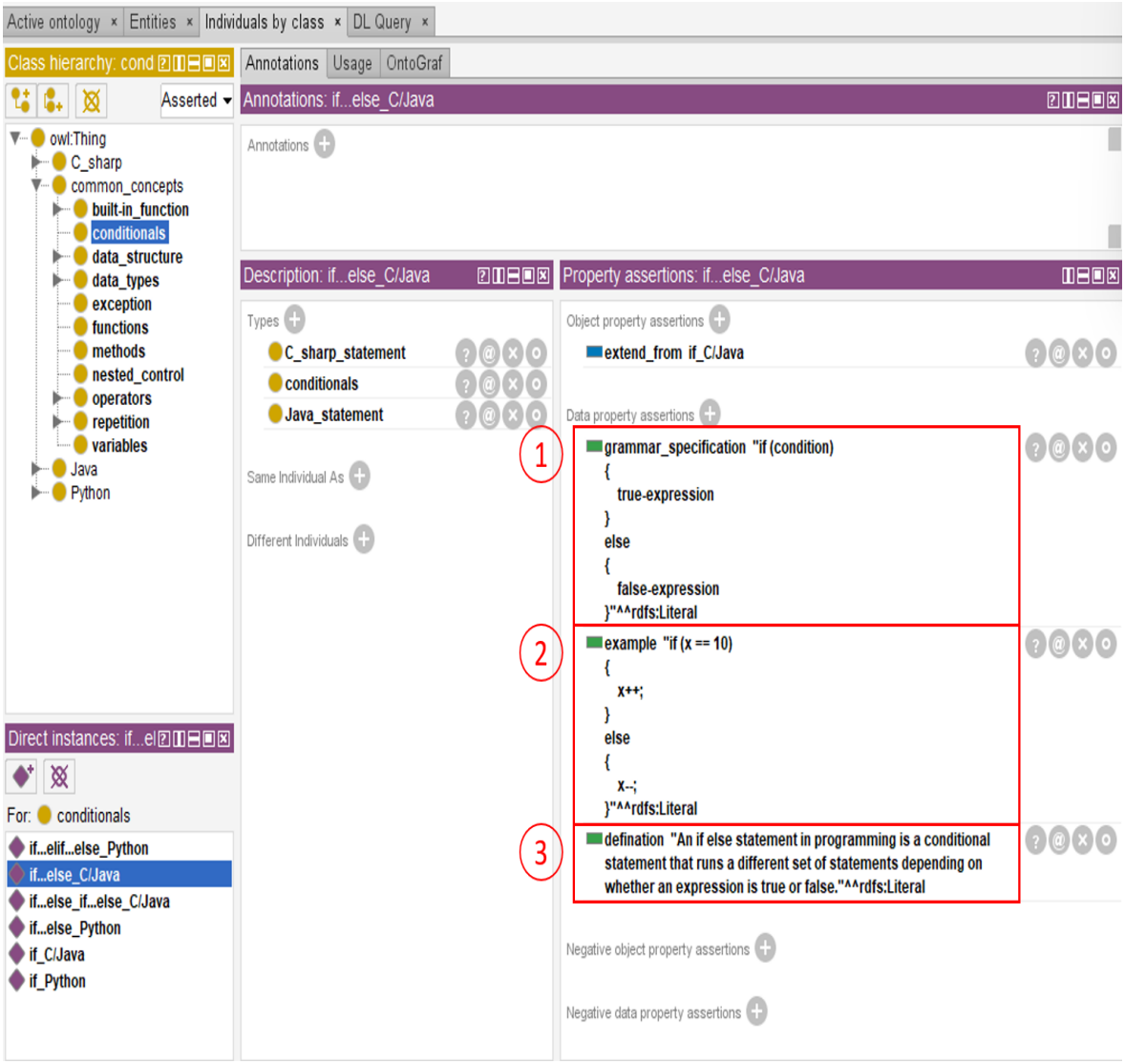

Figure 2. An example of all properties in "if…else_C/Java"

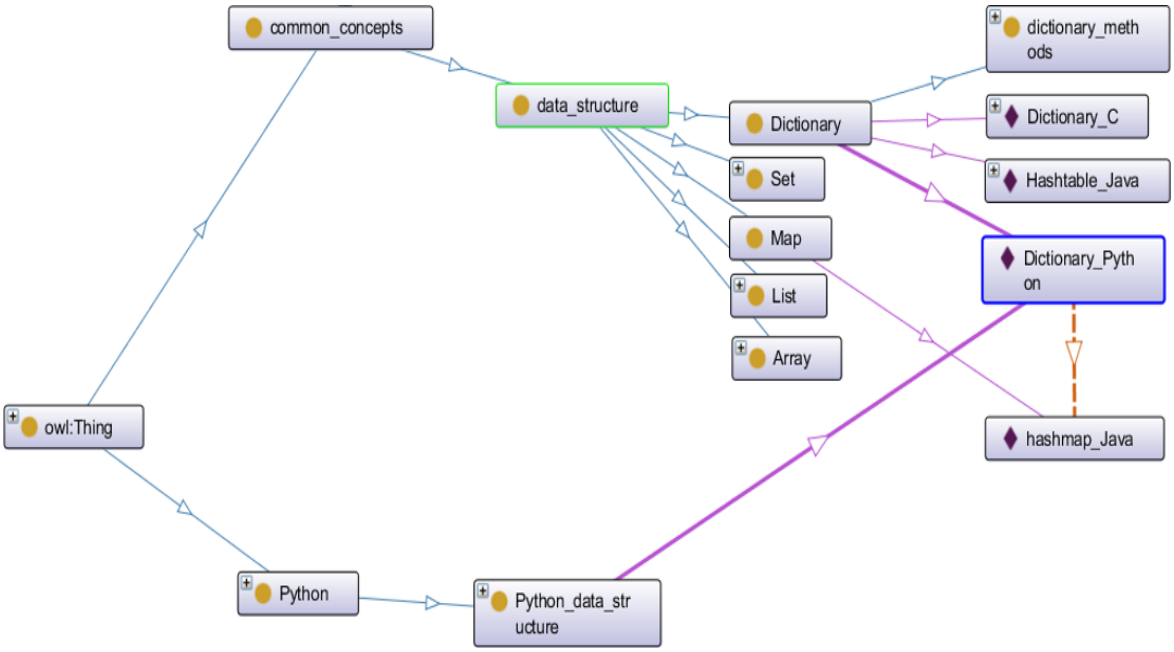

Figure 3. An example of "common concepts" and "Python"

3.2 The Design of "Data_Structure" in "Common_Concepts"

Another design focus of "common_concepts" is on "data_structure", which are the most essential in any PL (Wegner, 1971) but rarely appears in VP. As shown in Figure 3, there are 5 common Data structures: Dictionary, Set, Map, List and Array. Some of the structures (such as "List") are shared by the three languages, but some (such as "Dictionary") only appear in two of the languages: Python and C#. "Dictionary" is a data structure that shared by C# and Python, but its grammar in Python is different from the grammar in C#. Therefore, to describe the specific usage in Python, the "Dictionary_Python" is designed as an instance of the second-level concept "Python_data_structures" which is the subclass of the top-level concept "Python".

3.3 Abstract Concepts in "Common_Concepts"

The instances in CONTINUOUS could be either programming statements or abstract concepts. For example, "methods" as one of the "common_concepts", is designed to include abstract concepts of the functionality of each specific method (such as "add_an_element"), and "list_methods" consists of the methods in List (an abstract data type) We design "has_an_instance" (which has a reverse relation called "is_an_instance_of") to link the abstract concept (such as "sort_the_elements") to its corresponding method in each specific programming language (such as "sort()_Python", an instance of "Python_built-in_function"). These abstract concepts provide the summary of the common concepts needed in computational thinking logic while each programming language has their own syntax. This kind of design not only provide the perspective from one specific programming language but also from general similarity of all programming languages.

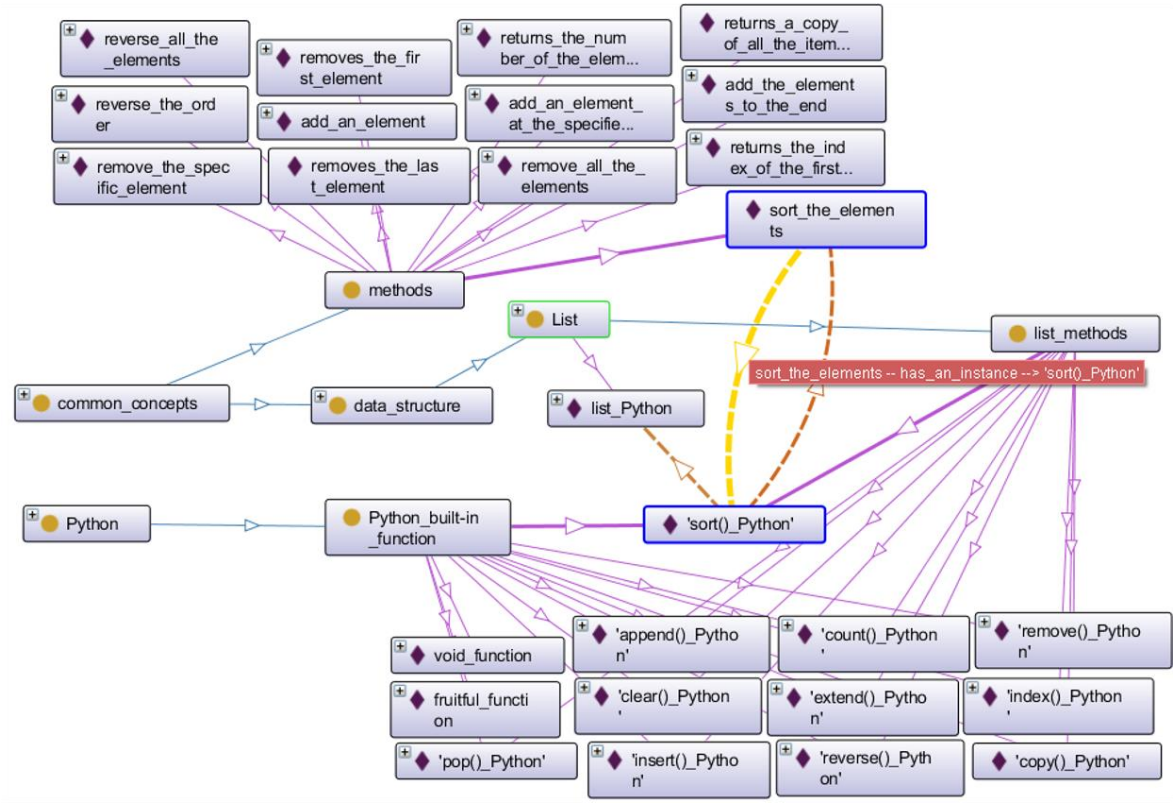

Figure 4. An example of direct instinct and object property of "sort_the_elements"

4. CONCLUSION AND FUTURE WORK

This paper describes some examples of CONTINUOUS to show how we identify the core concept of CT education by merging the concepts of existing programming languages. We intend to support CT education no matter what kind of programming language is used as a tool for the implementation. The common concepts designed in CONTINUOUS may benefits novice learners in determining learning orders, instructors in making strategies of pedagogies, and researchers in expanding CONTINUOUS or applying CONTINUOUS as metadata in any intelligent tutoring systems.

In the future, CONTINUOUS can be easily extended to include more programming language by simply adding more top-level concepts named with the names of programming languages and modifying the content inside the "common concepts". In other words, the feasibility and reusability of ontology facilitate the expansion of its contents. Based on CONTINUOUS, we are implementing a system (Na Nongkhai et al., 2021) to help instructors to design exercises that can be used to provide personalized support for learners. This system will allow instructors to add exercises based on the knowledge defined in CONTINUOUS and enable learners to check their personal progress with visualization support.

REFERENCES

Fraser, N. et al, 2012. *Blockly. Google Developers*. Retrieved from https://developers.google.com/blockly

Jeannette M. Wing. 2006. Computational thinking. Communication of the ACM. ACM 49, 3 (March 2006), 33–35.

Maloney, J. et al, 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, Vol. 10 No. 4, pp. 1-15.

Mercier, C. et al, 2021. Formalizing Problem Solving in Computational Thinking: An Ontology approach. *2021 IEEE International Conference on Development and Learning (ICDL)*. Beijing, China, pp. 1-8.

Ming-Che Lee et al, 2005. Java learning object ontology. *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*. Kaohsiung, Taiwan, pp. 538-542.

Mukai, N. and mLAB, 2016. Game programming with MOONBlock.Retrieved from https://mukai-lab.info/pages/tech/enchant_js/moonblock/

Musen, M. A., and Protégé Team, 2015. The Protégé Project: A Look Back and a Look Forward. *AI matters*, Vol. 1, No. 4, pp. 4–12.

Na Nongkhai, L. et al, 2021. A Framework of Exercise Recommendation for Novice Learners in Computer Programming. *ICCE 2021: The 29th International Conference on Computers in Education Online*. pp. 746-749.

Pierrakeas, C. et al, 2012. An Ontology-Based Approach in Learning Programming Languages. *16th Panhellenic Conference on Informatics*. Piraeus, Greece, pp. 393-398.

Rose, Simon P. et al, 2017. An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of e-Learning*, Vol. 15, pp. 297-309.

Sosnovsky, S.A., and Gavrilova, T., 2006. Development Of Educational Ontology for C-Programming. *International Journal "Information Theories & Applications"*, Vol. 13, No. 4, pp. 303-307.

Springin, 2016. *Springin' – Creative Programming Tool*. Retrieved from https://www.springin.org/en/

Wang, J. et al, 2014. A language learning support system using course-centered ontology and its evaluation. *Computers & Education*, Vol. 78, pp. 278-293.

Wang, J. et al, 2020. Development and evaluation of a visualization system to support meaningful e-book learning. *Interactive Learning Environments*, pp. 1-18.

Watanabe, T. et al, 2018. Programming Lessons for Kindergarten Children in Japan. *Constructionism2018*. Vilnius, Lithuania, pp. 741-744.

Wegner, P., 1971. Data structure models for programming languages. *ACM SIGPLAN Notices*, Vol. 6 No. 2, pp. 1-54.