MODIFIED CDIO FRAMEWORK FOR ELEMENTARY TEACHER TRAINING IN COMPUTATIONAL THINKING

Stephanie Hladik, Dr. Laleh Behjat, Dr. Anders Nygren

Electrical and Computer Engineering, University of Calgary

ABSTRACT

Computational thinking involves designing systems, finding solutions to problems, and understanding human behaviour through computer science concepts. Recent reports call for the inclusion of computational thinking in elementary (K-6) education, but there are barriers including a lack of teacher knowledge and confidence in the subject, and in Canada, a lack of a nationally-defined curriculum. Computational thinking is often taught outside of the formal educational system, and in some cases, alongside engineering design concepts. This provides an opportunity to use the CDIO framework to teach computational thinking. While the CDIO framework is designed for post-secondary engineering programs, it has been successfully used at the K-12 level, though it must be adapted and simplified for use at the elementary school level. This paper breaks down each of the Conceiving, Designing, Implementing, and Operating steps in the CDIO Syllabus to analyze and compare them against computational thinking and programming frameworks. This information is then used to adapt the C-D-I-O steps to teach computational thinking concepts to university level students in education and in-service teachers. The proposed technique provides a framework for teachers to create their own computational thinking activities at the elementary level, and for students to move through the steps as they complete such activities. This paper details the creation and use of the new framework, beginning with the existing CDIO framework and its modifications to be applicable for K-6 computational thinking activities. It also includes the design of a computational thinking activity using the new framework, and an example of working through that activity. Finally, it briefly details future work, such as other activities and their use in a professional development workshop for teachers.

KEYWORDS

Computational thinking, design thinking, K-12, C-D-I-O, Standards: 1, 3, 5, 7, 8

INTRODUCTION

The Information and Communication Technology (ICT) sector in Canada is a $74 billion per year industry (ICTC, 2015) that contributes to Canada's growth. The Information and Communications Technology Council released a report in 2016 which stated that 182 000 skilled digital workers will be needed by 2019; however, current domestic and international graduates will not meet this demand (ICTC, 2016). This report includes a national strategy to develop Canada's ICT talent, and one of its recommendations is to engage elementary and secondary youth in Science, Technology, Engineering, and Math (STEM) and ICT activities. A main component of ICT is computational thinking, which was defined by Jeannette Wing as "[involving] solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (Wing, 2006, p.33).

Many barriers exist that make it difficult to include computational thinking and programming activities in elementary classrooms, from the absence of a national curriculum to a lack of interest in programming from both students and parents (Wong, Ching, & Huen, 2015). However, a large barrier is the absence of teachers' training and teaching competency in coding and computational thinking (Wong et al., 2015). In Alberta, Canada, a report showed that less than half of Alberta elementary teachers had university education in math and science subjects (Rowell & Ebbers, 2004). Even if students are keen on the subject of computational thinking and a curriculum is put into place, elementary teachers may lack self-efficacy in the subject and struggle to implement computational thinking activities that teach the necessary content without decreasing time spent in other subjects.

This paper proposes a framework that is a modified version of the Conceive-Design-Implement-Operate steps from the CDIO Syllabus (Crawley, Lucas, Malmqvist, & Brodeur, 2011). The framework aims to support teachers in the design and implementation of computational thinking activities for K-6 students. The contributions of this paper are as follows:

- A method for modifying the existing CDIO Syllabus for other subjects
- A modified CDIO framework for teaching computational thinking to K-6 students and teachers
- An example activity created using the new framework, and a walk-through of the designed activity

The rest of this paper is organized into sections. First, background in computational thinking and K-12 CDIO applications is given. Next, the method for modifying the CDIO syllabus is described, as well as how the computational thinking activities may be designed. The new framework is then discussed, and put into context with a worked example of one of the designed activities. Finally, conclusions and future work will be briefly discussed.

BACKGROUND

Computational Thinking

Following Wing's 2006 definition of computational thinking, there have many different definitions of computational thinking over the years. Grover & Pea (2013) state that the following concepts are generally considered a part of computational thinking:

- Abstractions and pattern generalizations (including models and simulations)
- Systematic processing of information
- Symbol systems and representations
- Algorithmic notions of flow control
- Structured problem decomposition (modularizing)
- Iterative, recursive, and parallel thinking
- Conditional logic
- Efficiency and performance constraints
- Debugging and systematic error detection

Beyond the concepts, Brennan and Resnick also identify two other dimensions of computational thinking: computational thinking practices, which are the problem-solving practices that occur in the process of programming; and computational thinking perspectives, which involves the students' understandings of themselves, their relationships to others, and the technological world around them (Brennan & Resnick, 2012).

Computational thinking has been taught many ways, both in informal learning settings and inside formal education systems. Informally, computational thinking has been taught in after school coding clubs (Kafai, Fields, & Burke, 2008), museums (Horn, Leong, & Block, 2012), and public spaces (Sengupta & Shanahan, 2016). In formal education, computational thinking and programming are often taught at the high school level as an elective, though it is a mandatory subject in some European countries (European Schoolnet, 2015). There is less adoption of computational thinking at the elementary level. It can be seen in European countries including Slovakia, Estonia, and Finland (European Schoolnet, 2015), and especially in England, where the traditional ICT curriculum has been replaced with computer science, information technology, and digital citizenship topics (Berry, 2013). Topics for students of this age include understanding algorithms, writing simple programs, debugging, and understanding how search results are ranked (Berry, 2013). Currently, no such curricula for computational thinking exists nationally in Canada, though some provinces have made steps to include coding in their elementary schools (Province of British Columbia, 2016).

Computational thinking and engineering design are intertwined. Lee et. al. (2011) state that computational thinking has elements of mathematical, engineering, and design thinking, and can also be used to extend those skills. As well, modeling is understood to be a design activity as it involves selecting variables and phenomena, developing a representation, and then testing that representation (Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013). As CDIO is a design thinking framework, it can be used to teach and extend computational thinking concepts and practices.

CDIO in K-12 Education

While the CDIO Syllabus is mainly intended for use in post-secondary engineering programs, it has also been used at the K-12 level. For example, high school students have been involved in CDIO-based robotics projects (Arboleda, Pe, & Casta, n.d.). There are few uses of CDIO at the elementary (K-6) level. An egg-drop challenge was conducted in Sweden to strengthen design, building, and testing skills in 11-year-old students (Traff, Wedel, Gustafsson, & Malmqvist, 2007). Another study involved using the Conceive-Design-Implement-Operate steps to design and implement learning activities about electricity for grade 5 students (Marasco & Behjat, 2013). In that case, the students and teachers themselves were not explicitly using CDIO in those activities.

CDIO has also been used in training teachers. University students studying a BSc in science and technology education used the CDIO approach to balance pedagogy, engineering fundamentals, and teaching practice (Verner, 2015).

METHOD

Creating the Modified CDIO Framework

To begin, an in-depth look at the CDIO Syllabus was needed. The four stages of C-D-I-O are broken down into components, and each of these components contains various considerations. The components are highlighted in Table 1 below.

Table 1 - Sections 4.3 to 4.6 of the CDIO Syllabus (Crawley et al., 2011)

4.3 – Conceiving, Systems and Engineering Management	4.4 - Designing	4.5 - Implementing	4.6 - Operating
4.3.1 – Understanding needs and setting goals 4.3.2 – Defining function, concept and architecture 4.3.3 – System engineering, modeling and interfaces 4.3.4 – Development project management	4.4.1 – The Design Process 4.4.2 – The design process phrasing and approaches 4.4.3 – Utilization of knowledge in design 4.4.4 – Disciplinary design 4.4.5 – Multidisciplinary design 4.4.6 – Design for sustainability, safety, aesthetics, operability and other objectives	4.5.1 – Designing a sustainable implementation process 4.5.2 – Hardware manufacturing process 4.5.3 – Software implementing process 4.5.4 – Hardware software integration 4.5.5 – Test, verification, validation and certification 4.5.6 – Implementation management	4.6.1 – Designing and optimizing sustainable and safe operations 4.6.2 – Training and operations 4.6.3 – Supporting the system life cycle 4.6.4 – System improvement and evolution 4.6.5 – Disposal and life-end issues 4.6.6 – Operations management

Next, a 7-step programming problem-solving approach from Cornell University (Hilton & Bracy, 2015) was mapped to the existing CDIO structure. This process was done in order to see what steps exist in traditional courses which aim to teach programming, a subject that requires computational thinking. Finally, the new framework was built by considering both of the existing frameworks. Items of CDIO that would likely be beyond the scope of an elementary school project (i.e., system engineering, disposal issues, etc.) were removed. As well, components that were not applicable to K-6 computational thinking activities (such as hardware implementation) were also stripped. It is important to note however that these components may still be present in a computational thinking activity, such as using Lego Mindstorms to program a robot. Finally, terminology and language was simplified such that it is understandable and applicable to not only the K-6 teachers, but to their students as well.

Designing the Computational Thinking Activities

Once the new framework was completed, activities were designed which fit into the framework and would be applicable for K-6 teachers and their students. It was important to have activities that used digital technology, as well as those that did not. As well, activities that require the use of digital technology should not use paid software, as that may limit the ability of teachers to implement that activity in their classrooms. The two non-computer activities are simple and can be done with students of any age; the activities can be completed by pre-reader students. The other two activities use free software, and are a mix of block-based and text-based programming.

Students as young as grade 5 may already be losing interest in science, technology, engineering, and math subjects (Arnot, James, Gray, Rudduck, & Duveen, 1998; Bussiere, Cartwright, & Knighton, 2004). It is important to include opportunities for these students to engage with computational thinking outside of the STEM disciplines. Therefore, it was deemed necessary for these activities to be cross-disciplinary and tie into other school subjects outside of STEM, such as fine arts. This also provides the benefit of teachers being able to teach computational thinking alongside their mandatory curricula.

RESULTS

Modified CDIO Framework

The 7-step approach to solving programming problems from Cornell University was entirely mapped into three CDIO components, as shown in Table 2. It can be seen that this approach is entirely encapsulated in the CDIO Design step. Therefore, extra care should be taken in designing activities to ensure that they cover each of the Conceive, Design, Implement, and Operate steps.

Table 2 - Cornell 7-step approach mapped to CDIO Syllabus

CDIO Step	4.4.4 – disciplinary design	4.5.3 – software implementation process	4.5.5 – test, verification, validation and certification
Cornell 7-step approach	1. work example by hand • make sure the problem is fully specified • may require domain knowledge 2. write down what you did 3. find patterns • generalize your steps 4. check by hand test your algorithm to ensure your logic is sound	5. translate to code	6. run test cases • to uncover errors in the algorithm or its implementation 7. debug failed test cases • may need to go back and redesign the algorithm or check code translation/implementation

Table 3 contains two columns: the first column includes the original CDIO framework. The breakdown for each component is not included due to space requirements, but can be found in the CDIO Syllabus. The second column is the proposed adapted CDIO framework for use in teacher professional development in computational thinking.

Table 3 - Proposed modified CDIO framework for K-6 computational thinking

CDIO Syllabus 2.0 (Engineering)		**Computational Thinking/Programming for K-6 (Proposed)**
4.3 – CONCEIVING	4.3.1 – understanding needs and setting goals	-define the problem to be solved -determine what the program/algorithm needs to do -performance metric/rubric (how will we know if it worked?) -societal context
	4.3.2 – defining function, concept and architecture	-what functions are needed -determine what technology to use
	4.3.3 – system engineering, modeling and interfaces	
	4.3.4 – development project management	-consider schedules, time limits -allocate resources, both human and technological -consider risks and alternatives
4.4 – DESIGNING	4.4.1 – the design process	-requirements for each element or component derived from system level goals and requirements -alternatives in design -the initial design -life cycle consideration in design -experimental prototypes and test articles in design development -appropriate optimization in the presence of constraints -iteration until convergence -the final design -accommodation of changing requirements
	4.4.2 – the design process phrasing and approaches	
	4.4.3 – utilization of knowledge in design	-use technical and scientific knowledge -different types of thinking, including problem solving, inquiry, creative thinking, critical thinking -consider standardization

		-using prior work (reusing code)
	4.4.4 – disciplinary design	-consider the appropriate programming language -model the task/program (i.e. pretend to be the robot, step through the program)
	4.4.5 – multidisciplinary design	-interactions between disciplines
	4.4.6 – design for sustainability, safety, aesthetics, operability and other objectives	-reliability -consider sustainability, safety -digital literacy/citizenship -code readability, is it easy to understand? (aesthetics)
4.5 – IMPLEMENTING	4.5.1 – designing a sustainable implementation process	
	4.5.2 – hardware manufacturing process	
	4.5.3 – software implementation process	-the breakdown of high-level components into module designs (including algorithms and data structures) -algorithms (data structures, control flow, data flow) -the programming language and paradigms -the low-level design (coding) -the system build
	4.5.4 – hardware software integration	
	4.5.5 – test, verification, validation and certification	-does it work according to the design? -does it solve the problem?
	4.5.6 – implementation management	-consider process improvements: was there a better way to do the programming?
4.6 – OPERATING	4.6.1 – designing and optimizing sustainable and safe operations	-sustainable, safe, secure operation -digital literacy/citizenship
	4.6.2 – training and operations	-training to use final product
	4.6.3 – supporting the system life cycle	
	4.6.4 – system improvement and evolution	-consider ways to improve the product
	4.6.5 – disposal and life-end issues	
	4.6.6 – operations management	

Example Activity and Framework Mapping

Of the four activities developed using the framework, an activity titled *Scratch Stories* will be discussed in detail. It will highlight the role the modified CDIO framework played in its creation and the framing of guiding questions and steps to help teachers work through the C-D-I-O steps for the activity. Those questions are written in a such a way that they are appropriate for both the teachers and their students.

Scratch Stories uses MIT's Scratch (http://scratch.mit.edu), which is a block-based program that is said to be "low floor, high ceiling" (Papert, 1980). This means that it takes very little knowledge to write a basic program in Scratch, but at the same time it is possible to create very advanced projects. Scratch was chosen as it is well-known in the K-12 environment, is free, and can be completed on a browser rather than requiring downloaded software. Visual and block-based programming languages are preferred for K-6 students are they are not syntax-dependent and often sound like spoken English.

Scratch Stories is a cross-curricular activity which ties together the computational thinking concepts with English Language Arts requirements of story writing. Teachers will use MIT's Scratch program to write a story with a beginning, middle, and end. This story must have at least two characters. This provides a strong basis for all four C-D-I-O steps; teachers must *conceive* what their story is about before *designing* the scenes, characters, and actions in detail. They must then *implement* the story using Scratch. Finally, they will *operate* their own story and the stories of other teachers and students, taking care to ensure that it can reset itself, and thinking of ways it can be improved.

This activity is intended to be the third activity in a professional development workshop about computational thinking for K-6 teachers. Previous activities will have introduced computational thinking and its concepts such as sequences, loops, parallelism, events, conditionals, operators, and data (Brennan & Resnick, 2012). These concepts will have also been explained in the context of real-life algorithms. For example, recipes may have sequences (what you have to do in what order), parallelism (cutting a vegetable while boiling the water), and events (putting it in the oven once it reaches a certain temperature). Teachers will also have put these concepts into practice while learning about the two non-computer activities that require modelling tasks, such as navigating a maze or doing a dance as a robot. *Scratch Stories* makes the jump to digital devices and block-based programming, where teachers will still be using the concepts they are familiar with, but in an environment that may feel closer to their prior perceptions of programming.

Before beginning the activity, teachers are given a walkthrough of Scratch using a narrated screen capture. It covers the basics of Scratch and three examples of increasing difficulty that highlight different aspects of Scratch in relation to the computational thinking concepts learned previously.

The following sections will detail the researcher's experience as she works through each of the C-D-I-O steps for *Scratch Stories*.

Conceive

The Conceive step is essentially the storyboarding process. It asks questions such as: what will the story be about? What do the characters need to do? It also asked participants to think about how they should be graded (meeting requirements? enjoyment?) and what time limit they have for completing the project.

The researcher has decided that her story will be about two aliens who have built a spaceship to come to Earth. She has informed this decision by noting the various space-related sprites and backgrounds, as seen in Figure 1 below. She will need two different alien characters, a spaceship, and some kind of space background. The characters will need to be able to talk, move, and disappear so that it looks like they are in the spaceship. The spaceship will also have to move. There is no time limit for her project, and she will consider it a success if she creates a story that takes at least 30 seconds to run and is enjoyable.

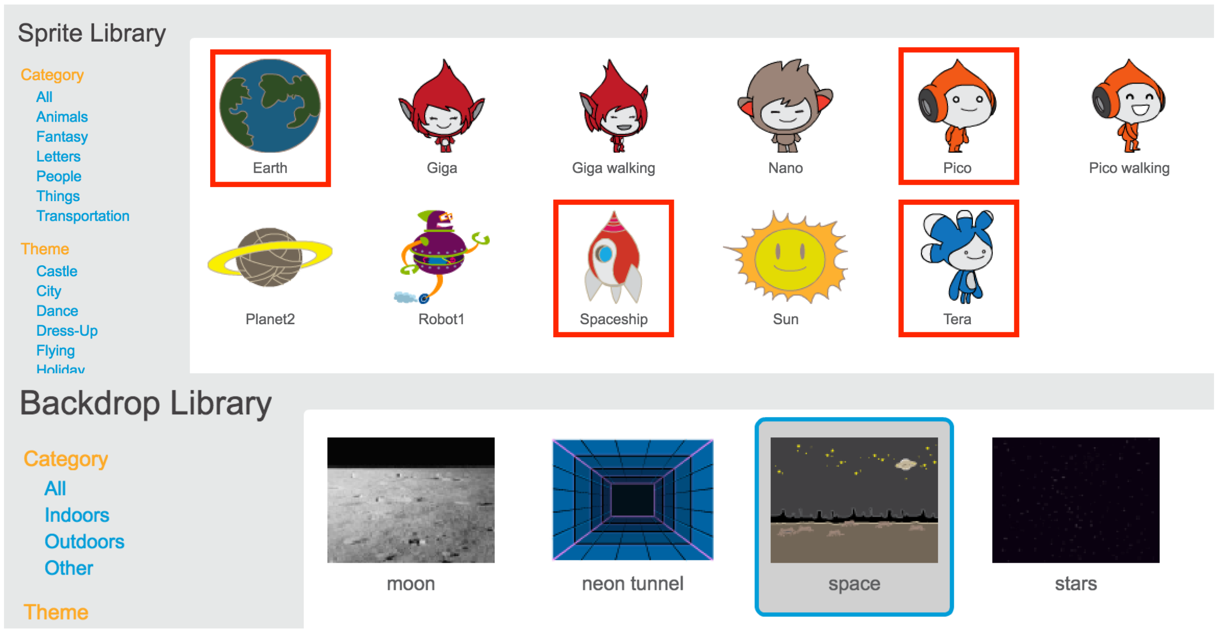


Figure 1 - Space sprites and backdrops in Scratch

Design

Next, the researcher begins designing her story by considering the requirements of each scene. She can then break down those scenes into actions to be coded separately. Different actions and scenes are based on the available Scratch blocks. She does not have any previous code from a project to reuse, though she could check Scratch's online database of projects to see what others have done. In order to figure out the scenes, she will need to model each scene from the point of view of each character, to break down what each character will need to do and say. Finally, she should consider how to reinitialize the scenes such that the story can be replayed over and over again easily.

Scene 1 (Beginning): Tera finishes working on her spaceship and announces it is done (speaking)!

Scene 2 (Middle): Pico walks over (moving) to check out Tera's good work (speaking), and tearfully says goodbye to his friend (more speaking).

Scene 3 (Middle): Tera gets into the spaceship (turns invisible using the 'show' block) and flies away (spaceship must move).

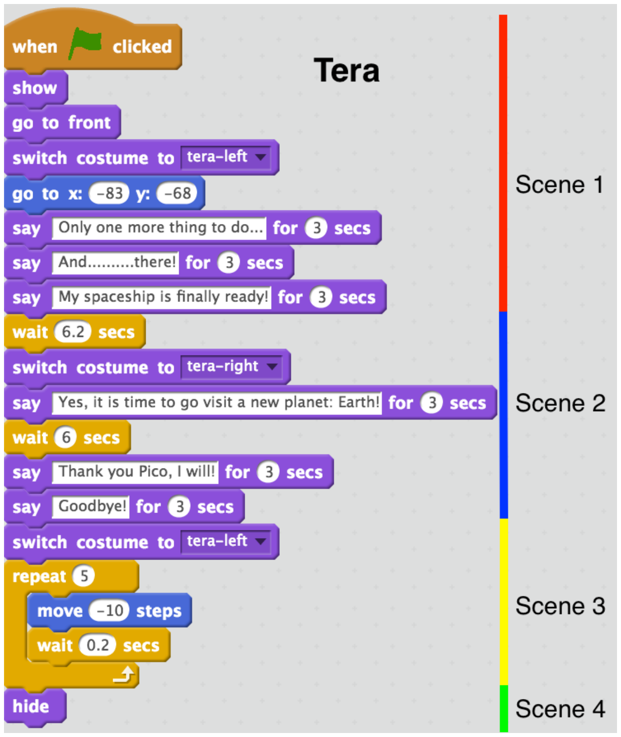
Scene 4 (End): Spaceship is seen moving in the stars towards Earth (will need a backdrop change, spaceship moving, and the Earth sprite to appear).

Implement

The Implement step is where all of the coding occurs. It should be done scene by scene in steps, coded separately, and then brought together at the end. Scene changes, sprite costumes, and timing should all be considered here to make the final story flow nicely. Again, resetting the story must be considered, especially since the researcher should be testing as she goes to ensure everything is working properly.

Various challenges were encountered while writing the code, such as Tera answering before Pico had even asked a question, Pico walking the wrong way, and trying to figure out exactly when the spaceship should start moving. Tera originally wasn't facing Pico during the conversation, which required a costume change so that she could look in the correct direction. It was also difficult to make Pico disappear and the Earth appear when the background changes to "stars". Rather than requiring on timing, a conditional statement was used that made the code more complicated. It seemed to work well for Pico disappearing, but the Earth was appearing too early in the story. In the end, a workaround was used as it was decided that it was a good thing for the Earth to be seen in the sky on the distant planet anyways. Finally, some text was created that would show up when the script was over, telling the user how to restart the story.

Figure 2 below shows the code written for Tera in the story, with the different scenes highlighted. The full project and code for all characters can be found at: https://scratch.mit.edu/projects/140885571/



```
when [flag] clicked
show
go to front
switch costume to tera-left
go to x: -83 y: -68
say Only one more thing to do... for 3 secs
say And..........there! for 3 secs
say My spaceship is finally ready! for 3 secs
wait 6.2 secs
switch costume to tera-right
say Yes, it is time to go visit a new planet: Earth! for 3 secs
wait 6 secs
say Thank you Pico, I will! for 3 secs
say Goodbye! for 3 secs
switch costume to tera-left
repeat 5
    move -10 steps
    wait 0.2 secs
hide
```

Figure 2 - Code for Tera in Scratch Project

Operate

Finally, the researcher should let others know how to operate the story. In this case, clicking the green flag at the beginning starts the story, and she has included some text that pops up at the end to tell users they can restart the story by clicking on the green flag again. She can consider how to improve or extend the story, such as having a scene with Tera landing on Earth and meeting humans for the first time. Finally, as Scratch projects can be posted online, she should consider digital citizenship and safety. She has not included any personal information in the story, and is using the internet in a safe and responsible manner. Her story has met her requirements as set in the Conceive step; it is enjoyable and takes 38 seconds to run.

CONCLUSION

In conclusion, the Conceive-Design-Implement-Operate steps can be modified such that they are applicable to an audience outside of post-secondary engineering education, including K-6 teachers and their students. It can also be tailored towards a specific subject, such as computational thinking, to create a framework that can then be used in teacher professional development. The new framework can be used to create activities that teach computational thinking concepts along with engineering design thinking. These activities can then be used in a professional development workshop for teachers.

Future work will see the implementation of this framework and activities within a professional development context. Quantitative data will be collected using surveys to determine changes in teachers' perceptions of computational thinking and their self-efficacy regarding the subject. As well, the modified CDIO framework will be used as a basis for directed content analysis for teachers' written reflections about their experience in the professional development workshop.

REFERENCES

Arboleda, H., Pe, E., & Casta, L. (n.d.). Engaging Engineering Students with an Introductory CDIO-Based Robot Programming Course (pp. 1–7).

Arnot, M., James, M., Gray, J., Rudduck, J., & Duveen, G. (1998). *Recent research on gender and educational performance*. London.

Berry, M. (2013). *Computing in the national curriculum A guide for primary teachers*. Retrieved from http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Bussiere, P., Cartwright, F., & Knighton, T. (2004). *The performance of Canadas youth in Mathematics, Reading, Science and problem solving: 2003 first findings for Canadians aged 15*. Ottawa.

Columbia, P. of B. (2016). *Area of Learning: APPLIED DESIGN, SKILLS, AND TECHNOLOGIES*.

Crawley, E. F., Lucas, W. A., Malmqvist, J., & Brodeur, D. R. (2011). The CDIO Syllabus v2.0 An Updated Statement of Goals for Engineering Education. In *Proceedings of the 7th International CDIO Conference*. Copenhagen, Denmark: Technical University of Denmark.

European Schoolnet. (2015). Computing our future. Retrieved from http://www.eun.org/c/document_library/get_file?uuid=3596b121-941c-4296-a760-0f4e4795d6fa&groupId=43887

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field.

Educational Researcher, *42*(1), 38–43. http://doi.org/10.3102/0013189X12463051

Hilton, A., & Bracy, A. (2015). *All of Programming*. Ithaca, New York, USA: Cornell University.

Horn, M., Leong, Z. A., & Block, F. (2012). Of BATs and APEs: an interactive tabletop game for natural history museums. *Proceedings of the …*, 2059–2068. http://doi.org/10.1145/2207676.2208355

ICTC. (2015). Strengthening Canada's Digital Advantage: Quarterly Monitor of Canada's Digital Economy | Summer 2015.

ICTC. (2016). Digital Talent Road To 2020 and Beyond a National Strategy To Develop Canada's Talent in a Global Digital Economy.

Kafai, Y. B., Fields, D., & Burke, Q. (2008). Entering the Clubhouse : Case Studies of Young Programmers Joining the Scratch Community. *Journal of Organizational and End User Computing (JOEUC)*, *22*, 21–35. http://doi.org/10.4018/joeuc.2010101906

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., … Werner, L. (2011). Computational Thinking for Youth in Practice. *ACM Inroads*, *2*(1), 32–37.

Marasco, E., & Behjat, L. (2013). Integrating creativity into elementary electrical engineering education using CDIO and project-based learning. *2013 IEEE International Conference on Microelectronic Systems Education, MSE 2013*, 44–47. http://doi.org/10.1109/MSE.2013.6566701

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Rowell, P. M., & Ebbers, M. (2004). *Elementary Science Education in Alberta Schools*. Edmonton, Canada.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380. http://doi.org/10.1007/s10639-012-9240-x

Sengupta, P., & Shanahan, M. (2016). STEM as Public Computation and Boundary Play.

Traff, P.-A., Wedel, M. K., Gustafsson, G., & Malmqvist, J. (2007). To Rescue Eggs; a Design-Build-Test Experience for Children. In *Proceedings of the 3rd International CDIO Conference*. Cambridge, USA.

Verner, I. M. (2015). Technology Teacher Education and Outreach using the CDIO Approach.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35. http://doi.org/10.1145/1118178.1118215

Wong, G. K. W., Ching, E. C. C., & Huen, J. M. H. (2015). School Perceptions of Coding Education in K-12 : A Large Scale Quantitative Study to Inform Innovative Practices, (December), 5–10.

BIOGRAPHICAL INFORMATION

Stephanie Hladik, is a M.Sc student in Electrical Engineering at the University of Calgary. Her research focuses on the inclusion of engineering education in K-12 curricula. In particular, she is interested in the ways that computational thinking and programming can be taught at the K-6 level, considering both the students and their teachers. Outside of her research, Stephanie is involved in many outreach programs to get K-12 students excited about STEM and alongside her colleagues has recently founded a company, LATTICE Development Inc., to bring research-tested cross-curricular STEM activities to schools.

Laleh Behjat, PhD, PEng, is an associate professor in the Department of Electrical and Computer Engineering at the University of Calgary. Her research interests include developing software for the automation of the design of computer hardware. She has won several awards for her research in international contests and forums. Dr. Behjat has a passion for increasing the status of women in science, technology, engineering and mathematics (STEM) and has an active research in the area of Engineering Education. Dr. Behjat was the recipient of the 2015 Association of Professional Engineers and Geoscientists of Alberta (APEGA) Women in Engineering Champion Award.

Anders Nygren, holds an MSc (Electrical Engineering) from the Royal Institute of Technology, Stockholm, Sweden; an MSEE from the University of Houston, Texas; and a PhD from Rice University, Houston, Texas. Prior to joining the Schulich School of Engineering in 2004, he completed postdoctoral training in Physiology & Biophysics at the University of Calgary. Dr. Nygren has taught courses in Biomedical Engineering and in the Common Core Engineering program at the Schulich School of Engineering.

Corresponding author

Stephanie Hladik
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N 1N4
skhladik@ucalgary.ca