

運算

CTE²⁰¹⁷ International Conference on
Computational Thinking Education

13-15 July 2017

思維

Conference Proceedings

Created & Funded by:

Co-created by:



香港賽馬會慈善信託基金
The Hong Kong Jockey Club Charities Trust
同心 同步 同進 RIDING HIGH TOGETHER



香港教育大學
The Education University
of Hong Kong



Massachusetts
Institute of
Technology



香港城市大學
City University of Hong Kong

Editor

Siu-cheung KONG

The Education University of Hong Kong, Hong Kong

Josh SHELDON

Massachusetts Institute of Technology, The United States

Robert Kwok-yiu LI

City University of Hong Kong, Hong Kong

Copyright 2017 The Hong Kong Jockey Club

All rights reserved.

ISBN: 978-988-77034-4-0

ISSN: 2664-035X (CD-ROM) / 2664-5661 (online)

Publisher

The Education University of Hong Kong

Preface

International Conference on Computational Thinking Education 2017 (CTE2017) is the first international conference organized by CoolThink@JC, created and funded by The Hong Kong Jockey Club Charities Trust, and co-created by The Education University of Hong Kong, Massachusetts Institute of Technology, and City University of Hong Kong.

CoolThink@JC strives to inspire students to apply digital creativity in their daily lives and prepare them to tackle future challenges in any fields. Computational thinking (CT) is considered as an indispensable capability to empower students to move beyond mere technology consumption and into problem-solving, creation and innovation. This 4-year initiative educated over 16,500 upper primary students at 32 schools on computational thinking through coding education. Moreover, through intensive professional training, the Initiative develops the teaching capacity of 100 local teachers and helps them master the coding and computational thinking pedagogy. Over time the project teams target to affect greater change by sharing insights and curricular materials with policymakers and educators in Hong Kong.

“Computational Thinking Education” is the main theme of CTE2017 which aims to keep abreast of the latest development of how to facilitate students’ CT abilities, and disseminate findings and outcomes on the implementation of CT development in school education. It comprises keynote and invited speeches by internationally renowned scholars, panel discussions, academic paper presentation, booth exhibition on STEM/Coding products and solutions, and student and teacher poster presentation and demonstration.

CTE2017 gathers educators and researchers around the world to share implementation practices and disseminate research findings on the systematical teaching of computational thinking and coding across different educational settings. There are 15 sub-themes under CTE2017, namely:

Computational Thinking

Computational Thinking and Unplugged Activities in K-12

Computational Thinking and Coding Education in K-12

Computational Thinking and Subject Learning and Teaching in K-12

Computational Thinking and IoT

Computational Thinking Development in Higher Education

Computational Thinking and STEM/STEAM Education

Computational Thinking and Non-formal Learning

Computational Thinking and Psychological Studies

Computational Thinking and Special Education Needs

Computational Thinking and Inclusive Society

Computational Thinking and Early Childhood Development

Computational Thinking in Educational Policy

Computational Thinking and Teacher Development

General Submission to Computational Thinking Education

The conference received a total of 43 papers (25 long papers, 12 short papers and 6 poster papers) by authors from 13 countries (see Table 1).

Table 1: Distribution of paper submissions for CTE2017

Country/Region	No. of submission	Country/Region	No. of submission
The United States	15	United Kingdom	1
Hong Kong	9	Canada	1
Taiwan	3	Israel	1
Singapore	3	Australia	1
Malaysia	3	China	1
Germany	2	Spain	1
Korea	2		

Each paper with author identification anonymous was reviewed by at least 3 International Program Committee (IPC) members. Related sub-theme chairs were responsible to conduct meta-reviews and make final decisions on the submitted papers based on IPC members' recommendations. With the comprehensive review process, the conference accepted 17 long papers, 12 short papers and 8 poster papers (see Table 2).

Table 2: Review results of submission acceptance for CTE2017

Sub-theme	Long paper	Short paper	Poster	Total
Computational Thinking	0	1	1	2
Computational Thinking and Coding Education in K-12	1	1	1	3
Computational Thinking and Subject Learning and Teaching in K-12	3	1	1	5
Computational Thinking and IoT	0	1	0	1
Computational Thinking Development in Higher Education	1	0	0	1
Computational Thinking and STEM/STEAM Education	5	1	1	7
Computational Thinking and Non-formal Learning	2	0	0	2
Computational Thinking and Psychological Studies	1	1	1	3
Computational Thinking and Inclusive Society	0	0	1	1
Computational Thinking and Early Childhood Development	2	1	0	3
Computational Thinking and Teacher Development	1	1	2	4
General Submission to Computational Thinking Education	1	3	1	5
TOTAL	17	12	8	37

CTE2017 has three conference days comprising five keynote speeches, two invited speeches, three panel discussions, five academic paper presentation sessions, booth exhibition on STEM/Coding products and solutions, and student and teacher poster presentation and demonstration.

Keynote and Invited Speeches

CTE2017 has invited five internationally renowned scholars as the conference keynote speakers: (1) Prof. Hal ABELSON from Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, The United States (“Computational Thinking, Computational Values, Computational Actions”); (2) Ms. Marjorie YANG from Esquel Group, Hong Kong (“Why is Computational Thinking Education Important as the Foundation for Innovation?”); (3) Prof. Tom CRICK from Cardiff Metropolitan University, The United Kingdom (“‘It’s not the Coding Curriculum’: Embedding Computational Thinking into England’s New Computing Curriculum”); (4) Dr. Shuchi GROVER from SRI International, The United States (“Computational Thinking in K-12: Considerations for Pedagogy and Assessment”); and (5) Prof. Uri WILENSKY from Northwestern University, The United States (“Transforming Knowledge and Learning through Agent-Based Modeling”).

The conference has also invited two internationally renowned scholars as the conference invited speakers: (1) Prof. Gautam BISWAS from Vanderbilt University, The United States (“CTSiM: A Computational Thinking Environment for Learning Science using Simulation and Modeling”); and (2) Ms. Eliane METNI from International Education Association, Lebanon (“Empowering Teachers to Nurture Computational Thinking and Innovation in K-12”);

Panel Discussions

CTE2017 has three panel discussions: (1) “Computational Thinking and Education Policy” (Facilitator: Prof. Siu-Cheung KONG from The Education University of Hong Kong, Hong Kong); (2) “Promotion of Computational Thinking Development in Local School Education” (Facilitator: Principal Tsz-wing CHU from Baptist Rainbow Primary School, Hong Kong); and (3) “STEM Education and Computational Thinking Development” (Facilitator: Dr. Daner SUN from The Education University of Hong Kong, Hong Kong).

Poster Presentation and Booth Exhibition

CTE2017’s Poster Presentation aims at showcasing worldwide researches on computational thinking education and CoolThink@JC achievement in the first year. There are four poster categories, namely international academic poster papers, posters presenting local teachers’ reflection prepared by The Association of I.T. Leaders in Education (AiTLE), posters displaying local students’ achievement presented by The Hong Kong Association for Computer Education (HKACE) and posters about 2017 CoolThink@JC Competitions participating teams.

The academic poster papers cover diversified international computational thinking research outcomes highlighting the importance and pathways of computational thinking development in aspects covering K-12 education, teacher development and STEM education.

The posters prepared by AiTLE share with participants on how Hong Kong teachers design and

implement lessons and post-lesson activities on computational thinking development, coding education and STEM / STEAM education. Teachers' pedagogy approaches, teaching methods and schemes, and reflection are also illustrated with real classroom examples.

For the posters presented by HKACE, students show their computational thinking learning experiences of different events such as Hong Kong Olympiad in Informatics (HKOI) and IT Challenge Award (ITCA). Also, they explain how these learning activities equip students with programming and problem solving skills.

This year the CoolThink@JC Competition attracted more than 100 applications from 40 primary schools. In the first round of the competition, the participating teams produced a 3-min video to describe the problem they would encounter in their daily lives and then apply their knowledge and skills on Computational Thinking to produce a creative and innovative solution with coding technique. 30 teams were shortlisted in June to enter the final of the competition which will be held in October 2017. The videos of these 30 finalists are showcased here at the conference poster display session.

Apart from poster display and presentation, there are 20 booth exhibitions, among which 4 are in-charged by 12 CoolThink@JC Cohort One School teachers and students demonstrating students' actual learning outcomes and how they benefit from computational thinking education. Students also share their experience with participants on designing Apps and programming robots. The other 16 booths are conducted by local technological companies and organizations related to computational thinking education and development. They exhibit a wide range of STEM/coding solutions and products.

On behalf of the Conference Organizing Committee, we would like to express our gratitude towards all members for their contribution to the success and smooth operation of CTE2017.

We sincerely hope everyone would enjoy and get inspired from CTE2017.

On Behalf of CoolThink@JC

Siu-cheung KONG

The Education University of Hong Kong, Hong Kong

Conference Chair of CTE2017

Tsz-wing CHU

Baptist Rainbow Primary School, CoolThink@JC Resource School, Hong Kong

Conference Chair of CTE2017

Table of Contents

Computational Thinking	1
Evaluations of Programming Complexity in App Inventor	
Lisa L RUAN, Evan W PATTON, Mike TISSENBAUM.....	2
An investigation into susceptibility to learn computational thinking in post-compulsory education	
Ana C. CALDERON, Tom CRICK , Catherine TRYFONA	6
Computational Thinking and STEM/STEAM Education	10
Assessing Students’ Computational Thinking in a Learning by Modeling Environment	
Ningyu ZHANG, Gautam BISWAS	11
Computational Thinking in the Science Classroom	
Hillary SWANSON, Gabriella ANTON, Connor BAIN, Michael HORN, Uri WILENSKY	17
Constructing Models in Physics: What Computational Thinking Occurs?	
Sarah POLLACK, Bruria HABERMAN, Orni MEERBAUM-SALANT	23
Domain Specific Modeling Language Design to support Synergistic Learning of STEM and Computational Thinking	
Asif HASAN, Gautam BISWAS.....	28
The Role Gender Differences in Computational Thinking Confidence Levels Plays in STEM Applications	
Nicole M HUTCHINS, Ningyu ZHANG, Gautam BISWAS	34
K-12 Computational Thinking Education in Germany	
Nguyen-thinh LE, Niels PINKWART.....	39
Computational Thinking and Subject Learning and Teaching in K-12	44
Gamified Mathematics practice: Designing with e-commerce and computational concepts	
Chien-sing LEE, Jing-wen WONG, Peh-yenc EE,.....	45
How Computer Scientists and Computing Teachers Think Differently in the Concepts to be Included in a Secondary School Computing Curriculum	
Chiu-fan HU, Cheng-chih WU, Yu-tzu LIN, An-tsu WANG	50

Teaching Computational Thinking by Gamification of K-12 Mathematics: Mobile App Math Games in Mathematics and Computer Science Tournament	
Chee-wei TAN, Pei-duo YU, Ling LIN, Chung-kit FUNG, Chun-kiu LAI, Yanru CHENG	55
Profile of a CT Integration Specialist	
Joyce MALYN-SMITH, Irene A. LEE, Joseph IPPOLITO	60
Enhancing the Link between Parent-Child in Learning Computational Thinking	
Jane Yat-ching WONG, Pam Hau-yung WONG, Robert Kwok-yiu LI, Chee-wei TAN,	64
Computational Thinking and Teacher Development	66
Teaching Computational Thinking with Electronic Textiles: High School Teachers’ Contextualizing Strategies in Exploring Computer Science	
Deborah A. FIELDS, Debora LUI, Yasmin B. KAFAI	67
Application of the Four Phases of Computational Thinking and Integration of Blocky Programming in a Sixth-Grade Mathematics Course	
Ting-chia HSU, Hsin-chung HU	73
The Design and Evaluation of a Teacher Development Programme in Computational Thinking Education	
Siu-cheung KONG , Ming LAI , Josh SHELDON , Mike TISSENBAUM	77
Connecting Design Thinking and Computational Thinking in the Context of Korean Primary School Teacher Education	
Hyungshin CHOI, Mi-song KIM.....	81
Computational Thinking and Coding Education in K-12	83
Curriculum Activities to Foster Primary School Students’Computational Practices in Block-Based Programming Environments	
Siu-cheung KONG, Hal ABELSON, Josh SHELDON, Andrew LAO, Mike TISSENBAUM, Ming LAI, Karen LANG, Natalie LAO.....	84
Emergent Roles, Collaboration and Computational Thinking in the Multi-Dimensional Problem Space of Robotics	
Florence R. SULLIVAN, P. Kevin KEITH.....	90
A Framework of Computational Thinking Curriculum for K-12 with Design Thinking by App Inverntor	
Peng CHEN, Ronghuai HUANG	94

Computational Thinking and Psychological Studies.....	96
Development and Validation of a Programming Self-Efficacy Scale for Senior Primary School Learners Siu-cheung KONG	97
Computational Thinking as a Key Competence – a Research Concept Amelie LABUSCH, Birgit EICKELMANN	103
Can Music Exposure Enhance Computational Thinking? Insights from the Findings on the Music-Creativity Relations Mei-ki CHAN, Wu-jing HE, Wan-chi WONG	107
Computational Thinking and Early Childhood Development	117
Imagining, Playing, and Coding with KIBO: Using Robotics to Foster Computational Thinking in Young Children Amanda A. SULLIVAN, Marina UMASCHI BERS, Claudia MIHM	110
Programming with ScratchJr: a review of the first year of user analytics Kaitlyn D. LEIDL, Marina UMASCHI-BERS, Claudia MIHM	116
Technology Strategy Mapping in My First Skool Childcare Centres, Singapore Ai-ling THIAN, Belinda CHNG, Meei-yen LONG	122
Computational Thinking Development in Higher Education	134
Integrating Computational Thinking into Discrete Mathematics Kwong-cheong WONG	127
Computational Thinking and Non-formal Learning	132
Computational Thinking Affordances in Video Games Sue-inn CH'NG, Yunli LEE, Wai-chong CHIA, Lee-seng YEONG.....	133
You Can Code – An innovative approach to transform the workforce in the textile and apparel industry Bessie CHONG,	139
Computational Thinking and IOT	144
Off the Screen, and Into the World of Everyday Objects: Computational Thinking for Youth with the Internet of Things Mike TISSENBAUM, Josh SHELDON, Evan PATTON, Arjun GUPTA, Elaine ZHANG, Divya GOPINATH.....	145

Computational Thinking and Inclusive Society	146
Developing interest to share and craft based on the Technology Acceptance Model	
Chien-sing LEE, Samuel Hong-shan LOW	150
 General Submission to Computational Thinking Education	153
Complementary Tools for Computational Thinking Assessment	
Marcos ROMÁN-GONZÁLEZ, Jesús MORENO-LEÓN, Gregorio ROBLES	154
 App Inventor VR Editor for Computational Thinking	
Jane IM , Paul MEDLOCK-WALTON, Mike TISSENBAUM.....	160
 Computational Thinking and Coding Initiatives in Singapore	
Peter SEOW, Chee-kit LOOI, Bimlesh WADHWA, Longkai WU, Liu LIU.....	164
 Enabling Multi-User Computational Thinking with Collaborative Blocks Programming in MIT App Inventor	
Xinyue DENG, Evan W. PATTON.....	168
 Evidences of Self-Development of TAs in CT Education	
Ray CHEUNG, Ron Chi-wai KWOK, Matthew LEE, Robert LI, Chee-wei TAN.....	172

Computational Thinking

Evaluations of Programming Complexity in App Inventor

Lisa L. RUAN, Evan W PATTON, Mike TISSENBAUM

Massachusetts Institute of Technology
llruan@mit.edu, ewpatton@csail.mit.edu, mtissen@mit.edu

ABSTRACT

To understand computational thinking in App Inventor, it is important to be able to effectively evaluate computational complexity in block-based programming languages. In the past, there have been a handful of complexity measures proposed for text-based languages (Weyuker, 1988). In this paper, we will attempt to implement 2 such measures, Halstead's Programming Effort and statement count, in App Inventor on a dataset of projects from 50 random users. The goal is to determine whether or not text programming standards for complexity can be generalized to block programming languages. This paper shows that the 2 complexity measures we implemented are not adequate measures for complexity in App Inventor. This result indicates a need for different measures of complexity that more accurately portray block programming proficiency. We hope this study will be a gateway into a better understanding of the intricacies of App Inventor's block programming language and its unique contributions to the development of computational thinking.

KEYWORDS

Data Analytics, Computational Thinking, Block-based Programming, Programming Complexity, App Inventor

1. INTRODUCTION

In an increasingly automated age, there is a growing recognition for individuals in all walks of life, not just programmers, to develop their computational thinking (Wing, 2006). Computational thinking is generally understood as the ability to recognize and solve problems using computational means (Brennen & Resnick, 2012) In this spirit, we aim to further understanding of how learners build computational thinking by analyzing the differences and similarities between block-based programming languages and text-based programming languages.

In this paper, we apply a data-driven approach to apply text programming complexity standards to block-based programming and discuss the implications of the resulting complexity progression for each sampled user, as well as several App Inventor-specific takes on the meaning of complexity, and possible directions for future work.

2. BACKGROUND

2.1. App Inventor

App Inventor is a block-based programming language (Glinert, 1986) that aims to increase access to programming capabilities and further the reach of programming education by simplifying programming concepts with visually intuitive blocks. The design of visual blocks in these

environments make the flow of logic and programming easier to understand for young learners (Weintrop & Wilensky, 2015). App Inventor leverages this programming approach to allow users to create fully functional mobile applications for Android devices.

App Inventor has nearly 3 million users from 195 different countries and has given rise to more than 7 million android apps (<http://appinventor.mit.edu/explore/about-us.html>). App Inventor's audience stems from a variety of backgrounds including educators, designers, researchers, government workers, and entrepreneurs; as such, the App Inventor dataset is rich in quantity and breadth. Also, since it occupies a niche between everyday logical thinking and traditional programming languages, it is an optimal language to study for conclusions on the progression of computational thought.

2.2. Software Complexity

In his 1977 book, *Elements of Software Science*, Maurice Howard Halstead introduced a software metric intended to measure program complexity and give structure to the understanding of the software development process. This metric, aptly dubbed "Halstead's programming effort" aims to compute the time (defined as "effort") a user takes to create a program by analyzing the relationship between operators, operands and their appearances in a program. In this study we define operands as bodies of information and operators as functions that interact with those bodies of information. This measure focused on the relationship between the total number of operands/operators and the number of unique operands and operators. The calculation for programming effort (E) was performed in 2 parts: difficulty (D) and volume (V). The difficulty and volume were defined as follows:

Given: $n1 = \# \text{ distinct operators}$; $n2 = \# \text{ distinct operands}$; $N1 = \text{total } \# \text{ operators}$; $N2 = \text{total } \# \text{ operands}$

$\text{Effort} = n1/2 * N2/n2 * (N1+N2)*\log_2(n1+n2)$

Thus, this difficulty measure rewards programs that use less distinct operands and instead reference previous operands. In other words, more consolidated programs will evaluate with higher complexity. For example, a program that utilizes one central function to execute similar procedures would be more complex than a program that defines each procedure individually. In the blocks given below, the first row represents a more complex program than the 2nd row.

In 1988, this measure was evaluated by Elaine J. Weyuker, who concluded 2 main drawbacks. First, the effort to write one program P may be greater than the effort to write a composite program P;Q. Second, the effort measure makes no use of statement order. We combat the first drawback

by employing another complexity measure, statement count (also mentioned in the same paper), in addition to the effort measure. We assume the second drawback holds less importance in terms of programming effort in App Inventor since App Inventor is a block-based language and statement order is generated by compiler code.

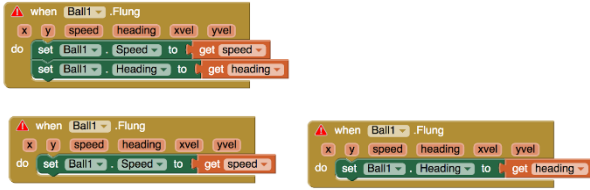


Figure 3. Complex vs not complex block programs

3. METHODOLOGY

3.1. Data Source, Sampling, File Types

For this study we looked at users who had created at least 20 projects, which we define as “experienced” users. This was so we would have sufficient data to analyze learning progression for each user. Attempting this type of analysis on users who have only created very few (such as 1 or 2) projects would be unreliable and would not give us a reasonable understanding of progression over time. It is important to note that we looked at users' entire programming history, not just their high complexity projects. Experienced users were at some point also beginners, and thus their project history also contains projects representative of beginner behavior.

From this pool of experienced users, we randomly sampled 50 and extracted their projects. Since the sample is completely random, we assume no knowledge of any demographical effects. We filtered out projects with more than one screen, since we believe multiple screens offer data transferring capabilities that should be further studied separately. Projects that used multiple screens accounted for less than 20% of all projects, so we still retained a large part of our dataset. A small number of corrupted or outdated projects without timestamps were omitted.

3.2. Built-in Blocks vs. Component Blocks

Within App Inventor, there are 2 broad classifications for blocks. First, built-in blocks are available for use in any program and are universal in application and are divided into 8 categories; control, logic, math, text, lists, colors, variables, and procedures.

The other group contains component blocks, which are specific to the components, or parts of the app. Components are akin to features of an app. For example, an app might use a text box. The text box as a component has component-specific blocks, such as a block that will change the font size of the text in the text box. Such blocks are very component-specific, and can vary wildly depending on the components that are used in a program. There are more than a hundred components, and each can have more than 20 component-specific blocks. As such, while built-in blocks can be tabulated and manually analyzed, component blocks are treated a little differently and grouped according to behavior. We will highlight the differences in analysis in section 3.4.

3.3. Implementing Measure Complexity

Our first approach to programming complexity was to use a statement count. The statement count represents the size of the program and acts as a naive measure of complexity. In App Inventor we define a statement as a block. To implement the statement count, we simply iterate through each xml file (which we convert to an element tree using the python element xml tree library) and count all the block declarations we have.

3.4. Programming Effort

Our second approach to programming complexity, Halstead’s programming effort, is much more complex and nuanced. In order to adequately implement this complexity measure, we went through several steps.

3.4.1. Operator vs. Operand

First, we needed a clear way to differentiate operators and operands. In this paper, we assume that every block is either an operator or an operand. This seems intuitive for component blocks as each component block interacts with the component in some way, so they must either retrieve information (which we will classify as an operand) or change the component somehow (which we will classify as an operator). The component blocks come in 4 different categories with the following classifications; methods, component blocks (reference blocks for the entire component), events, and set/get blocks (which modify or retrieve component variables, respectively). In the above order, we classified all blocks in the subcategories as operators, operands, operators, and operators for set blocks and operands for get blocks.

We mentioned above that we assume all blocks are either operators or operands. The distinction is not as obvious for built-in blocks, so we manually classified each built-in block as an operand or operator.

3.4.2. Determining Uniqueness and Final Calculations

For component blocks, each component block has an internal mutation (properties of blocks that allow them to change shape or slightly alter function) regardless of whether or not it has an external mutation, and this internal mutation combined with the block definition uniquely determines the block identity. For built-in blocks, we assumed each block tag is unique save for blocks that have dropdown menus that change the operation, definition blocks, and large grouped blocks defined by all parameters (such as “procedures_callnoreturn”), which we differentiated using additional block properties.

We also found that, due to the nature of App Inventor, it is possible to create many functioning programs that only use operators and thus cannot be evaluated in the original effort formula. We propose 2 treatments of such cases. The first and naive treatment is to assume that the effort made by the user is 0. This is clearly not always true, as there are many perfectly functioning apps that use no operands (such as the App Inventor tutorial “Hello Purr”). The second approach is to assume that the ratio of operators to operands is 1. In this case we end up with the following formula:

$$E = (N1) * \log_2(n1) * n1/2$$

3.5. Creating a picture of a user

Once we have functions to calculate the complexity of each project, we combine the complexity and timestamps of each project to create a graph of the progress of complexity of each user. At this point we note the timestamps of the projects we previously filtered out (projects with multiple screens or malformed files) and include them in the graphs as breaks in the graph. This is to preserve accuracy when comparing users against each other. An example of how such a graph would appear is shown next.



Figure 4. Statement count vs. project number
Above, the 6th project is not included in the data set.

4. RESULTS

Figure 5 plots the statement count vs. project number for all fifty sampled users (i.e. 4 on the x axis indicates a user's fourth project). Figure 6 represents the effort measure vs. project number.

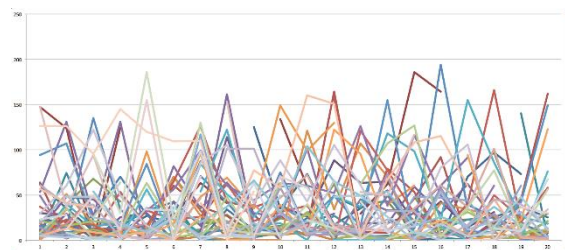


Figure 5. Statement count vs. project number

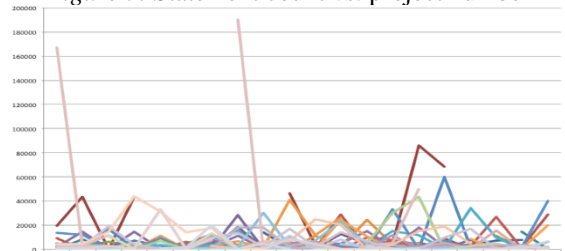


Figure 6. Effort vs. project number

Both sets of graphs are noisy, punctuated with peaks and appear as if there is no clear pattern of complexity progression as defined in both Halstead's effort measure and the statement count.

5. DISCUSSION

Results show that Halstead's programming effort and line count do not provide adequate insight into how users are becoming more proficient at block programming. As such, to understand complexity in block programming languages we cannot simply apply existing text programming language metrics. Given that metrics are invaluable for understanding important measures such as complexity, this paper is an important step towards developing measures that can help us better understand how people's code evolves in block programming. This in turn, has important implications for understanding the different ways to build computational thinking. Knowing that text and block

programming languages have some key differences highlights the need to adapt learners' approaches to computation. Below we discuss the implications of these findings for blocks-based languages, with a particular focus on several key features of App Inventor.

5.1. App Inventor Components

A central difference between App Inventor and traditional text programming languages is its feature-based approach to programming. Let us first elaborate on the programming process in App Inventor. There are 2 main interfaces of App Inventor, the designer and the blocks editor. The Designer is used to modify the layout of elements (components) on the screen - text boxes, buttons, sound players, etc. while the blocks editor lets a user access actual programming blocks, such as if statements and booleans. To create a functional app, a user must first drag and drop components onto the screen in the Designer view, and then decide how the rest of the programming blocks are going to interact with the apps' components. The bulk of the computational thinking needed to create a program involves this latter step of communicating with components through component blocks and using them in tandem with built-in blocks to create solutions.

This way of programming embeds a mental distinction between components and other variables and makes it possible to make valid programs that don't use any operand blocks at all. If we consider such programs, since the operators are communicating with information from the components, one approach could be to treat components as operands. However, this doesn't make sense because components are clearly more complex operands than an average block. In this sense components might act not as variables but as information sinks and sources. This means users' patterns of interaction with components and blocks are fundamentally different from text programming languages where everything is treated equally.

5.2. Effects of Visual Intuition

Another major aspect of block programming languages not available in text programming languages is the visual implications of the blocks themselves. Since the blocks can be placed in a 2-dimensional space, it is possible users may be clustering blocks according to their computational thought process. The 2D space adds an extra layer of consideration to the organization and subsequent understanding of blocks and how they relate to each other.

5.3. Rigidity

Users of App Inventor are restricted to the blocks that exist in App Inventor. This is very different from text programming, where there is a lot of freedom to define whatever functions or methods are needed. Computational thinking in this constrained environment could be different than in a more open text programming environment and thus affecting our results.

5.4. A Problem-Solving Mentality

It is likely many users are approaching App Inventor with the purpose of solving a specific problem, rather than to create increasingly complex projects. Thus, the complexity

of the project will likely depend on the complexity of the problem itself and may not represent a user's proficiency in using the language. This kind of understanding may require finer grained analysis comparing the problem space and an individual user's solution in comparison to an "expert's" solution to the problem.

5.5. Consequences in Educational Programs

Studying the intricacies of computational thought behind block programming languages may enable us to better understand how to design curricula for a given purpose. For example, if the end goal of a program was to teach text-based programming using App Inventor as an introduction, then it is imperative to understand the differences between the complexity of block-based programs and text programs so we could better design the transition from one to another (Parsons & Haden, 2007).

5.6. Classification of Blocks

In the process of implementing the complexity measures we also began classifying blocks in App Inventor as operands and operators. This is a shift from treating blocks as a means to achieve a product or desired function towards treating blocks as more traditional computational elements. This allows us to analyze blocks in App Inventor as a different type of computational thinking. Instead of analyzing if a user can create a specific end product using given blocks, we can focus on how users might be perceiving each block if they are building intuition for text-based programming concepts.

6. LIMITATIONS AND FUTURE WORK

Due to additional questions about data transfer, we did not include any programs with multiple screens, which is another possible area for us to study. Nevertheless, we hope the findings of this work will pave the way for other investigations of programming complexity in block-based programming languages, as well as opportunities for further research on communication between different bodies of information and how they relate to complexity.

6.1. New Measures of Complexity

In section 5, we mentioned many properties of App Inventor are unique and different from most text programming languages. A logical next project would be to create a new measure that accurately captures these differences and

allows us to more accurately evaluate the computational complexity of apps created in App Inventor. Possible directions could be to focus on component-block interaction or visual clustering of blocks. Other approaches include row or column organization, or novel ways of using existing blocks to replicate text programming functions that do not exist in App Inventor. Even more possible measures include the speed of adaptation of new App Inventor features, or the amount of data transfer between screens.

7. CONCLUSION

Text programming complexity standards, in particular Halstead's Programming Effort and Statement count, are not very applicable in determining App Inventor fluency. This paper is a call to action for more studies on revealing the nuances of complexity in block-based languages and towards providing insight into factors such as the physical organization of code blocks within a program.

8. REFERENCES

- Wing, J. (2006). *Computational Thinking. Communications of the ACM: Viewpoint*, vol. 49, 33-35.
- Brennan and Resnick. (2012). *American Educational Research Association annual meeting*. Vancouver, BC.
- Glinert, E. P. (1986). *Towards second generation interactive graphical programming environments. In Proceedings of IEEE Workshop on Visual Language. IEEE CS Press, Silver Spring, MD* 61-70.
- Weintrop, D., & Wilensky, U. (2015, July). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. *ICER*, vol. 15, 101-110.
- Halstead, M. H. (1977). *Elements of software science*. New York: Elsevier.
- Weyuker, E. (1988). Evaluating Software Complexity Measures. *IEEE Transactions On Software Engineering*, VOL. 14. 1357-1365.
- Parsons, D., & Haden, P. (2007). Programming osmosis: Knowledge transfer from imperative to visual programming environments. *S. Mann & N. Bridgeman (Eds.), Proceedings of The Twentieth Annual NACCCQ Conference*. Hamilton, New Zealand. 209-215.

An investigation into susceptibility to learn computational thinking in post-compulsory education

Ana C. CALDERON^{1*}, Tom CRICK¹ and Catherine TRYFONA¹

¹ Department of Computing & Information Systems, Cardiff Metropolitan University, Cardiff CF5 2YB, UK
{acalderon, tcrick, ctryfona}@cardiffmet.ac.uk

ABSTRACT

This paper presents the results of a preliminary investigation into how the teaching of computational thinking -- particularly algorithmic thinking and programming -- to university undergraduate students varies depending on aptitude and perceived enjoyment of STEM subjects during their secondary-level (pre-university) education. We investigated a specific component of computational thinking, algorithmic thinking, comparing against a student's ability to develop knowledge and understanding of introductory programming.

KEYWORDS

Perceptions, Algorithmic thinking, Computational thinking, STEM

1. INTRODUCTION

Computational thinking [Papert 1996; Guzdial 2008; Wing, J. (2008)] is increasingly being integrated into various national curricula, being regarded as a key skills, with wide potential utility, for school-age children. It is recognised both for its important role in developing knowledge and understanding of foundational computer science concepts, but also for its potential in developing more general-purpose problem-solving skills across the curriculum. This paper investigates whether algorithmic thinking (an integral part of computational thinking) can be as easily taught to those with a natural interest in computational science and those who do not process such an interest, and whether this changes with aptitude to more technical subjects in school. Aptitude and interest are restricted as to what students preferred subjects subjects were at the time of secondary school graduation.

There are many views of computational thinking, for instance a recent report of a workshop shows the range of definitions, and opinions on the subject (NRC 2010) Some researchers adopt the original notions of procedural thinking, as developed by (Papert 1981) to define what Computational Thinking is. This view sees it as a step-by-step list of detailed and unambiguous instructions such that can be interpreted and executed by an automated agent. Others view it as an effort to expand the human capacity for problem solving, by providing abstract tools able to aid in the management of tackling complex tasks. A lot of researchers also dismiss the notions of linking computational to the processing of numbers, whereas some argue it is a way of enabling humans to solve problems by means of providing precise methods for doing so. Whatever viewpoint adopted, most researchers seem to agree that computational thinking is an integral part of computer

science [Tedre 2016]. The skill set learn by studying Computational Thinking is complementary to more established areas taught at HE computing degrees. This investigation looks at students' aptitudes to STEM and Humanities in the final two years of school, in an attempt to see whether there are negative or positive correlations to leaning elements of Computational Thinking and of a core element of Computing degrees, programming. Focusing particularly on algorithmically thinking and on object-oriented programming, we found that an aptitude in STEM favoured performance in learning object-oriented programming notions, but found no difference between aptitudes in humanities and in sciences when learning Algorithmically Thinking (Futschek 2006) with a methodology highlighted in later sections.

2. Methodology

2.1 The Research Question

Our interest is on whether particular preferences in secondary school have a positive correlation with ability to learn algorithmically thinking in Higher Education. Using the methodology above we measured data gathered from students about attitudes and aptitudes of STEM-based and other subjects and how well they performed on the particular algorithm course.

2.2 Pedagogical Investigation

The investigation took part over two semesters in one academic year; one semester the students participated in an algorithm class, and the second semester different students participated in an object-oriented programming class. The choice for using different groups of students was due to the transfer of knowledge, performance in a latter module, for instance object-oriented programming could have been enhanced by attending an earlier, for instance, algorithmic thinking module.

We designed a one semester course such focusing on teaching algorithmic thinking to first-year, first-semester students enrolled in three undergraduate degree programmes: Computer Science, Software Engineering and Business Information Systems. Students participated in a total of 11 weekly sessions, where each session consists of three components, distributed during the week.

Algorithmic Thinking

The sessions consisted of:

- Part A consists of a one hour session (workshop) of a hands-on puzzle solving activity.
- Part B consists of a formative learning session (a one hour lecture)

- Part C consists of a one hour session (workshop) of a puzzle that includes writing pseudocode.

For the workshops (Parts A and B) students were required to work in groups. The first session was purposely kept simple, and we now use it as an example of the methodology, it consisted of:

- Part A (workshop): present students with physical copies of Tower of Hanoi puzzles with a large number of even and of odd disks.
- Part B (lecture): lecture on recursion
- Part C: (workshop) Tower of Hanoi puzzles handed out to students again, and asked them to write pseudocode to solve a Tower of Hanoi with either an even or an odd number of disks (students who do not immediately recognize recursion are given extra support until they are able to connect the concept from the lecture to the example from the workshop).

For another illustrative example, we detail the second session. The main aim behind this session was to develop understanding of sorting algorithms. Students were given cardboard pieces with numbers written on it, ranging 1-100, and asked to find the maximum. Following the same pattern as all other sessions, students were placed in groups. Differently from other sessions, they were asked (in their groups) to first think about attempting to find the maximum value of the numbers (sorting the cards) if they could only work by themselves, then if they could only work within the group, and finally to think about how they would solve if the groups could talk to each other and divide the cards. The idea behind this is to aid participants in teaching themselves what an algorithm is as well as to bring their awareness to the existence of parallelism as a means to efficiency. This session is based on ideas developed in (Adams 2005).

For the formative learning portion of the session students were taught the concept of a sorting algorithm and presented with some standard examples of sorting algorithms, namely insertion sort, selection sort, merge sort, heapsort, quicksort, bubble sort and variants. For the final workshop (Part B) of this particular session, students were given Rubik's cubes and given 3 sequences of moves, then asked to use these sequences to solve the cube, and write a pseudocode for their solution (an algorithm that would sort all sides to the desired configuration).

Programming

Teaching introductory programming within Higher Education can be particularly challenging due to the diversity of educational background of incoming undergraduate students, as a single annual intake of students is likely to include a broad range of prior learning experiences. As a consequence of school-level computer science education reform (Brown *et al*, 2014), an increasing number of first year students are likely to have had some exposure to programming in schools or colleges. Some students, perhaps through their own extracurricular efforts, may have developed considerable technical skills. This variance in ability seemingly increases the risk of disengagement because the teaching material may either be viewed as too difficult (Mohd *et al*, 2013) or too simplistic.

It could be argued, however, that software development and programming is an art as much as it is a science and that undergraduate students can best develop their programming skills through apprentice-style learning (Kolling and Barnes, 2008; Bennedsen and Caspersen, 2008). Recently, there has been more emphasis placed on the importance of "software carpentry" skills, so that student can develop a sense of "craftsmanship" towards the design and development of software solutions to real world problems. Seminars and tutorials can particularly lend themselves to this style of delivery, where experienced teaching staff are not only able to demonstrate the technical skills, but also explain the thinking behind the decisions that they make (Kolling and Barnes, 2008).

Given that sound computational thinking skills aids in most stages of the software development process, there is an increasing and explicit emphasis on developing these skills in modern undergraduate computing curricula. By focusing on key skills such as algorithmic thinking from early on in a programmer's career, students can more readily contextualise programming as a tool to be used for expression of creativity and for problem solving. Students are able to analyse problems and formulate a solution computationally (Cesar *et al*, 2017). An emphasis on computational thinking within the context of apprentice-style learning, may reduce the risk of disengagement as more technically-able skills will have the opportunity to refine their skills under the guidance of a more experienced academic member of staff.

Similarly to algorithmic thinking, the sessions were broken down into formative and practical learning, namely they consisted of:

- Part A consists of a formative learning session (a one hour lecture)
- Part B consists of a two hour practical session (coding the concepts learnt in the lecture).

In particular, during the term each week (note that each week contained Part A together with Part B), was given by:

- Week 1: Introduction to programming, including varying programming paradigms.
- Week 2: Introduction to integrated development environments.
- Week 3: Understanding how to perform operations, and their implications to varying paradigms.
- Weeks 4 and 5: Understanding statements and directing values.
- Week 5: Manipulating Data.
- Weeks 6, 7 and 8: Object Oriented concepts.

3. Results

We compared students' aptitude to STEM subjects and humanities at both A-levels and GSCE with their ability to learn algorithmic thinking, with the methodology highlighted above. More specifically, we focused on students who had grade C and above at a combination of mathematics, computing and physics at A-level, and those who had a grade C and above at a combination of history, literature and drama. The performance of both groups was similar; the first group had an average grade of 62.4%, with

a standard deviation of 13.4, whereas the humanities group had an average grade of 61.3% with a standard deviation of 9.4 (see Figure 1 for more details). Of the 92 students used for the first study (algorithmic thinking), 23 had taken the requirements of aptitude in the three stem subjects: mathematics, computing and a science subject, and 17 satisfied the requirements of having taken the humanities English literature, history and drama. For the second study (programming) 21 had taken the requirements of aptitude in the three stem subjects: mathematics, computing and a science subject, and 18 satisfied the requirements of having taken the humanities English literature, history and drama. Although the difference between STEM and humanities for the algorithmic group was significantly small, the difference for a more traditional approach to teaching object-oriented programming was more significantly different, the average programming grade for students with a STEM aptitude was 17.9%, with a standard deviation of 67.1, and those with an aptitude in humanities was 16.7% with a standard deviation of 47.5, more details can be found on Figure 1. This suggests that Computational Thinking approaches are more readily taught to varied skilled students, as compared to the core elements of Computer Science. This suggests that along side standard computer science subjects, HE students might benefit from having a dedicated module of "Computational Thinking" as that would "even the playfield" and thus allow educators to keep the levels of motivation similar to students regardless of their background. We also analysed their ability to write pseudocode.

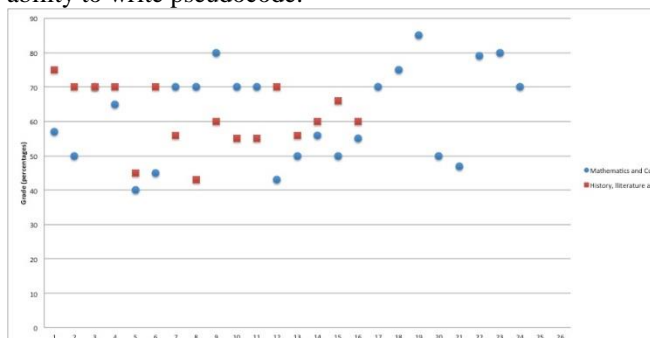


Figure 1. Distribution of grades for algorithmic thinking against humanities and STEM preferences at A-levels

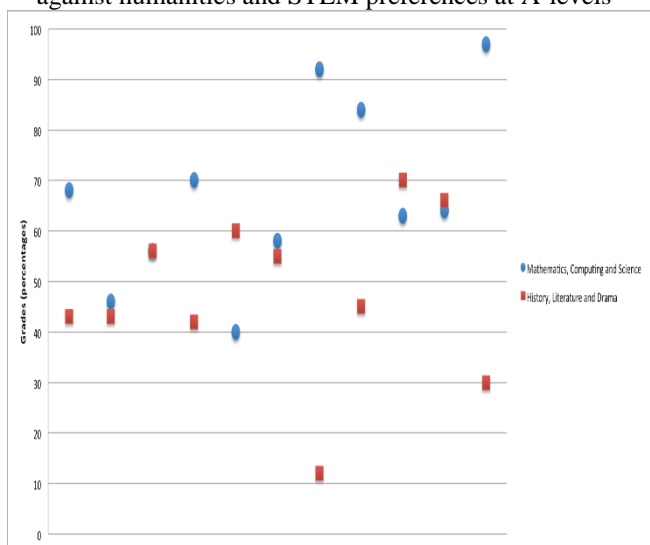


Figure 2. Distribution of grades for programming against humanities and STEM preferences at A-levels

4. CONCLUSION

We presented the beginnings of an on-going investigation into how susceptible students, of varying aptitudes and attitudes, are to learning computational thinking skills.

5. REFERENCES

- Adams, R., Bell, T., McKenzie, J., Witten, I. H., & Fellows, M. (2005). *Computer Science Unplugged: An enrichment and extension programme for primary-aged children*.
- Bennedsen, J., & Caspersen, M. (2008). Exposing the Programming Process. In J. Bennedsen, M. Caspersen, & M. Kolling (Eds.), *Reflections on the Teaching of Programming: Methods and Implementation*. New York: Springer.
- Brown, N., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The Resurgence of Computer Science in UK Schools. *ACM Transactions on Computing Education*, 14(2), 9:1–9:22.
- Cesar, E., Cortés, A., Espinosa, A., Margalef, T., Moure, J. C., ... Suppi, R. (2017). Introducing computational thinking , parallel programming and performance engineering in interdisciplinary studies ☆. *J. Parallel Distrib. Comput.* <http://doi.org/10.1016/j.jpdc.2016.12.027>
- Futschek, G. (2006, November). Algorithmic thinking: the key for understanding computer science. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 159-168). Springer Berlin Heidelberg.
- Guzdial, M. (2008). "Education: Paving the way for computational thinking". *Communications of the ACM* 51 (8): 25.
- Kolling, M., & Barnes, D. (2008). Apprentice-based Learning Via Integrated Lectures and Assignments. In J. Bennedsen, M. Caspersen, & M. Kolling (Eds.), *Reflections on the Teaching of Programming: Methods and Implementation* (p. -). New York: Springer.
- Matti Tedre and Peter J. Denning. 2016. The long quest for computational thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (Koli Calling '16). ACM, New York, NY, USA, 120-129. DOI: <https://doi.org/10.1145/2999541.2999542>
- Mohd, S., Shukur, Z., & Mohamad, H. (2013). Analysis of Research in Programming Teaching Tools: An Initial Review. *Procedia - Social and Behavioral Sciences*, 103, 127–135.
- National Research Council. (2010) *Report of a Workshop on the Scope and Nature of Computational Thinking*. Washington, DC: The National Academies Press. doi:10.17226/12840.. Chapter 2, page 4.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc..]

Seymour Papert, 1981, *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books)

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Wing, J. (2008) Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, 366(1881), 3717-3725

Computational Thinking and STEM/STEAM Education

Assessing Students' Computational Thinking in a Learning by Modeling Environment

Ningyu ZHANG*, Gautam BISWAS

Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
ningyu.zhang@vanderbilt.edu, gautam.biswas@vanderbilt.edu

ABSTRACT

Researchers have hypothesized strong connections between Computational Thinking (CT) practices and STEM learning. However, there is a lack of consensus on what constitutes an adequate set of CT knowledge and skills. In this paper, we present an initial framework for evaluating students' CT learning. We introduce the primary CT concepts and practices that students can learn and apply in a learning by modeling environment. Our overall goal is to develop assessments that study the synergy between STEM and CT concepts in K-12 curricula. Towards this end, we discuss the results from a teacher-led classroom study we conducted on STEM- and CT-learning in our CTSiM environment.

KEYWORDS

Computational thinking, learning by modeling, CT assessment, evidence-centered design, classroom study

1. INTRODUCTION

Computational thinking (CT) involves a collection of abilities and practices for solving problems analytically, thinking recursively, and using abstraction (Wing, 2006). CT could benefit communities beyond computer science practitioners, by drawing from fundamental skills and practices of various disciplines (Wing, 2006). In addition, CT can benefit teaching and learning in other domains, using skills and practices that originate within CT (Wing, 2011; Barr & Stephenson, 2011).

A series of studies have shown that appropriate use of CT skills and corresponding tools can deepen the learning of science, technology, engineering, and mathematics (STEM) subjects (García-Peñalvo et al., 2016). CT shares a reciprocally enriching relationship with math and science, meanwhile, synergistic learning of science and CT skills has been demonstrated through a series of studies (e.g., Weintrop et al., 2016a; Basu, Biswas & Kinnebrew, 2017). The deep CT skills can transfer to and benefit other learning and problem-solving contexts (Grover, 2015), as CT requires fundamental understanding and development of solutions rather than rote learning (Wing, 2006). Therefore, CT is essential for preparing students for future learning (Bransford, Brown, & Cocking, 2000).

These potential benefits of CT have led to the inclusion of CT into STEM classrooms. For example, the Next Generation Science Standards (NGSS) in the United States have included CT as a core scientific practice (The NGSS Lead States, 2013, Barr & Stephenson, 2011). However, although K-12 educators pushed for the

advancement of computing curricula, many aspects of CT concepts remained underrepresented in corresponding assessments (Grover, Cooper, & Pea, 2014). Therefore, fine-grained assessments are required to evaluate the subtle aspects of students' CT learning in STEM domains.

Our lab has developed *Computational Thinking using Simulation and Modeling* (CTSiM), a computer-based learning environment that promotes learning of science and computational thinking (CT) concepts and skills using a *learning by modeling* approach (Wilensky, Brady, & Horn, 2014; Sengupta et al, 2013). In this paper, we present an initial CT assessment framework linked to CTSiM and evaluate its effectiveness. The assessment framework defines key CT skills and practices that students need when they are building models in CTSiM, as well as the methods for assessing them. Section 2 reviews three aspects of relevant work from which we define the methodology used in this paper. Section 3 introduces CTSiM and the focal CT-related knowledge, skills, and practices that students need to learn and develop to become proficient model builders and problem solvers. In Section 4, we present a classroom study that was administered by a middle school teacher with no intervention from the researchers. In Section 5, we report the main results of (1) incorporating key CT components in CTSiM, and (2) assessing these components in the form of a case study. Finally, we discuss the implications of our results and future work in Section 6.

2. RELATED WORK

2.1. CT Constructs

Given the wide scope of CT, there has been little agreement among researchers on what constitutes CT (National Research Council, 2010; Brennan and Resnick, 2012). In addition, the close relationship between CT, mathematics, algorithmic thinking, and problem-solving skills also veils core ideas in computation that it encompasses (García-Peñalvo et al., 2016; Weintrop et al., 2016a).

To understand how programming supports the development of CT, Brennan and Resnick (2012) defined a framework for CT with three components: (1) computational concepts, (2) practices, and (3) perspectives. In this framework, computational concepts include the fundamental knowledge of a computing system, such as loops and conditionals; computational practices involve actions such as iterative building, testing, as well as debugging; and computational perspectives describe the learner's CT world-view (Brennan & Resnick, 2012). In addition to focusing on

what students learn about CT, Weintrop et al. (2016a) proposed key CT practices that are commonly applied in STEM domains that include (1) data, (2) modeling and simulation, (3) problem-solving, and (4) systems thinking. These CT practices define *how students learn CT* and provide a theoretical foundation for integrating CT in STEM classrooms.

2.2. Assessment of CT

Assessments provide information on how well students understand and apply the content they are taught. Such information can help instructors infer the effectiveness of their teaching and learning (Mislevy, Almond, & Lukas, 2003).

CT assessments have been applied in various learning domains, for example, authoring environments that cater programming game design activities for novice learners (e.g., Repenning, Ioannidou, & Zola, 2000; Berland et al., 2013; Moskal, Lurie, & Cooper, 2014; Weintrop et al., 2016b). For example, Scratch (Brennan & Resnick 2012) uses multiple means of assessment that involve analysis of student-created programming portfolios, artifact-based interviews, and design scenarios; meanwhile, AgentSheets (Ioannidou et al., 2011) uses reoccurring patterns in game design and science simulation contexts to evaluate students' understanding of CT. Despite the progress in advancing CT assessments, many fundamental aspects of CT have not received sufficient attention especially in the context of block-based programming environments (Grover et al., 2014). Therefore, more advanced test instruments need to be developed to enrich the CT assessment toolbox.

2.3. Evidence-centered Design of assessment

Evidence-centered design (ECD) is a methodology that emphasizes the use of evidentiary reasoning as the determining factor in designing assessments (Mislevy et al., 2003). Three components, i.e. the student model, the task model, and the evidence model, are essential while defining assessments under the ECD framework (Mislevy et al., 2003; Chrysafiadi & Virvou, 2013).

The student model consists of the knowledge, skills, and abilities (KSAs) that can be used to infer students' knowledge states. The task model describes a collection of tasks, their presentation material, and work products. The evidence model serves as the bridge between the student model and the task model that defines instructions on how a task response provides evidentiary information about the student's knowledge state (Mislevy et al., 2003). Since there is a lack of consensus in describing what constitutes CT constructs, our methodology presented in this paper eclectically draws from a set of key CT aspects presented in the literature to form CTSiM-specific knowledge, skills, and abilities (KSAs) (Mislevy et al., 2003). We give a detailed description of the student, task, and evidence models of CTSiM in Section 3.

3. THE LEARNING ENVIRONMENT

3.1. CTSiM

Open-ended learning environments (OELEs) have the potential to provide meaningful learning opportunities to students. While working with an OELE, students usually construct solutions to authentic problems. They may also generate and test hypotheses with artifacts (in the form of student-generated programs (Land 2000)).

CTSiM is an OELE that promotes synergistic learning of science and computational thinking (CT) concepts and skills using a *learning by modeling* approach (Sengupta et al., 2013). In CTSiM, students use block-structured constructs to model scientific scenarios using an agent-based framework (Wilensky, et al., 2014). Student models are converted into NetLogo simulations (Wilensky, 1999). The learning and model-building tasks in CTSiM involve five primary activities: (1) reading and comprehending domain contents and CT-related concepts from two built-in resource libraries; (2) building a conceptual model of the science scenario using an agent-based framework (defining the hierarchies of the agents' and their environment's properties and behaviors); (3) constructing computational models that define the agents' behaviors using a block-based visual programming language; (4) running their models as NetLogo simulations to analyze the behaviors generated; and (5) comparing their models' behaviors to an expert model that executes synchronously with theirs (Basu, Biswas, & Kinnebrew, 2017).

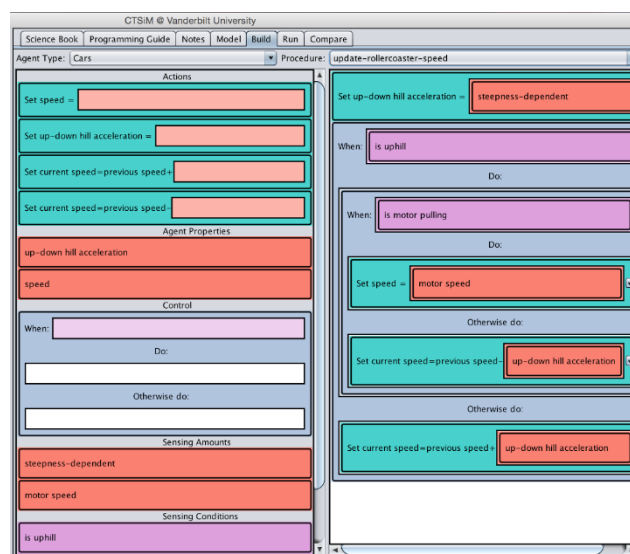


Figure 1. Computational model building interface.

CTSiM has a learning progression that consists of two introductory training activities and a series of modeling activities (Basu, Biswas, & Kinnebrew, 2017). Students begin by constructing shape-drawing agents in the two training units and then proceed to the primary learning and modeling activities that cover five science topics: kinematics, dynamics, collisions, diffusion, and ecology. Figure 1 shows a screenshot of CTSiM's user interface and the block-oriented domain-specific language for building the rollercoaster model (kinematics). The

learning activity on the foreground is constructing part of the computational model for the rollercoaster car agent.

3.2. Student, Task, and Evidence Models in CTSiM

We use the overlay model (Desmarais & Baker, 2012; Chrysafiadi & Virvou, 2013) to describe the states of students' knowledge. Students' mastery of CT aspects, as well as their ability to combine the CT constructs to solve complex problems in CTSiM, are inferred from their performance (i.e., whether or not they correctly answer a question). An overlay of students' correct answers on the student model captures their CT knowledge state. The learning gains between the pre- and post-tests indicates whether a student has improved his or her understanding of a CT concept. Although the value of learning gains does not necessarily associate with a probabilistic model, comparing an individual learning gain with the aggregated results from a classroom can give a reasonable measurement of how much a student's CT knowledge has progressed with respect to the average. Alternatively, one can also study students' progress by looking at their knowledge states through a series of assessments.

Summative paper-and-pencil pre- and post-tests on both the CT and the science domain topics constitutes CTSiM's assessment task outside of the system. In this paper, we focus on these paper-and-pencil CT assessments. It is noteworthy that the task model under the ECD assessment framework is different from the learning task model that involves modeling scientific scenarios and running simulations as described in previous work by our research group (e.g., Basu & Biswas, 2016). For the paper-and-pencil assessment, four types of question are administered: multiple choice, fill in the blank, short answer, and design. Each question will have a presentation format and an associated level of complexity.

The evidence model of CTSiM defines the grading rubrics for the summative test questions that human raters use in processing a student's responses on the tests. Each test question is associated with one or a combination of CT knowledge, skills, and abilities (e.g., determining which statement will be executed in a conditional structure). The evidence rules in the assessment can also update the question-KSA mapping in the student model when a student creates a work product (responding to a question).

4. STUDY SETTING

The data we analyze in this paper came from a classroom study with 37 eighth-grade students in the USA. The study lasted 9 days (one-hour per day) and was administered by a science teacher. Another purpose of conducting this study was to use the kinematics modeling activity of CTSiM to prepare for a hands-on activity of building paper rollercoasters in the teacher's science class. Prior to the study, we offered copies of CTSiM to the teacher and assisted her to become proficient with the functionalities of the learning environment.

On the first day (day 1) of the study, all participating students took paper-based pre-tests on CT skills and kinematics contents. On day 2 through day 4, the students

worked together as a class on the introductory units to familiarize themselves with the system's interface and basic concepts of agent-based modeling (e.g., agents and the environment, properties and behavior of agents). From day 5 to day 8, students worked individually on the rollercoaster modeling activity. Students modeled a rollercoaster car that moved on a track in 4 stages: (1) being pulled up by a motor at a constant speed, (2) accelerating along down slopes, (3) moving on a flat segment at a constant speed (ignoring friction), and (4) climbing up slopes and decelerating. As students built the computational representation of the motion of a rollercoaster car, they interacted with computational concepts, such as variables, if-conditionals, and loops. They also familiarized themselves with domain concepts such as acceleration, speed, distance, and their relationships. For example, that distance is a (linear) function of speed over time. On the last day, all students took the CT and science post-tests, which had the same questions as the pretest (the teacher and students never discussed the questions or solutions during the study).

5. RESULTS

We present a theoretical result and an analytical result in this paper. The theoretical result is a collection of key knowledge, skills, and abilities (KSAs) that we defined using ECD principles while drawing from the existing CT literature. We then analyze and discuss students' performance on the KSAs (as the analytical result). Vanderbilt researchers were not present during the study, and we did not collect any demographic information from the class. As a result, students' name, gender, and ethnicity are not known to us, so we cannot discuss issues, such as gender difference, in this paper.

5.1. CT Knowledge, Skills, and Abilities

With the methods described in Section 2, we define two categories of KSAs in CTSiM: (1) 4 key CT constructs and (2) 11 key CT skills and practices. The CT constructs in CTSiM are (1) sequential execution of statements, (2) loop structures, (3) conditionals, and (4) variables and assignments. The CT constructs consist of the most fundamental and domain-general computational block structures of CTSiM.

In addition, the CT skills and practices are: (1) gathering information; (2) defining the agents' properties as abstract conceptual models; (3) specifying in the conceptual model interface, environmental properties that affect agent behavior; (4) defining agent behaviors by building computational models; (5) Assessing student-constructed models by running simulations; (6) debugging models; (7) dividing problems into sub-problems; (8) modularizing and reusing computational solutions; (9) understanding relationships between variables in a system; (10) understanding systems at different levels of abstraction; and (11) solving inquiry problems using their models.

This collection of CT skills and practices is defined by synthesizing well-known CT frameworks (e.g. Brennan & Resnick, 2011; Weintrop et al, 2016a) and emphasizing CT aspects that are specific to CTSiM. For example,

gathering information is comparable to the *collecting data* practice in Weintrop et al (2016a); understanding systems is related to the *Systems Thinking* practices; and our agent-based-modeling-related CT skills and practices (No. 2, 3, 4, and 5) correspond to the *Modeling and Simulation* practices.

We believe the key CT constructs, as well as CT skills and practices, are necessary for the student to become successful in the learning activities in CTSiM. Some KSAs, although not directly linked to CT, also foster student’s learning. For example, while gathering information, students generate evidence as they read the two resource libraries in CTSiM. This information is necessary for understanding the domain content knowledge, building computational models, and reasoning about system behaviors. On the other hand, some KSAs involve metacognitive strategies and are difficult to assess in paper-and-pencil based assessments. For example, students divide problems into sub-problems in CTSiM as they work on the learning tasks and incrementally build computational models on a smaller scale. Yet this KSA cannot be directly assessed in our pre- and post-tests. Similarly, debugging and querying skills are not assessed as well.

To illustrate the questions asked in the pre- and post-tests, we present and briefly discuss question 3 as an example.

Q3 Consider the following program

If (quiz score is greater than 7)

Then: If (quiz score is equal to 10)

Then: Get the ‘You’re a pro’ sticker

Else: Get the ‘Good job’ sticker

Else: Get the ‘Try harder’ sticker

Bill gets a score of 9 on the quiz while Janet scores 10 points and Kim scores 5 points on the quiz. What stickers should each one receive?

This question assesses students’ understanding of nested if-conditional structures that requires them to analyze a conjunction of logic statements. Only when both conditions (“quiz score is greater than 7” and “quiz score is equal to 10”) evaluate to true is the statement “Get the ‘You are a pro’” executed.

We show the links between KSAs and the questions in our pre- and post-tests in Table 1. Based on the distribution, CT constructs are assessed in the format of multiple choice and fill-in-blank questions (Q1 through Q4), and CT skills and practices are mostly assessed as short answer questions and design code snippets (Q5 through Q8).

Table 1. KSAs assessed in CTSiM questions.

KSA		Appearance
Sequential (KSA1)	execution	All questions
Loop structures (KSA2)		Q1, Q6
Conditionals (KSA3)		All questions except Q1
Gather information (KSA4)		Q5, Q6, Q7, Q8
Define agent properties (KSA5)		Q6, Q7
Define agent behaviors (KSA6)		Q5, Q6, Q7, Q8
Define environment (KSA7)		Q6, Q7
Simulate w/ model (KSA8)		Q5, Q6
Divide and conquer (KSA9)		Q7, Q8
Modularize and reuse (KSA10)		Q7, Q8
Define relationships in systems (KSA11)		Q6, Q8
Define multi-agent systems (KSA12)		Q7

5.2. Summative Assessment Results

We then performed paired t-tests on the participating students’ pre-test and post-test scores. On an aggregated level, the students showed significant learning gains in CT ($p = 0.000025$). We also used Cohen’s d to measure the effect size associated with the learning gains. Table 2 summarizes the analysis of CT pre- post-test results.

Table 2. Means (and standard deviations) of pre- post assessment scores.

Pre-test	Post-test	t -stat	p -value	Cohen’s d
14.05 (2.36)	21.59 (2.47)	4.61	< 0.001	0.91

We also divided the aggregated results according to individual KSAs. Table 3 summarizes the students’ learning gains in each KSA. We discuss the results and their implication in the next section.

Table 3 . Average pre- post assessment scores (standard deviations) and p -values per KSA.

	Pre-test	Post-test	p -value
KSA1	14.05 (2.36)	21.59 (2.47)	< 0.001
KSA2	2.46 (2.13)	4.57 (2.10)	< 0.0001
KSA3	13.78 (8.70)	20.78 (7.81)	< 0.0001
KSA4	8.89 (7.07)	15.81 (7.76)	< 0.0001
KSA5	2.78 (3.08)	6.65 (3.63)	< 0.0001
KSA6	8.89 (7.07)	15.81 (7.76)	< 0.0001
KSA7	2.78 (3.08)	6.65 (3.63)	< 0.0001
KSA8	4.22 (2.36)	6.30 (2.12)	< 0.0001
KSA9	4.68 (5.55)	9.51 (6.59)	0.0002
KSA10	4.68 (5.55)	9.51 (6.59)	0.0004
KSA11	5.84 (5.65)	10.38 (5.65)	0.0002
KSA12	0.81 (2.26)	2.89 (2.76)	0.00012

6. DISCUSSION

From the results of the classroom study, we found that students not only achieved significant learning gains at the aggregated level, but also in each of the KSA's we defined using the ECD framework. As the existing work on CT education has stated, teaching CT content should not be carried out as a standalone subject that is isolated from the real world; instead, students should learn CT in the context of problem-solving and its application (Weintrop et al., 2016a). During the classroom study, the teacher documented a few anecdotes, which provided empirical evidence of benefits and rationales for continually integrating CTSiM into STEM classes.

To begin with, the teacher reported that the participants enjoyed the system. A girl told the teacher that she did not realize that she could fall in love with programming (Teacher: "I think one girl may have even found her calling in life, as she is a real 'natural' with the coding part and has never done it before"). Additionally, students benefited from CTSiM when they built actual paper rollercoasters. The teacher reported that the participants showed improvements compared to students in previous years, who only sketched and built the paper rollercoaster. For example, none of the students designed a loop at the beginning of the track, which was not uncommon among previous cohorts. The teacher also felt that the CTSiM activities helped her better manage the class because students more easily realized their own difficulties while interacting with the system and asked relevant and specific questions, making it easier for the teacher to adapt her scaffolding in a more effective manner. Finally, the teacher herself gained programming experience with the system. The study helped her become more comfortable with programming and agent-based modeling concepts.

The classroom case study shows that CTSiM is effective in helping middle students learn and improve their understanding of CT concepts and skills. In addition, CTSiM fitted well into the science classroom and helped students learn their science content better (Basu et al., 2016). To better define and assess CT with CTSiM, our future work will focus on (1) refining the CTSiM CT KSAs described in this paper to include concepts and practices from more studies; (2) increasing the CT assessment tools' coverage on these KSAs with questions that address concepts with a finer granularity (e.g. adding CT skills and practices such as debugging and resolving inquiry with computational models that are not currently being assessed), (3) delving into the test reliability and validity (e.g., showing that students behave similarly on questions covering same KSAs), and (4) aligning students' performance on the CT tests to characterizations of learning behaviors in CTSiM (Zhang, Biswas, & Dong, in press).

7. CONCLUSION

In this paper, we presented a case study of students' learning and using computational thinking with an open-ended learning environment in a classroom setting. We defined our focal knowledge, skills, and abilities in CT

that are synthesized from the literature of CT pedagogy and assessment. Results of this case study showed the potential of our assessment framework in understanding students' learning of CT concepts and skills as the participants achieved significant learning gains in the CT KSAs defined in our assessment framework. This paper also demonstrated the benefits and feasibility of integrating CTSiM in everyday STEM learning contexts even for teachers with little experience with computer-based learning environments.

ACKNOWLEDGEMENTS

This work has been supported by NSF Cyberlearning Grant #1441542.

REFERENCES

- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community?. *ACM Inroads*, 2(1), 48-54.
- Basu, S., Biswas, G., Kinnebrew, J.S. (2017) Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Model. User-Adapt.* 27
- Basu, S. & Biswas, G. (2016). Providing adaptive scaffolds and measuring their effectiveness in open-ended learning environments. In *12th International Conference of the Learning Sciences* (pp. 554-561). Singapore.
- Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, 22(4), 564-599.
- Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn*.
- Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada (pp. 1-25).
- Chrysafiadi, K., & Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications*, 40(11), 4715-4729.
- Desmarais, M. C., & Baker, R. S. (2012). A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction*, 22(1-2), 9-38.
- García-Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., & Jormanainen, I. (2016). *An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers*. Belgium: TACCLE3 Consortium.
- Grover, S. (2015). "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. In *Proceedings of the 2015 Annual Meeting of the American Educational Research Association* (pp. 15-20).

- Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57-62). ACM. Chicago
- Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). Computational Thinking Patterns. *2011 Annual Meeting of the American Educational Research Association (AERA)*, 2, 1–15. <http://doi.org/10.1098/rsta.2008.0118>
- Land, S. M. (2000). Cognitive requirements for learning with open-ended learning environments. *Educational Technology Research and Development*, 48, 61–78.
- Mislevy, R. J., Almond, R. G., & Lukas, J. F. (2003). A brief introduction to evidence-centered design. *ETS Research Report Series*, 2003(1), 1–29.
- Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1), 75-79.
- National Research Council (U.S.). (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, D.C: National Academies Press.
- NGSS Lead States (2013) *Next generation science standards: for states, by states*. The National Academies Press, Washington, DC.
- Repenning, A., Ioannidou, A., & Zola, J. (2000). AgentSheets: End-user programmable simulations. *Journal of Artificial Societies and Social Simulation*, 3(3), 351-358.
- 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.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016a). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147.
- Weintrop, D., Holbert, N., Horn, M. S., & Wilensky, U. (2016b). Computational thinking in constructionist video games. *International Journal of Game-Based Learning*, 6(1), 1-17.
- Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35
- Wing, J.M. (2011) *Research Notebook: Computational Thinking—What and Why?*. Retrieved January 1, 2017 from <https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why>
- Wilensky, U. (1999) *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston. Retrieved January 1, 2017 from <http://ccl.northwestern.edu/netlogo>
- Wilensky, U., Brady, C., & Horn, M. (2014) Fostering computation literacy in science classrooms. *Commun ACM* 57(8):17–21
- Zhang, N., Biswas, G., & Dong, Y. (in press) Characterizing Students' Learning Behaviors Using Unsupervised Learning Metho

Computational Thinking in the Science Classroom

Hillary SWANSON, Gabriella ANTON, Connor BAIN, Michael HORN, Uri WILENSKY

Northwestern University

hillary.swanson@northwestern.edu, gabby.anton@gmail.com, ConnorBain2015@u.northwestern.edu, michael-horn@northwestern.edu, uri@northwestern.edu

ABSTRACT

The importance of Computational Thinking (CT) as a goal of science education is increasingly acknowledged. This study investigates the effect of computationally-enriched science curriculum on students' development of CT practices. Over the course of one school year, biology lessons featuring the exploration of NetLogo models were implemented in the classrooms of three 9th grade biology teachers at an urban public secondary school in the United States. One-hundred thirty-three biology students took both pre- and post-tests that were administered at the beginning and end of the school year. The students' responses to relevant assessment items were coded and scored using rubrics designed to evaluate their mastery of two learning objectives relating to modeling and simulation practices. The first learning objective was to explore the relationship between a system's parameters and its behavior. The second learning objective was to identify the simplifications made by a model. Each item's pre- and post-test scores were compared using a Wilcoxon signed-rank test. Results indicate a statistically significant improvement with respect to the second of the two learning objectives, suggesting that the computationally-enriched biology curriculum enhanced students' ability to identify the simplifications made by a model.

KEYWORDS

Computational Thinking, STEM Education, Learning Objectives, Curriculum, Assessment.

1. INTRODUCTION

The importance of Computational Thinking (CT) as a goal of science education is increasingly acknowledged (Quinn, Schweingruber, Keller, 2012; Wilensky, Brady & Horn, 2014). Teaching CT in the context of science not only presents students with a more authentic image of science as it is practiced today, it also increases access to powerful modes of thinking and marketable skills for many careers (Levy & Murnane, 2004). It is estimated that by 2020, one out of every two STEM jobs will be in computing (ACM Pathways Report 2013). However, students from groups that have been historically underrepresented in STEM fields (such as women and racial minorities) are less likely to enroll in computer science classes (Margolis, 2008; Margolis & Fisher, 2003) and thus are not traditionally exposed to CT practices. We believe we can improve access for all students, especially those underrepresented in CS, by embedding CT practices in subjects such as biology, chemistry, and physics, which all high school students are expected to take. While this does not ensure that these students will be personally motivated to engage in our CT curriculum, it ensures that

they will at least be exposed to CT practices and given the opportunity to learn about them.

For the reasons given above, we believe that developing CT practices in the context of science subjects is a productive endeavor. However, the character of CT practices in the science disciplines is not yet well understood, nor is how to create curriculum and assessments that develop and measure these practices (Grover & Pea, 2013). To address this gap, our group has worked to explicitly characterize core CT practices as specific learning objectives and used these to guide our development of science curriculum and assessment. We developed our learning objectives upon a theoretical taxonomy of CT in STEM that our group previously proposed (Weintrop et al., 2016). The taxonomy consists of four strands of CT practices: *Data Practices*, *Modeling and Simulation Practices*, *Computational Problem Solving Practices*, and *Systems Thinking Practices*. We translated elements from each strand of the taxonomy into learning objectives through a process involving interviews with computational scientists and feedback from high school science teachers.

The general aim of our larger research agenda is to address the question: "Can engaging in computationally-enriched science curriculum help students develop CT practices?" In the present study, we address a more focused version of this question and investigate whether engaging in three computationally-enriched biology units over the course of the school year helped participant students develop CT practices, specifically two practices within the *Modeling and Simulations* strand of our taxonomy. Below, we describe our study design and analytical approach, then present results from a comparison of students' scores for pre- and post-assessments. Our results provide support for our claim that computationally-enriched science curriculum can foster students' development of particular CT practices.

2. STUDY DESIGN

We investigated our research question by analyzing data from the fourth iteration of a design-based research cycle (Collins, Joseph, Bielaczyc, 2004). The implementation spanned the 2015-2016 school year and was tested in three 9th grade biology classrooms at our partner school. Students were given a CT practices pre-test at the beginning of the school year and a CT practices post-test at the end of the school year. Over the course of the school year they participated in three CT science units, each unit approximately four days long. We investigated the role of the CT science units in students' development of particular CT practices by looking for statistically significant gains in scores for particular items from pre- to post-test.

2.1. Participants

We partnered with a public secondary school (serving grades 7 – 12) in an economically depressed neighborhood in a large city in the Midwestern region of the United States. The school was selected on the basis of the willingness of its teachers and students to participate in our study. The size of the school was typical for an urban public secondary school, with approximately twelve hundred students enrolled. The majority of the students at the school are considered to be of racial minority within the United States (71.1% Black, 24.5% Hispanic, 1.6% Asian, .3% American Indian, .2% Pacific Islander, .9% Bi-Racial, 1.4% White), with sixty-two percent from low income households. The school is characterized as selective-enrollment, meaning that the student population is academically advanced and highly motivated. We addressed our research questions by analyzing a selection of the pre- and post-test responses given by participating 9th grade biology students. A total of 133 of these students, distributed across three biology teachers, took both tests. Due to time constraints, a number of these students did not complete the entire assessment. Ten students did not complete the assessment item measuring learning objective 1 and 24 did not complete the assessment item measuring learning objective 2; these students' responses were therefore not included in the analyzed datasets.

2.2. CT Science Lessons

The biology students participated in three computationally-enriched biology units over the course of the school year. Each unit took approximately four school days and emphasized the exploration and manipulation of computational models of scientific phenomena or concepts. The first unit was on predator-prey dynamics and ecosystem stability. For this unit, students explored population dynamics in a simulation of an ecosystem consisting of three organisms (grass, sheep, and wolves) (Wilensky, 1997b). Students investigated the population-level effects of parameters for individual organisms (such as initial population and reproduction rate) by running the simulation with different values for each organism. Through their exploration, the students learned about the complex population dynamics that emerge from the interactions between individual organisms. The second unit was on AIDS. For this unit, students explored a model that simulated the diffusion of the infectious disease through a population (Wilensky, 1997c). Students investigated the effects of parameters for individual interactions (such as the probability of individuals to form a couple, and the probability of the disease transfer between partners) on the rate of spread of the disease. The third unit was on genetics. For this unit students explored a model that allowed them to change mating rules in a population of fish. Students investigated how changing parameters such as life span and mating choice could bring about changes in the overall allele frequencies in a population of fish. All units were meant to help students develop expertise regarding learning objectives for *Modeling and Simulations Practices* by engaging in science content through the exploration of NetLogo (Wilensky, 1999) simulations. NetLogo simulations were

chosen because the agent-based modeling environments make complex systems phenomena (such as those featured in the biology lessons) more intuitively accessible (Wilensky, 2001). Additionally, the NetLogo user interface makes transparent the relationship between a model's code and the phenomenon it simulates. This makes NetLogo a powerful tool for scaffolding students' transition from consumers, to designers and builders of computational models. In order to help students develop a flexible set of CT practices, other CT-STEM units feature simulations built in modeling environments such as Molecular Workbench (Concord Consortium, 2010) and PhET (Perkins et al., 2006) and introduce students to a range of computational tools for data analysis and problem solving.

2.3. CT Assessments

The pre- and post-tests were designed to evaluate students' mastery of CT practices. In this report, we present results concerned with two particular learning objectives within our *Modeling and Simulations Practices* strand. The first learning objective falls under the sub-strand element *Using Computational Models* and states that a student should be able to "explore a model by changing parameters in the interface or code." This is a very basic skill but it plays an important role in students' (and scientists') abilities to learn about the relationship between particular parameters and system behavior at the macro-level. The second learning objective falls under the sub-strand element *Assessing Computational Models* and states that a student should be able to "identify the simplifications made by a model." This learning objective is important to students' epistemological development, as it relates to their understanding of a computational model as a tool that is both powerful and limited with regards to the construction of new knowledge.

Both pre- and post-tests required students to interact with computational simulations. For the pre-test, students interacted with a simulation (shown in Figure 1, below) that modeled climate change and showed the relationship between temperature and amount of CO₂ in the atmosphere (Tinker & Wilensky, 2007). For the post-test, students explored a simulation (shown in Figure 2, below) that modeled the relationship between the pressure of a gas and its volume and number of particles in a sealed environment (Wilensky, 1997a; 2005).

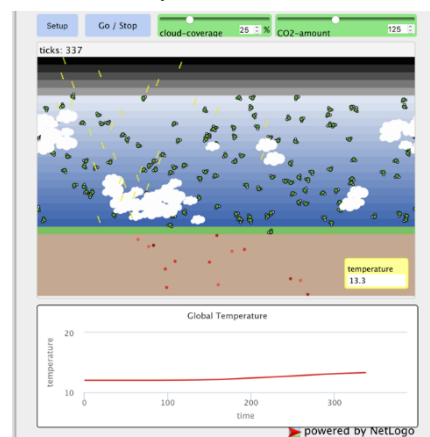


Figure 1. Screenshot of pre-test simulation modeling the relationship between temperature and atmospheric CO₂ levels.

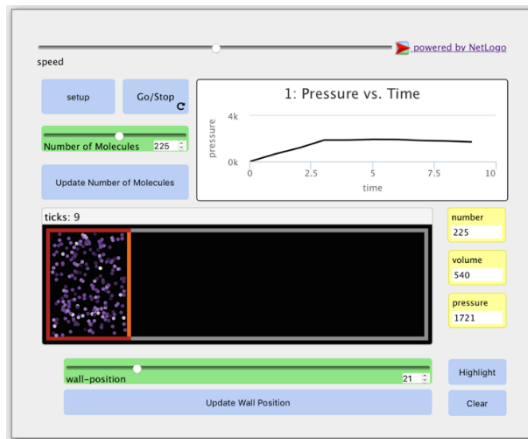


Figure 2. Screenshot of post-test simulation modeling the relationship between the pressure of a gas and its volume and number of particles.

To assess students' abilities to explore a model by changing parameters in the interface or code, we analyzed their responses to test items (quoted below) that asked them to attend to the relationships between adjustable parameters and system-level characteristics. In order to assess students' abilities to identify simplifications made by a model, we analyzed their responses to test items that asked them for the ways in which the simulations differed from the real-world. These assessment items were selected to investigate students' mastery of the same learning objectives across two very different computationally modeled phenomena.

2.4. Data Analysis

We used a combined top-down (learning objective driven) bottom-up (data driven) approach to create rubrics for evaluating students' responses to pre- and post-test questions and characterizing their mastery of both learning objectives.

2.4.1. Learning Objective 1

For the pre-test, in the context of the greenhouse gas simulation, students were asked to *explore the relationship between a system's parameters and its behavior* by changing a particular parameter and reporting on the resulting system-level behavior. In particular, they responded to the prompt: "Set cloud coverage to 0%. Take some time to experiment with different settings for the 'CO₂-amount' slider. What happens to the temperature if you increase the amount of the CO₂ in the model?" For the post-test, in the context of the gas-law simulation, students were asked to explore the relationship between a system's parameters and behavior by changing parameters to get a specific result. In particular, they responded to the question: "What values for container size and number of particles will result in the lowest pressure in the container? What steps did you take to come up with these values?"ⁱ

We examined students' pre- and post-test responses, sorting responses into categories based on similarities that

were relevant to our focal learning objective. Four categories emerged that characterized response types across both pre- and post-test responses. These categories are Noticing Parameter-System Relationships, Including Explanatory Factors, Comparing Across Trials, and Correctness.

These categories are outlined, described and illustrated with examples from the data in Table 1, below. We scored students' responses by awarding one point for each category included in their response and taking the sum of these points. This resulted in scores ranging from 0-3.

Table 1. Pre- and post-test rubric for analyzing students' responses and characterizing their ability to explore a model by changing parameters in the interface or code.

	Student Example
Relationships	
Response describes relationship between system parameters and macro-level patterns.	
<i>Pre-Test</i>	"The temperature increases."
<i>Post-Test</i>	"I slid the wall-position to its maximum and the number of particles to its minimum."
Explanatory Factors	
Response provides some explanation for relationship between system parameters and macro-level patterns.	
<i>Pre-Test</i>	"IR light does not get a chance to go into the sky because it is blocked by CO ₂ ."
<i>Post-Test</i>	"A bigger area and less particles shouldn't produce a large amount of pressure since it's a lot of space for the particles."
Comparison	
Response compares data across multiple simulation trials.	
<i>Pre-Test</i>	"When I increase the CO ₂ amount there seem to be IR light flying all over the place. But when there are smaller amounts of CO ₂ molecules the IR light have a better chance of going straight into the sky."
<i>Post-Test</i>	"To come up with these values I first tried putting the number of particles and the container size at its max. After that, I tried the number of particles at its minimum and the container size at its maximum."
Correctness	
Response correctly addresses the assessment prompt.	
<i>Pre-Test</i>	"The temperature increases."
<i>Post-Test</i>	"Number of particles: 25 Wall position: 96"

2.4.2. Learning Objective 2

As part of the pre-test, students were asked to *identify the simplifications* made by the greenhouse simulation. As part of the post-test, students were asked to identify the simplifications made by the gas-law simulation. For both tests, they responded to the question: “All computational simulations are only approximations of reality. What are some of the simplifications of this simulation that make it different from the real world?”

We examined students’ pre- and post-test responses, sorting responses into categories based on similarities that were relevant to the learning objective we were analyzing. Six categories emerged that characterized response types across both pre- and post-test responses. These categories are General Issues, Representational Issues, Controllability, Completeness, Procedural Limitations, and Off-Task. They are arranged in order of increasing sophisticationⁱⁱ, described and illustrated with examples from the data in Table 2, below. We scored students’ responses by awarding them the point-value of the highest category included. “Off-Task” (of point-value zero) was given to responses that did not address the assessment item, or consisted of “I don’t know.” Scores ranged from 0-3.

Two researchers analyzed students’ responses to the two assessment items for both pre-and post-tests. They coded responses (identifying the categories presented in the rubrics) and then scored them. The researchers’ inter-rater reliability for the pre-test was at 97% for the item measuring the first learning objective and 90% for the item measuring the second learning objective. Inter-rater reliability for the post-test was at 95% and 80%, respectively.

Table 2. Pre- and post-test rubric for analyzing students’ responses and characterizing their ability to identify simplifications made by a model.

	Student Example
General Issues – Score: 1 Response refers to general, as opposed to specific, inaccuracies or missing factors.	
<i>Pre-Test</i>	“In reality, other factors could come into play rather than just CO2 and clouds.”
<i>Post-Test</i>	“Inaccuracy in particles and wall position can make it different from the real world.”
Representation Issues – Score: 1 Response refers to representational limitations of the model.	
<i>Pre-Test</i>	“Obviously, sunlight is not a bunch of little sticks raining down.”
<i>Post-Test</i>	“It’s not actually life size.”
Controllability – Score: 2 Response refers to the existence of control over factors in the model that one does not have control over in real life.	

<i>Pre-Test</i>	“Because you can control how much CO2 and cloud coverage there is.”
<i>Post-Test</i>	“In real life, you cannot add or subtract molecules nor can you adjust the wall positioning.”
Completeness – Score: 2 Response refers to specific elements or factors that are missing from, or extraneous to, the model.	
<i>Pre-Test</i>	“There are humans on earth and humans also can add to the amount of heat.”
<i>Post-Test</i>	“The real world, does not have this many boundaries and an infinite number of particles.”
Procedural Limitations – Score: 3 Response refers to interactions, behaviors, or relationships within the model that differ from real life.	
<i>Pre-Test</i>	CO2 might not speed up that much when it absorbs IR light.
<i>Post-Test</i>	Particles don’t travel in and out of room in this simulation, when in real life they do.

To test whether the intervention played a role in their development of CT practices, students’ scores for each item on both pre- and post-tests were compared using a Wilcoxon signed-rank test. The findings of this analysis are reported below.

3. Findings

3.1. Learning Objective 1

Students’ average score for the pre-test item measuring their ability to explore a model by changing parameters in the interface or code was 2.03. Their average post-test score was 2.19. The p-value obtained using the Wilcoxon signed-rank test was 0.23 ($V = 1486$). The difference in student scores is therefore not statistically significant and we cannot make the claim that engagement in our curriculum helped students improve their CT skills with regard to this learning objective.

In addition to comparing students’ pre- and post-test scores for this learning objective, we compared the frequencies of categories of ideas that appeared in students’ pre- and post-test responses. Examination of the bar chart below reveals that during the pre-test, many students were concerned with macro-level effects of changing parameters, while at the time of the post-test, many more students referred to explanatory factors in their responses. This suggests they looked more closely at the model and tried to understand the interactions at the micro-level that explained the macro-level phenomenon. While the comparison of pre- and post-test scores indicates that students are not necessarily developing sophistication regarding their ability to explore a model, the changing frequency of categories gives us insight into

one specific way students may in fact be developing expertise.

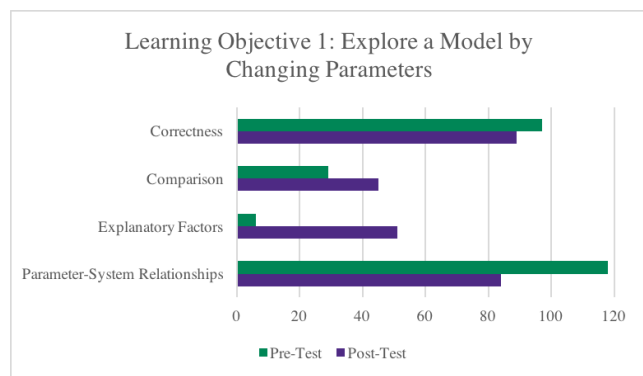


Figure 3. Frequencies of categories included in students' responses to the pre- and post-test items assessing their mastery of learning objective 1.

3.2. Learning Objective 2

Students' average score for the pre-test item measuring their ability to identify simplifications made by a model was 1.39. Their average post-test score was 1.63. The p-value obtained using the Wilcoxon signed-rank test was 0.02 ($V = 647.5$). The difference in student scores is therefore statistically significant (at the 5% significance level) and this supports our claim that engagement in our curriculum helped students improve their CT skills with regard to this learning objective.

In addition to comparing students' pre- and post-test scores for this learning objective, we compared the frequencies of categories of ideas that appeared in students' pre- and post-test responses. For ease of coding, we combined categories of the same score. This is reflected in the categories shown in the bar chart below. Examination of this bar chart reveals that during the pre-test, many students reported general or representational simplifications, whereas at the time of the post-test, this number decreased and the number of students reporting controllability or completeness as a limitation increased.ⁱⁱⁱ The number of students reporting procedural simplifications also increased. While the comparison of pre- and post-test scores indicates that students are developing sophistication regarding their ability to identify simplifications within a model, the changing frequency of categories gives us insight into the specific ways in which students are becoming more sophisticated.

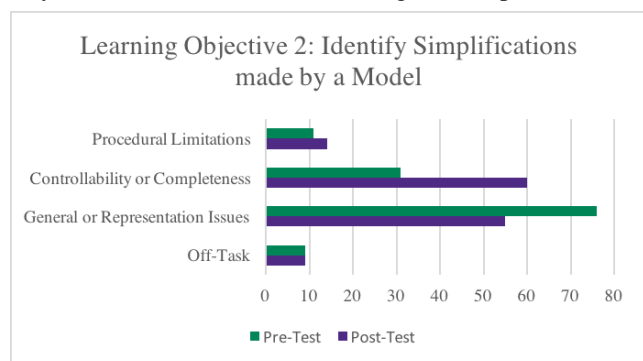


Figure 4. Frequencies of categories included in students' responses to the pre- and post-test items assessing their mastery of learning objective 2.

4. Discussion

This study extends our group's previous work by translating our theoretical taxonomy into learning objectives that can be used to guide the design of curriculum and assessment. The study makes an empirical contribution by presenting evidence that engagement in our CT-STEM curriculum helped participating students develop their ability to identify simplifications made by computational models. Our data also gives us insight into how students might develop their ability to explore a computational model. Toward this, we will conduct qualitative analysis of particular students and examine individual developmental trajectories. Our next steps also include refining our pre- and post- assessment items so that they are more closely aligned with each other, and with our learning objectives. As well, we are refining our curriculum (across the science subjects) so that it is more closely aligned with our learning objectives and assessment items. This refinement includes creating more opportunities for students to explicitly reflect on and discuss their individual ways of exploring models, as well as the simplifications they notice in different models.

5. References

- Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *The Journal of the learning sciences*, 13(1), 15-42.
- Concord Consortium. (2010). Molecular workbench. *Java simulations and modeling tools*, (2004–2013).
- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(1), 38–43.
- Levy, F. & Murnane, R. (2004). *The new division of labor: How computers are creating the new job market*. Princeton, NJ: Princeton University Press.
- Margolis J (2008) Stuck in the shallow end: education, race, and computing. The MIT Press, Cambridge
- Margolis J, Fisher A (2003) Unlocking the clubhouse: women in computing. The MIT Press, Cambridge
- Perkins, K., Adams, W., Dubson, M., Finkelstein, N., Reid, S., Wieman, C., & LeMaster, R. (2006). PhET: Interactive simulations for teaching and learning physics. *The Physics Teacher*, 44(1), 18-23.
- Quinn, H., Schweingruber, H., & Keller, T. (Eds.). (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.

- Tinker, R. & Wilensky, U. (2007). NetLogo Climate Change model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147.
- Wilensky, U. (1997a). NetLogo GasLab Gas in a Box model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. <http://ccl.northwestern.edu/netlogo/models/GasLabGasinaBox>.
- Wilensky, U. (1997b). NetLogo Wolf Sheep Predation model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. <http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation>.
- Wilensky, U. (1997c). NetLogo AIDS model. <http://ccl.northwestern.edu/netlogo/models/AIDS>. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
- Wilensky, U. (1999). NetLogo. Evanston, IL. *Center for Connected Learning and Computer-Based Modeling, Northwestern University*. <http://ccl.northwestern.edu/netlogo/>.
- Wilensky, U. (2001). Modeling nature's emergent patterns with multi-agent languages. In *Proceedings of EuroLogo* (pp. 1-6).
- Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM*, 57(8), 24-28.
- Wilensky, U., Novak, M. & Levy S.T. (2005). NetLogo Connected Chemistry 6 Volume and Pressure model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

ⁱ It is important to note that while both items are concerned with students' abilities to learn about a parameter's influence on a system's behavior, they are inversely structured. While the pre-test item instructs students to change a parameter and report its effect on the system, the post-test item instructs students to change parameters until they achieve a specified system behavior. We argue that while they are different in this way, both items are concerned with the causal relationship between parameter values and system-level behavior and are therefore comparable assessments of students' abilities to explore a model by changing parameters in the interface or code.

ⁱⁱ General comments about accuracy and representational limitations seemed to be the easiest to make with attention to mere surface-features. These simplifications were therefore awarded the lowest score (one point). The

completeness of the model and control given to its various parameters seemed to require more careful consideration of the interface and comparison with the real-world. These simplifications were therefore awarded a slightly higher score (two points). Finally, comments about the procedural correctness of behavior and interactions within the model required students to run the model and track cause and effect relationships between elements at the micro-level and comparison of this with scientific laws or theories. These simplifications were therefore awarded the highest score (three points).

ⁱⁱⁱ This point is especially interesting given that the gas-law simulation is just as unrealistic, regarding the visual representation of the system, as the greenhouse effect model.

Constructing Models in Physics: What Computational Thinking Occurs?

Sarah POLLACK^{1*}, Bruria HABERMAN², Orni MEERBAUM-SALANT³

^{1,3}Davidson Institute of Science Education, Weizmann Institute of Science, Israel

²Holon Institute of Technology, Holon, Israel, and Davidson Inst. of Science Education, Israel,

Sarah.Pollack@weizmann.ac.il, Bruria.Haberman@weizmann.ac.il, Orni.Meerbaum-salant@weizmann.ac.il

ABSTRACT

Computational thinking (CT) practices, especially abstraction and evaluation, are central to developing expertise in scientific disciplines, and considerable synergies exist between CT and scientific expertise. We present a pedagogical model based on the Equation-Based Model (EBM) for developing computerized simulations to describe physical phenomena. Specifically, EBM emphasizes the importance of mathematics as a central tool in science, and aims at fostering students' abstraction and evaluation practices, as part of their modeling processes.

We analyzed a final team-project of participants who decided to investigate a specific physical phenomenon in a course based on the EBM approach. Our analysis focused on characterizing the abstraction and evaluation practices, and the role they play in the scientific inquiry. The students applied multiple levels of abstraction, starting with the mathematic-system-level perspective of the conceptual model, and eventually constructed a computerized model of the conceptual model. They applied mathematical tools throughout the process, and verified and validated their models. The graphical simulation that the students built enabled them to investigate and enhance their comprehension of the problem explored. We concluded that this pedagogic approach has the potential to promote meaningful learning and knowledge transfer of computational thinking that were acquired during the course.

KEYWORDS

Computational thinking, scientific inquiry, equation-based model, abstraction, model evaluation.

1. INTRODUCTION

Computational thinking (CT) draws on concepts and practices that are fundamental to computer science and computing (Wing, 2006). Some of these practices are also central to developing expertise in scientific disciplines, and there are considerable synergies between CT and scientific expertise (Sengupta et al., 2013, Weintrop et al., 2016). Therefore, it is not surprising that recently, much effort has been invested in exploring the potential of CT to enhance model-based learning approaches using computing in STEM education. Indeed, using computing in model-based learning has been recently recognized as a suitable pedagogical means to engage students in scientific inquiry (National Research Council (U.S.), Pellegrino and Hilton, 2012).

"Scientific models are tools for expressing scientific theories in a form that can be directly manipulated,

allowing for description, prediction, and explanation." (Rapp & Sengupta, 2013, p. 2320). Scientific modeling is an iterative process, consisting of building, testing, and revision. More specifically, this process involves: (a) embodying key aspects of theory and data of phenomena into a model, (b) evaluating the model using the criteria of accuracy and consistency, (c) investigating the characteristics of the model in order to illustrate theoretical arguments about the mechanism or internal structure, and (d) interpreting the model and obtaining insights about the investigated phenomenon (Schwarz and White, 2005; Hughes, 1977). Furthermore, today computers serve as an important tool for creating and using scientific models. Thus, modeling requires students to develop, among others, the following interrelated key practices: abstraction and evaluation, on which this paper focuses.

Abstraction enables the problem-solver to handle complex data and to think in terms of conceptual ideas rather than merely in terms of their details (Wing, 2006). Therefore, Wing (2006, 2008) claimed that abstraction is a key practice in computing and that the abstraction process concerns making decisions as to what to emphasize and what to hide. This process, when successful, brings about a representation of the phenomena studied, that is, a generalized idea or an abstract structure, from which one can learn about a wide range of more concrete items with shared characteristics. Additionally, there are multiple levels and ways of abstracting. Therefore, mastering this practice involves the ability to understand the relationships between the different levels, transform from one level to another, and choose the most suitable form to represent the model. Abstraction also plays an important role in scientific inquiry, since scientific inquiry requires one to generalize a range of phenomena into one coherent conceptual model. Sengupta et al. (2013) investigated the degree of correspondence between abstractions in computational thinking and scientific inquiry. Modeling in scientific inquiry using computing involves two types of models: a conceptual model and a computerized model whose output consists of a simulation that enables one to study the behavior of the investigated physical system (Oberkampff, Trucano and Hirsch, 2004). Specifically, in physics, a conceptual model consists of a mathematical description of the physical phenomenon, and a computerized model that consists of implementation of the conceptual model in terms of programming a computerized system.

Because abstraction concerns constructing a conceptual presentation of the phenomenon, *evaluation* is necessary throughout every phase of the modeling process.

Therefore, the following question should be asked by the problem-solver: “How confidence in modeling and simulation should be critically assessed?” (Oberkampff, Trucano and Hirsch, 2004, p. 352). Accordingly, evaluation consists of the following dimensions: (a) Verification refers to determining whether the computerized model is an accurate implementation of the conceptual model and (b) Validation involves determining whether the computerized model accurately represents the real-world experimental measurements. This is achieved by using the simulation obtained from the computerized model. Accordingly, when evaluating their modeling artifact, students should carefully examine the simulation’s obtained output; they should justify the output logically and avoid intuitively relying merely on the similarity to the results of other experiments.

2. RATIONALE AND RESEARCH GOALS

Scientific modeling, and in particular, abstraction and evaluation, are not trivial practices. In fact, there is much empirical evidence on students’ difficulties when they are asked to employ these practices. One prominent example is the report by Schwarz and White (2005), according to which students’ understanding as to how to evaluate and revise a model in light of new data and insights remained limited, after they participated in an inquiry-oriented physics curriculum and engaged in the process of building computerized models.

Here we describe a pedagogical model aimed at fostering students’ abstraction and evaluation practices, as part of their modeling processes in physics; we also present the results of our investigation into students’ work. Our main objectives are as follows: (a) to identify and describe the abstraction and evaluation practices that were manifested in students’ physics modeling processes, (b) to understand how (if at all) these practices can enhance deep scientific inquiry, and (c) whether and how the pedagogical model can enhance or hinder these practices.

3. PEDAGOGICAL MODEL

We describe a unique program in computational physics aimed at introducing students to content knowledge and practices involving analyzing and solving physics problems by building computer simulations. Using an integrative approach, the program introduces, concepts, tools, and practices from physics, computer science, and applied mathematics (Landau, Paez and Bordeianu, 2011).

The program was implemented at The Davidson Institute of Science Education, the educational arm of the Weizmann Institute of Science, in Israel. Thirty high-school students (11th grade), who major in physics at school, attend 4-hour weekly meetings during which they study topics in physics, math (differential equations), and MATLAB programming.

The processes of scientific inquiry and the building of computerized models are demonstrated, with emphasis

on evaluation practices. More specifically, in addition to physics content, the pedagogical approach exposes students to the inquiry approach and practices that physics experts consider and apply when modelling physics phenomena. Special emphasis is on teaching content knowledge in the physics domain when relating to the knowledge of how, why, and when to apply this knowledge to answer questions and to solve problems (National Research Council (U.S.), Pellegrino and Hilton, 2012). We believe that this pedagogic approach may promote meaningful learning and knowledge transfer.

The course is based on the Equation-Based Model (EBM) in which modeling is first performed by describing the conceptual model of the system using a set of differential equations. EBM was chosen because it resembles a general systems-level approach to describe physical phenomena (Parunak et al., 1989). Uhden et al. (2012) referred to the role of mathematics in physics: “the role of mathematics in physics has multiple aspects: it serves as a tool (pragmatic perspective), it acts as a language (communicative function) and it provides a way of logical deductive reasoning (structural function).” (p. 486). Indeed, EBM emphasizes the importance of mathematics as a central tool in sciences and in physics, in particular. The participants in the course practice programming in MATLAB, which is a high-level language and is used in scientific and engineering computation, especially when dealing with differential equations, manipulating data and functions, and visual representation (Sen and Shaykhian, 2009). MATLAB was used for implementing the conceptual model as computer simulation.

The course is based on a learning-by-doing approach. Initially, participants are given a scientific paper that presents a physics problem and its computerized solution using the EBM approach. The students are requested to reconstruct the experiment described in the paper, and use it to evaluate the model, the experimental data, and the results described in the paper. In addition, they are requested to raise a new question and to inquire about it by using the model that they developed.

At the end of the course the students develop a final project. They choose and define a new problem and perform the whole process in pairs. While the process develops, they write a report in which they describe the conceptual and computerized models and the scientific inquiry processes that they encounter. They are asked to describe their considerations, assumptions, and to justify their actions. In the next section, we describe the analysis of one report out of 15 projects that students conducted at the end of the 2015 course. This particular work was chosen by the teacher of the course, who justified his choice, since this work reflects in general his students’ projects.

4. FINDINGS

We analyzed the final report of the team project of two students who decided to investigate the two-body problem in physics. In the analysis, we focused on

characterizing the *abstraction* and *evaluation* practices, as identified in the students' report, and related to their explanations and justifications.

4.1. Abstraction

Development and representation of the conceptual model: To define a system-level perspective of the problem that needs to be solved, the students began their investigation by using Newton's equation, which describes the magnitude of the gravity force that occurs between two objects in space.

$$F = \frac{m1 \times m2 \times G}{r^2}$$

Figure 1. Newton's gravitation laws

Next, the students used vector representation to describe the gravity force by relating to its direction as well:

$$\vec{F}_1 = \frac{m1 \times m2 \times G \times \vec{r}}{r^3} \quad \vec{F}_2 = -\frac{m1 \times m2 \times G \times \vec{r}}{r^3}$$

Figure 2. Vector representation

In the next step, the students decided to use Newton's second law with the previous vector equations to find new equations that enable one to find the location and speed of an object at any time in space.

Further steps led to new second-order differential equations, which, as the students explained, "connect between the position vector and the acceleration vector. This is possible because the gravitation force is the only force acting on the bodies."

$$\vec{r}_1''(t) = \frac{m2 \times G \times (\vec{r}_2(t) - \vec{r}_1(t))}{|\vec{r}_1(t) - \vec{r}_2(t)|^3} \quad \vec{r}_2''(t) = \frac{m1 \times G \times (\vec{r}_1(t) - \vec{r}_2(t))}{|\vec{r}_1(t) - \vec{r}_2(t)|^3}$$

Figure 3. Representation of differential equations

Finally, in order to enable the system to be tested for specific cases, the students defined a set of initial conditions. They justified this decision as follows: "It must be remembered that the solution of these equations will actually provide a set of functions, rather than one specific function. Hence, to find a specific function we must define a set of initial conditions; different initial conditions will lead to different functions."

Multiple levels of representation: Beyond the system-level perspective of the conceptual model in terms of Newton's laws, presented in the previous section (which we will refer to as "The First abstraction level"), the students described three additional levels of abstraction, needed to construct a computerized model of the conceptual model, and they explained the role of each level.

The second abstraction level was to represent the system in terms of a set first-order differential equations. The students explained that they need to transform the second order differential equations to the first order differential equations because they use Ordinary Differential

Equation (ODE) solvers in MATLAB (Sen & Shaykhan, 2009).

The third abstraction level is implementing the system in terms of a MATLAB code. In this stage a computerized model of the conceptual model is obtained.

The fourth abstraction level is graphically representing the objects' movement, obtained through simulation, which is actually an output of the computerized model. The students explained that the information, illustrated by the simulation, was obtained by solving the position and velocity functions: "We construct the simulation and use the information to create different graphs, to draw the objects' paths, and to present the dynamic occurrence and the objects' movement through time." This level actually enabled the students to perform inquiry using different case studies and enhanced their comprehension of the physical problem investigated.

4.2. Evaluation

The students wrote a 30-page report; half of it (15 pages) was dedicated to a chapter entitled "evaluation of the model and discussion of the results". The students stated: "In this part we will try to examine the simulation we have built. We will also compare the simulation to other known experimental data and we will use different tools and try to understand if the description and the results of the simulation are correct, reliable, and realistic."

Validating the model: Initially, the students used the computer simulation that they built to examine the conceptual model. To this end, they used the existing experimental data starting with Kepler's three laws of planetary motion. Owing to the limited scope of this paper, here we will describe only the evaluation of the results obtained from the simulation using Kepler's first law of planetary motion. Accordingly, the students tried to confirm that the simulation creates an elliptical path for all the planets, with the sun as one of ellipse's focal points. To test this, they determined that "the initial data that will comply with Kepler's laws... the mass of the sun is 10,000 kg and the other planet's mass is 1 kg." They created the simulation accordingly and observed the visual display (Figure 4) to determine whether the objects' paths indeed look like an ellipse.

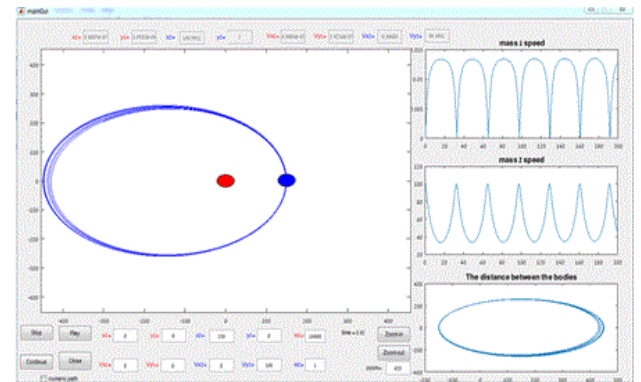


Figure 4. Simulation of Kepler's first law

Next, they set the goal of proving that what they see is indeed an ellipse using the mathematical definition of an ellipse. After they had proved it, they checked the path

obtained with different initial values. Finally, they also mathematically proved the second part of Kepler's first law, according to which "the sun is one of the focal points of the ellipse." Similarly, they examined the second and third laws of Kepler.

Verifying the model: When analyzing the simulation's results over time, they noted an error expressed in several paths: "After the first round, it can be seen that the graph becomes much less regular ... as time progresses." They explained the error as being a result of the numerical computation, claiming that "Most of the inaccuracy in the simulation is due to a lack of precision in solving this differential equation. Therefore, as time progresses, the path changes and diverges from reality."

As a result, aiming to improve the computerized model, they decided to develop an alternative algorithm for computing the objects' paths. They explained: "One way to check the correctness of the analytic algorithm [of the solution of differential equations] by using a computer is to compare it to a different algorithm." They chose a numerical algorithm to calculate a planet's path based on dividing it into fragments.

Next, the students ran the models (based on the two alternative algorithms) multiple times and examined the graphs of the object's movement, which were obtained. They concluded that "it shows that the orbits are very similar to each other, which means that the two algorithms are "close" and approximately describe the reality."

However, they noted some time-dependent differences between the paths obtained from the analytical algorithm and the numerical algorithm: "the longer the time since the beginning of the simulation, the differences (between paths) become bigger and they drift apart; this also occurs as the velocity speeds up."

Based on the simulation results, the students also noted that the deviations in the paths obtained by the numerical algorithm are smaller than those obtained by the analytic algorithm. Their analysis led them to the following conclusions:

(a) The first conclusion relates to the computing effect, in the context of proximity to reality: "the numerical solution remains closer to reality than does an analytical solution. This can be explained by the possible lack of precision of the computing. At lower speeds and less acceleration, the difference may be negligible and hardly noticeable; however, when dealing with high velocities, the velocity affects the position each time it becomes larger, and this might lead to different results."

(b) The second conclusion relates to cumulative errors resulting from the computing process: the students concluded that computing the analytic algorithm causes a cumulative error that significantly increases over time, compared with computing the numerical algorithm.

5. DISCUSSION AND CONCLUDING REMARKS

The analysis of students' reports revealed that the students used high-level abstraction and evaluation practices, which in turn, enhanced their scientific inquiry. The students created multiple abstraction levels, explained the assumptions they had made, and this helped them to deal with the system's complexity. They described mathematically a computational abstraction of the physics system governing the two-body problem, transformed this representation to code in the MATLAB environment, and created a graphical simulation to describe the dynamics of the physical system they chose to investigate. They also performed various actions in order to evaluate the conceptual and computerized models. They validated the conceptual model using experimental data (Kepler's laws of planetary motion) and verified the computerized model using different algorithms. Based on the above, we can infer that the students' performance, actions, and decisions resemble experts' scientific inquiry.

Moreover, we assume that the modeling used in the EBM approach will enable students to acquire CT practices and will promote understanding the synergy between CT and scientific thinking. In the EBM approach, the modeler first has to define the conceptual model of the system, usually using mathematics, as our study demonstrated.

Mathematical thinking has been perceived by the researchers as a tool that helps one to reason precisely and analytically about formally defined abstract structures and it "helps to move from [an] informal and complicated real world to a simplified abstract model" (Kramer, 2007, p. 41). Indeed, the students in our study used mathematical thinking to describe the physical phenomena, and transferred between multiple abstraction levels. We concluded that mathematical thinking was also important for students when they verified and validated the conceptual and computerized models and communicated their ideas.

In today's global and digital age, students need to master computational practices that will enable them to solve problems in different contexts and various domains. Thus, students should take advantage of deep learning opportunities and use transferable knowledge (Pellegrino and Hilton, 2013). Transferable knowledge involves the ability to use concepts and practices learned in one context, transfer them to another one, and apply their cognitive ability, which Salmon and Perkins (1989) termed as high-level transfer. More specifically, knowledge transfer can be defined as mindful action based on analytic analysis and reasoning about the connection between the two contexts, and then suitable ways can be found to use the knowledge in a new context.

We can conclude that the course described here indeed exposes students to knowledge regarding the use of abstraction and evaluation in scientific modeling. More specifically, students were constantly exposed to the

tools, strategies, considerations, and assumptions that scientists used in the modeling process. They also were requested to describe the artifacts they had built as well as explain and justify their actions during the development process. We believe that this pedagogical approach will contribute to students acquiring the cognitive knowledge and practices that are needed to perform high-level transfer of CT.

Finally, there are many studies that describe students' difficulties in using mathematical thinking in physics and science. However, educators should support and encourage excellent students, as we demonstrated in this study. More work is needed to examine the role of mathematics in enhancing high-level computational thinking, which may encourage students to engage in deep learning and transferrable knowledge.

6. ACKNOWLEDGMENT

We would like to thank Dr. Arik Ben-Haim, the Davidson Institute of Science Education, for his many helpful comments and insights.

7. REFERENCES

- Hughes, R. I. G. (1997). Models and Representation. *Philosophy of Science*, 64, S325–336.
- Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM*, 50(4), 36–42.
- Landau, R. H., Paez, J., & Bordeianu, C. C. (2011). *A survey of computational physics: introductory computational science*. Princeton University Press..
- National Research Council (U.S.), Pellegrino, J. W., & Hilton, M. L. (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century*. Washington, D.C: The National Academies Press.
- Oberkampff, W. L., Trucano, T. G., & Hirsch, C. (2004). Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57(5), 345-384.
- Parunak, H. V. D., Savit, R., & Riolo, R. L. (1998). Agent-based modeling vs. equation-based modeling: A case study and users' guide. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation* (pp. 10-25). Berlin, Heidelberg: Springer.
- Pellegrino, J. W., & Hilton, M. L. (Eds.). (2012). *Education for life and work: Developing transferable knowledge and skills for the 21st century. A report of the National Research Council*. Washington, DC: National Academies Press.
- Rapp, D.N., & Sengupta, P. (2012). Models and modeling in science learning. *Encyclopedia of the Sciences of Learning* (pp. 2320-2322). New York: Springer.
- Salomon, G., & Perkins, D. (1988). Teaching for transfer. *Educational Leadership*, 22-32.
- Schwarz, C., & White, B. (2005). Meta-modeling knowledge: Developing students' understanding of scientific modeling. *Cognition and Instruction*, 23(2), 165-205.
- Sen, S. K., & Shaykhian, G. A. (2009). MatLab tutorial for scientific and engineering computations: International Federation of Nonlinear Analysts (IFNA); 2008 World Congress of Nonlinear Analysts (WCNA). *Nonlinear Analysis: Theory, Methods & Applications*, 71(12), e1005-e1020.
- 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.
- Uhden, O., Karam, R., Pietrocola, M., & Pospiech, G. (2012). Modelling mathematical reasoning in physics education. *Science & Education*, 21(4), 485-506.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, physical, and engineering sciences*, 366(1881), 3717-3725.

Domain Specific Modeling Language Design to support Synergistic Learning of STEM and Computational Thinking

Asif HASAN^{1*}, Gautam BISWAS¹

¹ Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science, Vanderbilt University, 1025 16th Avenue South, Nashville, TN 37212, USA
d.asif.hasan@gmail.com, gautam.biswas@vanderbilt.ed

ABSTRACT

Computational Thinking involves core computer science concepts and practices that apply to multiple disciplines including science and mathematics. Currently, there is a strong drive toward integrating computer science into the K-12 STEM curricula. Several general-purpose programming environments have been developed to support the learning of CT and computing concepts and practices. Domain-specific modeling languages (DSMLs), on the other hand are designed for specific applications in engineering domains. As compared to general-purpose programming languages, DSMLs provide ease of use and more power to express domain-specific concepts, thus increasing productivity in specific application domains. In this paper, we present design guidelines and a design process for constructing DSMLs to facilitate STEM learning by computational modeling. To illustrate the process, we provide a case study of designing a DSML specifically for the kinematics domain.

KEYWORDS

Computational thinking, Domain specific modeling languages, Visual programming environments, STEM learning, Design guidelines

1. INTRODUCTION

Computational thinking involves crosscutting concepts and practices that apply to multiple disciplines including science and mathematics (National Research Council 2008). Wing (2006) introduced the term “Computational Thinking (CT)” stressing “*It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.*” But the idea of synergy between programming and science learning goes back decades, e.g., Papert’s (1980, 1991) pioneering work with Logo programming that showed procedural thinking development in children, and Perkins and Simmons (1988) research that showed the existence of similar patterns of novice misconceptions in math, science and programming. Other researchers have explored similar ideas to leverage the synergistic benefits of computational modeling and STEM learning (Sengupta, et al., 2013).

Several programming environments have been developed to support the learning of CT and computing concepts and practices (e.g. Alice (Pausch, et al., 1995), AgentSheets (Repenning, 2000), Scratch (Resnick, et al. 2009)) and for synergistic learning of CT and science (e.g. CTSiM (Basu, et al. 2013), CTSTEM (Jona, et al., 2014)), DeltaTick

(Wilkerson-Jerde, Wagh & Wilensky, 2015). These environments employ visual programming languages (VPLs) to facilitate program and model building, and graphical simulation output tightly integrated within the environment to demonstrate the results of executing the program and model structures. VPLs limit the chances of making syntactic errors allowing learners to focus more on the logic and execution flow of their programs, and to visualize the results of program execution.

When building complex scientific and mathematical models using general purpose VPLs, students may require significant support (Wilkerson-Jerde, Wagh & Wilensky, 2015). System designers may provide students with pre-implemented modeling constructs, to scaffold modeling tasks that are beyond the scope of what they need to learn. Providing students with a framework of such constructs may help them focus on tasks that are matched to concepts and processes they are expected to learn while the complex, and sometimes, unnecessary details of the implementation are kept hidden.

Such a framework can be systematically developed using *domain specific modeling languages* (DSMLs). DSMLs are frequently used in software design to systematize and facilitate the development of systems for specific application domains. The DSML concept is explained in Van Deursen, et al. (2000): “*A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.*” Characteristics of this approach are that they define constructs based on domain terminology for building models and applications, and specific constraints imposed by the domain can be incorporated into these constructs to avoid violations of domain principles.

In this paper, we describe a design process and design guidelines for constructing DSMLs in support of learning environments in science disciplines. To illustrate the process, we provide a case study of designing a DSML for the learning of Physics by building computational models, specifically in the domain of mechanics revolving around Newton’s laws of motion.

2. BACKGROUND

Visual environments for programming typically provide a set of block constructs to build computational artifacts. For example, Scratch (Resnick, et al. 2009), a widely used visual programming environment adopts a Lego-like framework for joining blocks to construct programs

(artifacts). Under the hood, these blocks are converted into textual code, which is executed, much like a traditional interpreted programming language. Snap! (Harvey & Mönig, 2010) expands Scratch's features. Though students have used these environments to build models that support STEM learning, their primary focus has been on learning programming and computing concepts, and by extension, the learning of CT concepts and practices (Maloney, et al. 2008, Brennan, et al. 2012, Werner, et al. 2012, Koh, et al. 2010).

AgentSheets (Repenning, 2000) is mainly targeted to learn CT by making games and science simulations. Alice (Alice (Pausch, et al., 1995), is another rich programming environment where students can build 3D virtual worlds. RoboBuilder (Weintrop, et al. 2012), FormulaT racing (Holbert et.al. 2010) and IPRO (Martin et al., 2013) are examples of game-based visual programming environments where students program agents or game-parts using DSMLs.

In contrast, systems like CTSiM (Basu, et al., 2013), DeltaTick (Wilkerson-Jerde, Wagh & Wilensky, 2015) and CTSTEM (Jona, et al., 2014), are visual computational modeling environments that are designed specifically to support synergistic learning of STEM and CT concepts. They differ from systems like Scratch and Snap! in that their building blocks are derived from DSMLs specifically designed for the target science domain that is the focus of student learning.

3. MOTIVATION

Although general-purpose programming environments provide the power and functionality to build models in STEM domains, basing the model building language on DSML constructs may help students to:

- Express solutions in the terminologies and at the level of abstraction of the target domain
- Build programs that are concise and self-documenting
- Enhance productivity
- Make it easier to reuse knowledge and procedures
- Make it easier to verify and validate models and results generated from the models
- Relate the constructed model with the actual phenomenon

4. DSML DESIGN GUIDELINES

In our work, we have adopted the following design guidelines/principles for DSML design for science learning environments. These guidelines are inspired by Van Deursen, et al., 2000 and Karsai, et al., 2014 among others.

Simplicity: The language constructs should be intuitive. Well-established notations from the domain should be used rather than inventing new ones. The constructs should be descriptive and distinguishable, yet compact.

Conciseness: Only relevant concepts in the domain should be targeted. Duplicate constructs that serve the same purpose should be avoided.

Separation of Concerns: If the target concepts can be separated into multiple non-overlapping sets, separate DSMLs may be designed for each which will enable each to grow independently and be more adaptable for future changes.

Consistency: All the constructs of the DSML should contribute to the purpose of the language.

5. THE DSML DESIGN PROCESS

In this paper, we target only task independent DSML constructs that can be used across tasks and possibly across different units in a specific domain (e.g., mechanics, electricity, fish tank ecology). In some situations, it may be desirable to scaffold students with task specific high level modeling constructs, but our focus in this paper is on the more generic modeling constructs (e.g., variables, laws) that support model building in a domain. Furthermore, assessment characteristics may also impact the design decisions of DSML, but in this paper, we consider the task independent DSML to be agnostic of the assessments we may develop in the learning environment. This does not preclude assessments being designed around specific DSML constructs. DSML design to support a science learning environment is likely to be iterative since it involves close interactions between the instructional design expert, the language developer, and domain experts. To simplify the language definition task, one may go through three step process for designing a particular DSML.

5.1. Define the learning and instructional goals in the domain

The target of this phase is to identify the learning and instructional goals, jointly by the domain, instructional, and system designers (one person may play multiple roles in this task). This will generally involve specifying domain concepts at the right levels of abstraction, and practices that the instructional tasks will be based on. The learning goals will also include CT concepts and practices, which will further influence how domain concepts are represented. A formal process, such as Evidence Centered Design (ECD) may provide a systematic approach for developing goals, tasks, practices, and constructs (Harris, et al. 2016).

5.2. Identify the scope of the computational modeling tasks

The learning goals and instructional tasks identified need to be translated into the scope of the modeling and problem solving tasks that the students will work on in the domain. In doing so, the types of tasks students will perform are to be identified. Therefore, it is essential to identify at the conceptual level, how students will perform each type of task, what computational constructs that they may use, what domain constructs they may be provided with and what kinds of relationships they would need to define among these constructs. The scope of the tasks will also identify features that need to be included in

programming and modeling environments. Sometimes the target environment may be pre-decided, and this impacts how students will perform the required tasks.

5.3. Link the designed DSML structures to their implementation in a specific environment.

Finally, design decisions are to be made on how the conceptual form of the domain constructs defined in the previous step are to be mapped to the implementational details. These decisions will be specific to the target modelling environment, but the designer must consider how each construct will be implemented in the modeling environment. For example, model building may employ a simple drag and drop interface, but to simulate the model, it will have to be converted into a form that runs in a separate programming environment (e.g., Netlogo, Simulink). Another example is to have the modeling constructs integrated into the target programming environment.

At this point, the design must consider the student's perspective (e.g. complexity of modeling, ease of use, system performance, etc.) and then refine accordingly. Often the need for new constraints or the need to modify the current constructs to meet specified learning goals may become apparent. These findings may result in backtracking to the first step of instructional design.

Some of the domain constructs may be mapped to library modules instead of language constructs. Providing libraries is an elegant approach to scaffolding, and making the learning process manageable.

6. Case Study: Developing a DSML in Kinematics

We will use Snap! (Harvey & Mönig, 2010) as our target modeling environment to develop and illustrate our case study of developing a DSML in kinematics. Snap! is an agent based visual programming environment, where each agent is represented by a Sprite. Snap supports creating and destroying sprites programmatically as well as manually. Each sprite can have its own set of variables (properties), functions and a script defining its behavior. There are options to create global variables and functions which may be shared between the sprites. Using Snap! as our implementation environment, we now describe the design of a DSML for 1- and 2-dimensional study of motion in Kinematics.

6.1. Define the learning goals and instructional tasks in the domain

As the first step, we identify the scope of the domain and then define the concepts and practices that matches the scope. Lastly, we will identify the instructional tasks.

6.1.1. Define the scope of the target domain

For this case study, we choose two-dimensional motion as our domain, and limits its scope to the kinematics concepts of position, velocity, acceleration, and time. We exclude circular motion from the scope of learning domain. The concept of gravity is simplified and represented as acceleration in a specific, i.e., negative y direction. We

further assume that all motion is relative to a fixed frame of reference. In the modeling environment, this is represented by an x -axis parallel to the bottom edge of the computer display with positive values corresponding to moves to the right. Similarly, the y -axis is orthogonal to the x -axis with a positive y implying moves upward on the y -axis.

6.1.2. Define the concepts & practices within the scope

The target concepts we want to cover in this case study are: (1) *position*, which is specified as a vector with two components x and y specified relative to the origin; (2) *displacement*, which is the difference between current position and a pre-specified origin; (3) *distance*, which is a scalar quantity implying how far away an object is from the origin; (4) *velocity* as the vector rate of change of position; (5) *speed*, which is the magnitude of the rate of change of position (velocity has two components: a speed (magnitude) and a direction which is defined with respect to the x and y axes; and (5) *acceleration*, which is the vector change in velocity.

There are many practices that may be targeted when designing a complete curriculum, but here we target the following: *develop a model representing the acceleration, speed, and position of a point object that is derived from the laws of kinematics, and use the model to solve problems or generate data to support explanations, predict phenomena, analyze systems, and/or solve problems.*

As part of the modeling tasks, we want students to use various computational constructs, such as variables, functions, control flow, conditional statements, and Boolean operators to model kinematics phenomena. We also want them to learn CT practices categorized as data practices, modeling & simulation practices, computational problem solving practices and systems thinking practices in Weintrop, et al., 2016.

6.1.3. Define the instructional tasks

For this case study, we assume that the students will work with a single physical object which starts at a specified position. We want them to go through the following tasks:

- If the object has a constant velocity, incrementally record the distance traveled over a period of time.
- If the object has a constant velocity, calculate the time required to travel a certain distance or to go to a certain position.
- If the object has a constant acceleration, calculate the velocity and position of the object over a specified time interval.
- Calculate the acceleration needed to reach a velocity in a specified period of time.
- Calculate the acceleration needed to travel given distance in a specified period of time.
- Calculate the acceleration needed to bring an object to zero velocity in a certain time or at a certain distance or position.

To accomplish these tasks, the students may use plotting functions provided by the system. This helps them learn and explain the targeted domain and CT practices.

6.2. Identify the scope of the computational modeling tasks

As part of this phase, we discuss how the various instructional tasks are to be mapped to computational modeling tasks. For this case study, the basic flow for all the modeling tasks would be as following:

- Students will be provided with a scenario with one or more objects. The objects may or may not be assigned an initial position and initial speed.
- Students have to specify what variables to associate with each of the objects, these variables are linked to physical quantities defined in the DSML.
- Students will build and execute computational models that described the motion of those objects. In other words, they will need to model the relationships between different physical properties such as how velocity impacts the position, acceleration impacts the velocity using Newton's laws as interpreted in kinematics. Their models have to be consistent in the way they specify scalars and vectors and their relations.
- Students will verify the correctness of their models by comparing the behaviors generated by their model against those produced by an expert simulation model. They will not have access to the expert model. The comparison of behaviors will be done by studying animations of the scenario modeled and plots of variable values across time. In some cases, students may not be provided with the results of an expert simulation. They may have to study the plots to determine the correctness of their models.
- Students may change parameters in their models and simulate them again to solve additional problems and answer questions. They can use the plots to justify their answers.

To accomplish such modeling tasks, students will also have access to general-purpose computational concepts such as constructs for updating variables, using conditional statements, Boolean operators, specifying functions, and imposing a control flow.

6.3. Link the designed DSML structures to their implementation in a specific environment

We will provide two versions of the DSML to illustrate the ease with which DSMLs may be scaled to include other tasks and constructs. For this case study, the DSMLs will be used to program the behaviors of Snap! Sprites. The first version of the possible DSML appears in Tables 1-3. Table 1 lists the variables, Table 2 the behaviors, and Table 3 lists the functional constructs.

The simulation of the physical objects (Sprites) in Snap!, can be thought of as a continuous representation process in which at every simulation step (multiple times in a second) it inquires the state of the model and graphically represents the state. To scaffold the simulation model building task for the students, the DSML provides a template behavior, where the students model how the state of the physical object should be updated for one simulation time step, and this repeats for a period of time that may be specified in the problem definition or by a variable set by the students. That template behavior is listed in Table 2 as "UpdateModel".

Table 1 lists all the variables. Basically, the variables students manipulate are a subset of variables defined in the DSML. Most variables represent physical quantities associated with objects, a restriction that the DSML imposes on the modeling environment to avoid physically meaningless models. The names of the variables are self-explanatory other than "DeltaTime". The variable "DeltaTime" records the period of elapsed time from the last simulation step to the current, which may be used to calculate for example, how much the physical object moves from a current position to the next position based on the current velocity and acceleration.

Table 3 lists all the functional constructs. The term `setPosition(X, Y)` sets the variable `PositionX` and `PositionY` to the values passed in as parameter `X` and parameter `Y` respectively. The construct `setDisplacement(X, Y)`, `setDistance(distance)`, `setVelocity(Vx, Vy)` and `setAcceleration(aX, aY)` acts accordingly. `ChangePosition`, `ChangeVelocity` and `ChangeAcceleration` updates their corresponding physical properties with respect to their current values and the values provided.

The rest of the functions support plotting capabilities. The term `plot(name, x, y)` can be used generally to plot a point on the graph. Assuming the target environment supports plotting on multiple graphs, each of the plotting functions takes in as parameter, the name of the graph to plot on. Rest of the plotting functions take in "Axis" as a parameter. The value of the parameter "Axis" can be "X" or "Y". Each of these functional term plots the value of the corresponding physical property on the axis provided and on the other axis automatically tracks the time elapsed from the start of the simulation. If no axis is provided it will use "Y" as the default value of the parameter "axis".

Table 1. DSML version 1 – Variables.

DeltaTime	Distance
PositionX	VelocityX
PositionY	VelocityY
DisplacementX	AccelerationX
DisplacementY	AccelerationY

Table 2. DSML version 1 – Behaviors.

UpdateModel()

Table 3. DSML version 1 – Functions.

setPosition(X, Y)	plotPositionX(name, axis)
setDisplacement(X, Y)	plotPositionY(name, axis)
setDistance(distance)	plotVelocityX(name, axis)
setVelocity(Vx, Vy)	plotVelocityY(name, axis)
setAcceleration(aX, aY)	plotAccelerationX(name, axis)
changePosition(dX, dY)	plotAccelerationY(name, axis)
changeVelocity(dX, dY)	plotDistance(name, axis)
changeAcceleration(dX, dY)	setLabel(name, xLabel, yLabel)
plot(name, x, y)	

To simplify design, the plotting and kinematics constructs are kept separate as shown in Tables 4 and 5. The variables and behaviors are omitted as they are same in both versions. The advantage of separating the DSMLs is that they can evolve independently, when more features or behaviors are added to one or the other. Another advantage is that a very high level of abstraction can be used for one, without affecting the other. For example, the plotting DSML could be abstracted just saying Plot(varX,varY), where the points X and Y are plotted at every simulation step. This provides students a visualization of the dynamic behavior without having to learn plotting functions. In other cases, details of the plotting functions, such as choosing axes, setting scales, and then plotting variables may be adopted.

Table 4. DSML version 2 – Kinematics.

setPosition(X, Y)	changePosition(dX, dY)
setDisplacement(X, Y)	changeVelocity(dX, dY)
setDistance(d)	changeAcceleration(dX, dY)
setVelocity(Vx, Vy)	
setAcceleration(aX, aY)	

Table 5. DSML version 2 – Plotting.

plot(name, xValue, yValue)
setLabel(name, xLabel, yLabel)

7. DISCUSSION

Visual programming languages emphasize the control flow of a program and reduce the syntactic burdens of programming in a conventional language, making computational modeling and problem solving more accessible to the students. In addition, DSMLs make the primary focus on representing specific domain modeling constructs, and how these constructs may be put together to simulate behaviors of the system. Whereas computational constructs are not the primary focus, they create a nice synergy between domain focus and computation focus in creating environments that students may employ to study domain principles, use them to build simulation models, and then study and justify the behaviors described by these models. Therefore, DSMLs provide a nice synergy in supporting both domain modeling and computational practices.

In additions, DSMLs can be specified at different levels of detail. The decision of ‘what abstraction level’ the DSML shall be designed at, is much depended upon the design of the curriculum. However, DSMLs also promote domain-general computations. For example, a DSML designed for

kinematics can be merged with a DSML for electricity, and used to model circuits, where charges move based on kinematics principles. A carefully designed DSML should be scalable and thus should support iterative evolution of the language. For example, here we did not provide any language construct for gravity. But, that can be added without any changes to any current construct and can be used with the existing functional constructs (e.g. changeAcceleration).

8. CONCLUSION

In this paper, our focus was on the design of DSMLs for learning Physics by computational modeling. In our related work, CTSiM (Basu, et al., 2017), we have adopted domain specific modeling constructs for students to model various science phenomenon e.g. mechanics with roller coaster, the fish tank ecosystem, for middle school students. In the future, we would like to run formal experiments to compare student’s performance of programming and model building with and without a DSML-based environment. Furthermore, we are currently exploring the domain of high school Physics to identify suitable set of DSMLs for high school students to develop systems that support synergistic learning of CT and STEM.

9. REFERENCES

- Basu, S., Dickes, A., Kinnebrew, J.S., Sengupta, P., & Biswas, G. (2013). CTSiM: A Computational Thinking En-vironment for Learning Science through Simulation and Modeling. In *Proceedings of the 5th International Confer-ence on Computer Supported Education* (pp. 369-378). Aachen, Germany.
- Basu, S., Biswas, G., Kinnebrew, J.S. (2017). Learner modeling for adaptive scaffolding in a Computational Thinking-based science learning environment. *User Modeling and User-Adapted Interaction*, 27(1) (pp. 5-53).
- Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (pp. 1-25).
- Harris, C. J., Krajcik, J. S., Pellegrino, J. W., & McElhaney, K. W. (2016). Constructing Assessment Tasks that Blend Disciplinary Core Ideas, Crosscutting Concepts, and Science Practices for Classroom Formative Applications.
- Harvey, B., & Mönig, J. (2010). Bringing “no ceiling” to scratch: Can one language serve kids and computer scientists. *Proc. Constructionism*.
- Holbert, N. R., & Wilensky, U. (2010, June). FormulaT racing: Combining gaming culture and intuitive sense of mechanism for video game design. In *Proceedings of the 9th International Conference of the Learning Sciences-Volume 2* (pp. 268-269). International Society of the Learning Sciences.
- Jona, K., Wilensky, U., Trouille, L., Horn, M. S., Orton, K., Weintrop, D., & Beheshti, E. (2014). Embedding

computational thinking in science, technology, engineering, and math (CT-STEM). In future directions in computer science education summit meeting, Orlando, FL.

Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., & Völkel, S. (2014). Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*.

Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010, September). Towards the automatic recognition of computational thinking for adaptive visual language learning. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on* (pp. 59-66). IEEE.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). *Programming by choice: urban youth learning programming with scratch* (Vol. 40, No. 1, pp. 367-371). ACM.

Martin, T., Berland, M., BenTon, T., & SMiTh, C. P. (2013). Learning programming with IPRO: The effects of a mobile, social programming environment. *Journal of Interactive Learning Research*, 24(3), 301-328.

National Research Council. (2008). *Taking science to school: Learning and teaching science in grades K-8*. Washington, DC: National Academy Press

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

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism*. (pp. 1-11). Nor-wood, NJ: Ablex.

Pausch, R., Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., & White, J. (1995) Alice: Rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, 15(3), 8-11.

Perkins, D. N., & Simmons, R. (1988). Patterns of misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research*, 58(3), 303-326.

Repenning, A. (2000). AgentSheets®: An interactive simulation environment with end-user programmable agents. *Interaction*.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, November 2009, 52(11), 60-67.

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.

Moenig, J., & Harvey, B. (2012). *BYOB Build your own blocks (a/k/a SNAP!)*, Retrieved Feb 14, 2017 <http://snap.berkeley.edu>.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6), 26-36.

Weintrop, D., & Wilensky, U. (2012). RoboBuilder: A program-to-play constructionist video game. In C. Kynigos, J. Clayson, & N. Yiannoutsou (Eds.), *Proceedings of the Constructionism 2012 Conference*, Athens, Greece.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 215-220). ACM.

Wilkerson-Jerde, M., Wagh, A., & Wilensky, U. (2015). Balancing curricular and pedagogical needs in computational construction kits: Lessons from the deltatick project. *Science Education*, 99(3), 465-499

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

The Role Gender Differences in Computational Thinking Confidence Levels Plays in STEM Applications

Nicole M HUTCHINS¹, Ningyu ZHANG², Gautam BISWAS³

^{1,2,3} Vanderbilt University, Nashville, Tennessee

nicole.m.hutchins@vanderbilt.edu, ningyu.zhang@vanderbilt.edu, gautam.biswas@vanderbilt.edu

ABSTRACT

Significant research has been done on gender disparities in STEM and computer science with the goal of broadening participation in these male-dominated fields. At the same time, the role of computational thinking (CT) as a tool to improve computer science skills along with STEM learning is becoming increasingly significant. This work seeks to add to this research through an analysis of the role confidence in computational thinking plays in developing STEM engagement and abilities. In the study reported in this paper, 40 high school students (21 girls and 19 boys) completed a Scratch project on modeling inelastic collisions in their Physics class. Pre- and post- surveys were conducted to analyze confidence levels in CT. Results showed a statistically significant difference in confidence levels in four CT dimensions: abstraction, flow of control, decomposition, and conditional logic. The results show that boys were more confident than girls in applying each of these dimensions. However, performance on the modeling assignment showed no statistical difference. We discuss the results and its applications to future work.

KEYWORDS

computational thinking, gender, STEM, motivation, curriculum

1. INTRODUCTION

Rapid advances in technology have created an environment in which computation is changing science and math research and practice. This has also had implications in pedagogy, where curricula are being reshaped to ensure students experience, understand, and learn to use computational tools in multiple disciplines. Combining computational modeling with STEM content has also been shown to synergistically deepen learning of the STEM topic and computing concepts (Sengupta et al., 2013, Wilensky, Brady & Horn, 2014). Furthermore, Wing's influential 2006 article, "Computational Thinking," has resulted in a growing number of studies that have sought to describe and analyze the role computational thinking (CT) encompasses in "solving problems, designing systems, and understanding human behavior by drawing on concepts fundamental to computer science"(p.33). CT impacts a number of educational disciplines – from science (Weintrop et al., 2016) to literature (Burke, Q., & Kafai, Y. B., 2012).

At the same time, there is growing awareness of the gender disparities in STEM. This has led to a number of studies on building female interest in predominantly male disciplines, such as computer science (Wang et al., 2015).

In environments where computation thinking (or CT) has been used as a tool for learning domains other than computer science, it is important to take into account disparities in attitudes on computer science and the specific STEM discipline as attitudes because differences in attitude can impact learning. While Van Braak (2004) concluded that girls felt less confident with computers than boys, to our knowledge, no work has looked into confidence levels in computational thinking and the influence the levels have on performance in a science or math activity.

The purpose of this study is to analyze the effect confidence levels in CT dimensions have on cross-disciplinary lessons that integrate computer science and physics. This study analyzed individual CT confidence ratings prior to the completion of an assignment utilizing Scratch to build a simulation model of a physics scenario. An analysis of students' abilities in applying CT concepts to build their models showed that girls' ability to finish their assignment correlated poorly with their overall confidence in applying CT concepts and practices. We discuss the results and their interpretation in subsequent sections.

2. THEORETICAL PERSPECTIVE

Studies have been conducted that show girls have low confidence in doing well in science topics (e.g., Kay, K. & Shipman, C., 2017). As previously mentioned, they also show low confidence in using computers (e.g., Van Braak, J.P., 2014). We briefly review CT concepts and practices, and then discuss prior work on girls' confidence levels in Computer Science subjects.

2.1. Computational Thinking

Our definition of CT is framed within two theoretical constructs: 1.) The Royal Society's definition of CT – "*Computational thinking is the process of recognizing aspects of computation in the world that surrounds us and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes*" (Wing, 2006; p.29). and 2.) Grover and Pea (2013) list elements that comprise CT including abstractions, systematic processing of information, symbol systems and representations, flow of control, decomposition, iterative, recursive, and parallel thinking, conditional logic, efficiency and performance constraints, and debugging (pgs. 39-40).

2.2. Girls' Confidence Levels in Computer Science

Significant research has been done to study girls' interests in computer science; most notably, the effect of computer science stereotypes on interest (Master, A., Cheryan, S., & Meltzoff, A., 2015). The authors described three predominant stereotypes that have been studied: (1) "male, technologically oriented, and socially awkward," (2) "a perception that it requires 'brilliance,'" and (3) computer science "is isolating and does not involve communal goals, such as helping or working with others" (2015). This study takes into account the stereotype that *computer science requires a specific level of intelligence*, and uses pre-post test questions as a means for assessing confidence levels.

In terms of building girls' interest in computer science, a study by Vekiri concluded that girls benefited from instructional practices that highlighted the relevance of information science to other disciplines (2013). This result seemingly supports further integration of CT related assignments into other STEM disciplines as it may positively influence girls' interest in the field of computer science.

3. Method

3.1. The Classroom

This study was implemented in a high school physics classroom in Nashville, Tennessee. The classroom, consisting of 40 students (21 girls and 19 boys), previously completed three physics assignments using Scratch: forces, one-dimensional motion, and projectile motion. Prior to the first assignment, the teacher introduced the students to the Scratch environment.

3.2. Pre-Survey

For the purpose of analyzing confidence levels in CT, we focused on four core dimensions of the CT framework: abstraction, flow of control, decomposition, and conditional logic. Students were given descriptions of each dimension, as shown in *Table 1*.

pre-test were exported to a Google Sheet for analysis.

Table 1. Computational Thinking Dimensions

Dimension	Description
Abstraction	Hiding all but relevant data about an object in order to reduce complexity and increase efficiency
Flow of Control	When designing an algorithm to solve a problem, computer scientists have the option of using control structures such as sequential structures, selection, or repetition
Decomposition	Breaking down a complex problem or program into parts that are easier to create, understand, design, and maintain
Conditional Logic	If an action or condition is true or false, it will result in a specific action

The pre-survey consisted of a Google Form in which all students were required to rate their confidence level in

applying each CT dimension to solve real-world problems. Confidence levels were determined using a five-point Likert scale. Students had access to the definition of the four CT terms while working on the survey. Answers to the

3.3. The Assignment

The learning objective of the collision project was to construct an inelastic collision simulation model using Scratch. Students set initial locations, direction and velocities for their chosen sprites (representing rigid objects) and then wrote code to model and visualize the collision between two sprites. Students chose sprites, such as spaceships, cars, or other relevant objects to visualize their collisions. Students were also asked to depict the mass of each sprite in the visual representation.

As previously mentioned, this physics class had previously completed three Scratch assignments. In each assignment, students were initially introduced to the physics topic in class through lectures, readings, or non-programming assignments. Then the students were given the relevant Scratch assignment. A primary component of each assignment involved each student's ability to translate concepts and laws in physics often represented by equations that they had learned in class into a computational model.

Final grades were determined by three factors: the student's ability to (1) include all variables specified in the assignment instructions, (2) build the simulation model using the block programming language (multiple flow of control structures were allowed and utilized) from the equations learned in class, and (3) provide an accurate visualization of the collision process. In other words, their task was to help others gain an intuitive understanding of inelastic collision processes from the visualizations they created.

The CT concepts and practices analyzed in this study were chosen based on their relevance to the completion of the assignment: abstraction (variable use), flow of control (process chosen for demonstration), decomposition (understanding relevance of each variable or in-class discussion topic on ability to accurately model), and conditional logic (what happens when the two sprites collide).

3.4. Post-Survey

Following the completion of the assignment, students were

asked to complete a CT post survey using Google Forms. This form included all CT definitions previously given and students were asked to provide an example of how they utilized each CT dimension in their collision assignment (if they thought it was applicable). Students were not graded on their ability to define the CT concepts and practices they used in their program on the post-survey. Rather, the examples on each CT dimension provided were used to relate their confidence levels to their understanding of the respective CT dimension.

4. Results

4.1. CT Confidence Levels

As shown in *Figures 1* and *2*, there were considerable differences between the initial confidence levels of the boys and girls on each of the CT dimensions. *Table 2* summarizes the quantitative results, and clearly indicates that the confidence levels for the boys were significantly higher ($p < 0.05$) than the girls on three CT dimensions (Flow of Control, Decomposition, and use of Conditional Logic). For the fourth dimension, $p < 0.1$, indicating a trend.

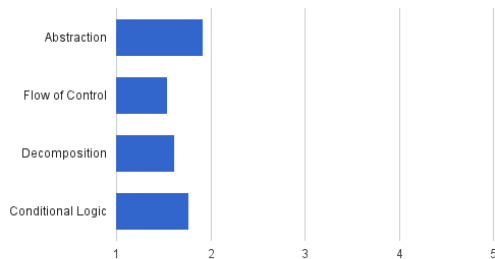


Figure 1. The Girls: Initial Confidence Levels in CT

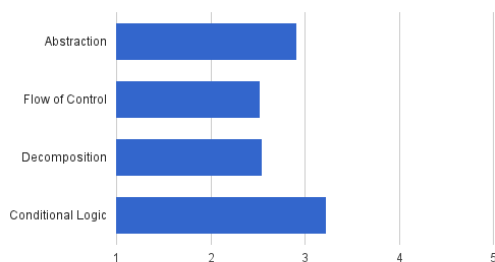


Figure 2. The Boys: Initial Confidence Levels in CT

Table 2. t-test Results on Confidence Levels.

CT Dimension	Girls	Boys	t-test Level of Significance
	Mean (SD)	Mean (SD)	
Abstraction	1.92 (1.12)	2.62 (1.45)	0.0921
Flow of Control	1.538 (0.877)	2.5 (1.508)	0.0306
Decomposition	1.61 (1.121)	2.538 (1.450)	0.0409
Conditional Logic	1.769 (1.166)	3 (1.581)	0.0166

4.2. Assignment Grading

Students were required to use the Scratch snipping tool to paste an image of their code to a Google Doc that was submitted to their teacher. In addition, students were required to submit a link to their projects in order for the teacher to evaluate the visual performance of the model.

Assignments were graded by the physics teacher based on the three project factors described in Section 3.1. *Table 3* shows the average grade in the class for girls and boys as well as the average grade on the collision assignment.

Table 3. Class and Assignment Averages.

	Class	Assignment
Girls	89.9	92.4
Boys	88.2	88.4

Upon further analysis, the difference between performances of girls and boys in terms of the class average and the individual assignment is insignificant. While confidence levels indicate that girls experience lower confidence in CT applications, there is no difference in abilities in both the model building task and in the class average.

4.3. Post-Survey: Qualitative Review

Students were not required to complete the post-survey due to time constraints; however, 18 students did complete the survey. *Table 4* showcases responses from two girl students and two boy students along with their respective confidence level in the CT dimension.

Table 4. Post-Survey Responses.

	CT*	CL*	Post-
G1	A	4	I only made new variables for things that I would use, rather than creating this based off of what I originally thought I might need.
	FC	4	When making the scratch (program) I used the sequential structure for putting together the commands.
	D	5	I broke it down by writing out the momentum formula in terms which I was then able to enter in.
	CL	5	This happened because of when one of the sprites hit the other, they would then move off the screen together. This was the end game.
G2	A	2	I originally started with a big equation to solve for final velocity, but that didn't work. I then decided to use only specific variables to solve for Vf, using a different approach and fewer total variables.
	FC	1	Repetition--I didn't think one of my sprite's masses would work, but I just put something in and it happened to work, so I continued my project that way while it was working. I decided that as long as the mass was working, I would keep going.
	D	1	Instead of using one big equation as I'd planned, I made several small equations to solve for one thing.
	CL	2	I decided that if the mass for one sprite worked, I could make the

			other work based on the mass of the one before.
B1	A	4	Writing out all necessary variables before figuring out what I needed to do with them.
	FC	3	
	D	3	Setting up loops.
	CL	4	Making if/then situations.
B2	A	1	No unnecessary variables
	FC	1	The placement of the change in the variable by which the sprite moved and the command for movement was purposeful, the command lines for the two sprites were identical barring individual variables, the use of a forever loop and an if/else clause
	D	1	Not much decomposition used as the design was very simplistic
	CL	2	If/else loop with the Boolean of contact used

*CT (Computational Thinking Dimension), CL (Confidence Level from Pre-Survey)

In the table above, it is important to note that both the girl and boy that submitted lower initial confidence levels in the CT dimensions were able to produce quality examples for each CT dimension used in their code for the post-test survey. Also of note is the length of responses by the girls versus the boys. We should also note that on the average, girls tended to submit longer examples of each CT dimension, with girls average 14.82 words per response and boys average 9.65 words per response.

5. DISCUSSION

The role of confidence has been shown to play a significant role in likelihood to pursue STEM careers (e.g., Moakler, M. & Kim, M., 2014) and the preliminary findings of this study correlate to the lower computer confidence levels experienced by female computer science majors compared to their male counterparts (e.g., Beyer, S. et al, 2003). These two studies were conducted with first year college students, but based on the findings of this study – confidence issues need to be addressed at an earlier age.

As previously noted, the goal of this study was to understand the effect of confidence levels on cross-discipline abilities. With this work, it can be seen that initial confidence levels are not a good indicator of content understanding (Physics and CT) and ability to apply the content to solving problems; however, the significant difference between initial confidence levels in CT needs to be addressed – particularly based on previous findings relating confidence levels to career choices. For instance, though no post-interviews were conducted to determine girls' confidence, it may be helpful to point out to them how well they perform with respect to the rest of the class, and they should be encouraged to become more engaged in STEM disciplines.

A unique component of this study is that confidence levels regarding computational thinking were assessed prior to

undertaking the physics activity. Previous studies have ranged from research on opinions following an educational activity (Atmatzidou, S., & Demetriadis, S., 2015) to an understanding of computer science perceptions in terms of motivation to take computer science courses (Vekiri, 2013). This study highlights the initial lack of confidence experienced by girls, and suggests further studies that delve into improving initial assignment scaffolding that can better address the initial confidence disparities.

Lastly, although there is no significant variation in performance in this classroom (the majority are high performers), the takeaway message may be that a weaker student, irrespective of gender, may need more scaffolding to aid them in their model building tasks.

5.1. Project Limitations and Future Implications

Concept assessment results indicate that the girls may benefit from the use of programming tools to simulate a physics concept. While this study conducted a confidence level survey prior to the completion of the assignment, an analysis of post-assignment confidence levels (via a Likert scale or similar) may provide additional insight into the effect the completion of a programming assignment has on not only confidence, but also interest in CT and computer science in general. We conjecture that building scenario models provide a better understanding of how STEM concepts may relate to real-world scenarios, and that may provide additional motivation for both girls and boys to pursue STEM disciplines. Indirectly, this may also help overcome the low confidence levels experienced by girls, thus increasing their engagement with STEM disciplines early in their education.

This study included a small cohort of students. Based on confidence level and performance results, future studies should be implemented that analyze the effect of CT confidence levels in a programming in physics application of a larger cohort to determine whether this trend holds. In addition, this study specifically assessed the role of CT confidence levels in a physics application. In order to determine if a broader STEM impact exists, studies should take into account multiple STEM disciplines to analyze if the effect remains the same on a broader scale.

However, keeping in mind the importance of introducing the relevancy of computer science in other disciplines, described in the Theoretical Perspective, initial instruction related to the computational tools needed to better understand a scientific concept can be seen as a beneficial approach to building confidence and interest in both STEM and computer science. Content performance of the girls indicated a significant ability to complete a STEM assignment using a computer science tool. Future work with this framework may contribute to an understanding of the synergy between STEM and computer science – a synergy that can impact future career directions.

Future applications of this approach would involve multiple components. As a means of improving the analysis of CT understanding in this physics assignment, CT content assessments could be developed to analyze each student's understanding of a relevant programming

tool separate from its usage in the physics model. Furthermore, research should be done on the translation of programming and CT knowledge developed using block-based programming languages into text-based programming abilities. For example, in this physics classroom students completed four Scratch assignments. A unit could be developed with the four completed assignments, as an increasing amount of CT and programming ability is needed for each new assignment. Following the completion of the unit, students could be introduced to an object oriented programming language and tested on their ability to program collision using the new language with a focus on whether students were able to capture CT dimensions, such as conditional logic, abstraction, etc., using the new language.

The results from this study indicated that while girls have a significantly lower confidence in CT applications, there is no difference in their ability to perform CT tasks when compared to their male counterparts. There are many initiatives currently working to bring computer science education to high school classrooms (Office of the Press Secretary, 2014). This study further supported the concept that the introduction of computer science and CT related content to already existing STEM curriculum can provide a resource for building girls' interest and confidence in computer science. As we move towards a broader availability of computer science education at the secondary school level, it is important to take into account CT confidence levels as a means of more effectively impacting a greater number of potential female computer scientist.

6. REFERENCES

- Beyer, S., Rynes, K., Perrault, J., Hay, K., & Haller, S. (2003). In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, (pp. 49-53). ACM.
- Burke, Q., & Kafai, Y. B. (2012, February). The Writers' Workshop for Youth Programmers: Digital Storytelling with Scratch in Middle School Classrooms. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 433-438). ACM.
- Grover, S. & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(1), pp. 38-43.
- Kay, K & Shipman, C. (2017). The confidence gap. Retrieved April 26, 2017 from <https://www.theatlantic.com/magazine/archive/2014/05/the-confidence-gap/359815/>
- Martin, M. & Kim, M. (2014). College Major Choice in STEM: Revisiting Confidence and Demographic Factors. *The Career Development Quarterly*, 62(2), 128-142.
- Master, A., Cheryan, S., & Meltzoff, A. (2015). Computing Whether She Belongs: Stereotypes Undermine Girls' Interest and Sense of Belonging in Computer Science. *Journal of Educational Psychology*, 108(3), pp. 424-437.
- Office of the Press Secretary. (2014). Fact sheet: New commitments to support computer science education. The White House. Retrieved February 4, 2017, from <https://obamawhitehouse.archives.gov/the-press-office/2014/12/08/fact-sheet-new-commitments-support-computer-science-education>
- Royal Society. (2012). Shut down or restart: The way forward for computing in UK schools. Retrieved February 4, 2017, from <https://royalsociety.org/~media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf>
- Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., & Clar, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, 18(2), pp. 351-380.
- Van Braak, J. P. (2004). Domains and determinants of university students' self-perceived computer competence. *Computers and Education*, 43, pp. 299-312.
- Vekiri, I. (2013). Information science instruction and changes in girls' and boys' expectancy and value beliefs: In *search of gender-equitable pedagogical practices*. *Computers & Education*, 64, pp. 104-115.
- Wang, J., Hong, H., Ravitz, J., & Ivory, M. (2015). Gender Differences in Factors Influencing Pursuit of Computer Science and Related Fields. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp.117-122). ACM.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, pp. 1-21.
- Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering Computational Literacy in Science Classrooms. *Commun. ACM*, 57(8), pp. 24-28.
- Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), pp. 33-36.

K-12 Computational Thinking Education in Germany

Nguyen-thinh LE, Niels PINKWART

Department of Informatics
Humboldt-Universität zu Berlin
nguyen-thinh.le@hu-berlin.de, niels.pinkwart@hu-berlin.de

ABSTRACT

How is computational thinking education in Germany? This paper aims to investigate computational thinking education in K12 German secondary schools. The methodology is based on analyzing the competence-based curricula frameworks for Computer Science in four highest populated federal states in Germany. In addition to Computer Science education, we also consider other subjects, e.g., Physics, because computational thinking may also exist in other subjects. Finally, we compare computational thinking education in Germany with international level by taking the ACM recommendation for Computer Science curriculum into account.

KEYWORDS

Computational thinking education, K12, Germany, ACM Computer Science curriculum.

1. INTRODUCTION

What is Computational Thinking? No clear definition for this concept has been developed yet. Yadav and colleagues (Yadav et al., 2017,) and Denning (2009) suggested that the notion of computational thinking had the root in Computer Science when Polya (1945) discussed about “algorithmic thinking” approaches in the paper “How to Solve It”. Recently, this notion of computational thinking is embedded in the work of Papert (1980, 1991), which focuses on the LOGO programming language and which aims at supporting student’s algorithmic thinking and problem solving abilities. Yadav and colleagues have summarized different components for computational thinking based on Wing (2008) and Barr and Stephenson (2011). According to Wing (2008), computational thinking consists of the process of choosing the right abstractions and automation of those abstractions. Based on this idea, nine core computational thinking concepts have been proposed by Barr and Stephenson (2011): **data collection (DC)**, **data analysis (DA)**, **data representation (DR)**, **problem decomposition (PD)**, **abstraction (AB)**, **algorithms and procedures (AP)**, **automation (AU)**, **parallelization (PA)**, and **simulation (SI)**.

Based on these nine components of computational thinking, we aim at investigating the computational thinking education in German schools (5th grade to 12th grade).

2. METHOD

Germany has 16 federal states and each state defines a framework of output-oriented requirements in terms of expected competences for each grade. School teachers are

required to specify curricula by themselves considering a state-specific curriculum framework.

We will analyze documents of requirement curriculum framework for different federal states. Since the four federal states (North Rhine-Westphalia, Baden-Württemberg, Bavaria, and Lower-Saxony) have the highest population and the population of these four states is higher than half of Germany’s population, we intend to analyze the curriculum frameworks in these states. Since these four states are based in the West Germany, we also take one state with highest population in East Germany (Saxony) into account. The analysis is based on the nine core components of computational thinking summarized by Yadav and colleagues (2017).

Since most components of computational thinking are related to Computer Science, we will investigate the state-specific curriculum frameworks for Computer Science. In addition, we also consider the curriculum framework in other subjects, e.g. Physics, in order to examine the coverage of computational thinking education.

We analyze whether any vocabulary in the description of expected competences or learning objectives in the curriculum frameworks match the term (or synonyms of the terms) of a specific component of computational thinking. In this case, we can conclude that this component is covered in the curriculum framework being investigated.

3. RESULTS

North Rhine-Westphalia has the highest population in Germany (17.9 Mio., Statista, 2015). The curriculum framework of this state is based on five competence areas ((1) argumentation, (2) modeling, (3) implementing, (4) representation and interpretation, (5) communication and cooperation) and five content fields ((1) data and structuring, (2) algorithms, (3) formal languages and automata, (4) computer systems, (5) computer science, human and society). Here, we use the curriculum framework for secondary education. The expected competences “student identify by analyzing problems objects, their attributes, their operations and their associations” (NW, 2014, pp. 28) and “students identify for a specific problem entities, attributes, and relationships and their cardinalities and represent them in an entity-relationship diagram” match the component **problem decomposition (PD)**. The competences „students model classes with attributes, methods and association relationships noted with cardinality“ and “students apply the concept of polymorphy to appropriate problems” (NW, 2014, pp. 28) are specific to object-oriented programming paradigm. However, this competence may

be considered a part of the component **abstraction (AB)** of computational thinking. Similarly, the expected competences “students modify a database model” and „students model a relational database schema for an entity-relationship diagram“ (NW, 2014, pp. 34) are related to database modeling, however, can be considered a part of **abstraction (AB)**. In the content field “Algorithms”, the competences “students analyze, explain, and modify algorithms and programs”, “students develop iterative and recursive algorithms applying the strategies *Modularization, Divide and Conquer, and Backtracking*”, and “students evaluate the efficiency of algorithms with respect to memory usage and the number of operations” (NW, 2014, pp. 29, pp. 34) cover the component **algorithms and procedures (AP)** of computational thinking. In the content field “formal languages and automata”, the competences “students analyze and explain the attributes of finite automata/ push-down automata and their behavior for a specific input”, “students develop a formal language, which is accepted by a finite automaton or a push-down automaton”, and “students develop and modify finite automata or push-down automata for a problem” (NW, 2014, pp. 30, pp. 36) can be considered a part of the component **automation (AU)** of computational thinking. Also in the same content field “Algorithms”, “students explain the principle of concurrency” (NW, 2014, pp. 35) may partly match the component **parallelization (PA)** of computational thinking.

The curriculum framework in Physics in North Rhine-Westphalia seems to complement the curriculum framework in Computer Science with respect to computational thinking.

The curriculum framework for Physics addresses the components **data collection (DC)** and **data analysis (DA)** of computational thinking that we do not find in the curriculum framework for Computer Science. The following expected competences are specified in the curriculum framework for Physics in North Rhine-Westphalia address the component **data collection (DC)**: “Students have the ability to search, to analyze, and to evaluate by comparison relevant information and data in different sources as well as in selected scientific publications for physical questions.” (NW-Ph, 2014, pp.28); “Student have the ability to observe and measure criterion-driven, and explain and use complex devices for observations and measurements appropriately.” (NW-Ph, 2014, p.28). In addition to competences addressing **data collection (DC)**, several other competences emphasize data analysis: “Students have the ability to analyze data qualitatively and quantitatively with regards to coherences, rules or mathematical axioms” (NW-Ph, 2014, p.28); “Student have the ability to develop models, and explain and predict physical-technical processes using theoretical models, mathematical modeling techniques, thinking experiments and simulations.” (NW-Ph, 2014, p.62).

Table 1. Coverage of computational thinking in German schools.

	DC	DA	DR	PD	AB	AP	AU	PA	SI
Computer Science									
North Rhine-Westphalia				x	x	x	x	x	
Baden-Württemberg				x	x	x			
Bavaria			x	x	x	x	x	x	
Lower-Saxony						x	x		x
Saxony	x	x	x	x	x	x	x		x
ACM CS			x	x	x	x			x
Physics									
North Rhine-Westphalia	x	x							
Baden-Württemberg	x	x							
Bavaria	x	x							
Lower-Saxony	x	x							
Saxony	x	x							

Baden-Württemberg starts Computer Science education from the 11th grade to 12th grade. From the 6th grade to the 10th grade, school students in Baden-Württemberg are offered the so-called “information-technical basic education” courses, which serve as the basis for Computer Science education. The framework of requirements comprise five areas: (1) Information and data, (2) algorithms and data, (3) problem solving and modeling, (4) work principles of computer systems, and (5) informatics and society. Since in the 2nd area, the framework specifies three competences: “Students have the ability to apply basic datatypes and data structures”, “students have the ability to develop algorithms and implement them in programs”, and “students have the ability to apply modularization techniques” (translated from German, BW, 2004, pp. 439). These requirements of expected competences cover the following components of computational thinking: **data representation (DR)**, **problem decomposition (PD)**, **abstraction (AB)**, **algorithms and procedures (AP)**. In the area of problem solving and modeling, two competences cover the components **problem decomposition (PD)**, **abstraction (AB)**, and **algorithms and procedures (AP)** of computational thinking: “Students know basic principles of problem solving”, “Students can decompose the problem solving process”, “Students have the ability to map real problems into objects and classes” (BW, 2004, pp. 440).

Similar to the curriculum framework for Physics in North Rhine-Westphalia, Physics education in Baden-Württemberg addresses the two components **data collection (DC)** and **data analysis (DA)** that complement to Computer Science education with respect to computational thinking education: “Students have the ability to observe and describe phenomena and

experiments goal-oriented.”; “students have the ability to collect measurement data digitally and assess them.”; “students have the ability to conduct experiments, collect and assess data” (BW-Ph, 2016, pp. 10); “Students have the ability to evaluate results of experiments” (BW-Ph, 2016, pp. 12).

The present curriculum framework for Computer Science in Bavaria is not based on output-oriented competences yet, rather it is based on learning contents that need to input into school curricula in Bavaria. The competence-based requirement framework for schools in Bavaria has been developed and is planned to be applied from the school year 2017/2018¹. Since the competence-based requirement framework is available on the website of Bavaria’s State Institute for School Quality and Education Research (BA, 2017). According to this framework, Computer Science education starts from the 9th grade. After the 9th grade, the following competence is expected from the student: “Students analyze and decompose data of simple real authentic examples (e.g., inventory or client administration) and represent the developed data model graphically”

(<http://www.lehrplanplus.bayern.de/fachlehrplan/gymnasium/9/informatik>). This competence covers the components **data representation (DR)**, **problem decomposition (PD)**, **abstraction (AB)** of computational thinking. After the 10th grade, the following is expected: “Students represent algorithms in pseudocode or graphically for a given process-oriented problem using control structures.”

(<http://www.lehrplanplus.bayern.de/fachlehrplan/gymnasium/10/informatik>). This competence requires learning in **algorithms and procedures (AP)**.

After the 12th grade, “students have the ability to design finite automata using formal languages.” (<http://www.lehrplanplus.bayern.de/fachlehrplan/gymnasium/12/informatik>). This competence is in accordance with **automation (AU)**. The competence “Students model typical concurrent scenarios using Petrinets.” may be achieved by the component **parallelization (PA)** of computational thinking education.

Complementary to the curriculum framework for Computer Science, the curriculum framework for Physics in Bavaria address the components **data collection (DC)** and **data analysis (DA)**: “Students are in a position to infer physical knowledge from course texts, to search information and to work up results in documentation and presentation appropriately.” (BA-Ph, 2017) and “students reflect impacts of physical insights in historical and societal relations and are aware of chances and limits of physical solutions.” (BA-Ph, 2017).

The curriculum framework of Lower-Saxony is competence-based. This document distinguishes between process-oriented and content-oriented competences. Lower-Saxony offers Computer Science education from the 5th grade. At this moment, only the curriculum framework in Computer Science for 5th grade to 10th grade

is available on the Internet (NI, 2014). The specified competences are summarized in four learning areas: (1) data and their traces, (2) computer competence, (3) algorithmic problem solving, and (4) automated processes. In the learning field “algorithmic problem solving”, the competences “students describe a given algorithm in their own words”, “students represent an algorithm graphically”, “students execute a given algorithm”, and “students develop an algorithm using elementary control structures” (NI, 2014, pp. 20-21) cover the component **algorithms and procedures (AP)**. In the learning area “automated processes”, the competences “students describe automata as a composition of their states and transitions” and “students develop and implement an automaton model in form of a state graph” (NI, 2014, pp. 20-22) match the component **automation (AU)** of computational thinking. The competence “students model and simulate a given automaton using an appropriate simulation software” may be considered a part of the component **simulation (SI)** of computational thinking. The curriculum framework for 5th-10th grade schools in Lower-Saxony supports few components (AP, AU, and SI) of computational thinking. We hope that the curriculum framework for secondary schools in Lower-Saxony support more other components of computational thinking.

Considering the curriculum framework for Physics in Lower-Saxony, the expected competences specified in this framework address the components **data collection (DC)** and **data analysis (DA)** of computational thinking: “Students plan simple experiments, carry out them and document experiments’ results.”, “Students evaluate data using appropriate diagrams and identify functional relations” (NI-Ph, 2009, p.14), “Students add missing information by themselves.” (NI-Ph, 2009, p.22), “Students use for documentation and evaluation of measurement data GTR/CAS or table calculation” (NI-Ph, 2009, p. 23), “students evaluate and justify a result of an observation of measurement’s uncertainty.” (NI-Ph, 2009, p. 27). These competences indicate a complementary part to Computer Science curriculum in Lower-Saxony with respect to computational thinking education.

Saxony has the highest population (4.0 Mio., Statista, 2015) among the five federal states in East Germany. Page 2 of the curriculum framework (https://www.schule.sachsen.de/lpdb/web/downloads/lp_gy_informatik_2011.pdf?v2) summarizes the goals of Computer Science education. This summary of goals includes the components **data collection (DC)**, **data analysis (DA)**, **data representation (DR)**, **problem decomposition (PD)**, **abstraction (AB)**, **algorithms and procedures (AP)**. On the contrary to the four states in West Germany, where DC and DA are not addressed in Computer Science education, the federal state Saxony does. The specification of the learning area “Theoretical Informatics” addresses **automation (AU)** (SA, 2011, pp. 15). The learning area “Applied Informatics” covers the

¹ <http://www.isb.bayern.de/schulartuebergreifendes/paedagogik-didaktik-methodik/kompetenzorientierung>

component **simulation (SI)** (SA, 2011, pp. 17). Saxony's curriculum framework addresses almost all components of computational thinking except **parallelization (PA)**.

In the subject Physics, the curriculum framework in Saxony addresses electric parallel circuits (SA-Ph, 2011, pp. 15), which are not in the context of **parallelization (PA)** (Barr & Stephenson, 2011). Similar to other Physics curriculum frameworks, in the state Saxony, the components **data collection (DC)**, **data analysis (DA)** are supported: "Students learn to acquire, to categorize, and to use information in order to extend, to structure, and to apply their knowledge. Acquisition, usage, evaluation and presentation of information is important." (SA-Ph, 2011, pp. VIII).

4. ACM CURRICULUM FOR K-12 CSE

The ACM model curriculum divides Computer Science education (CSE) into four levels: Level 1 - Foundations of Computer Science, Level 2: Computer Science in the Modern World, Level 3 - Computer Science as Analysis and Design, and Level 4 - Topics in Computer Science.

On level 1 (recommended for grade K-8), students are expected to "apply strategies for identifying and solving routine hardware and software problems that occur during everyday use." (ACM, 2003, pp. 13), "understand the graph as a tool for representing problem states and solutions to complex problems" (ACM, 2003, pp. 14), and "understand the fundamental ideas of logic and its usefulness for solving real-world problems" (ACM, 2003, pp. 14). These competences meet the components **problem decomposition (PD)**, and **algorithms and procedures (AP)** of computational thinking. In addition, the competence "Use content-specific tools, software, and simulations (e.g., environmental probes, graphing calculators, exploratory environments, Web tools) to support learning and research" (ACM, 2003, pp. 13) addresses the component **simulation (SI)** of computational thinking. On level 2 (recommended for grade 9 or 10), students should have conceptual understanding of "the basic steps in algorithmic problem-solving (problem statement and exploration, examination of sample instances, design, program coding, testing and verification)", which meets again the components **problem decomposition (PD)**, and **algorithms and procedures (AP)** of computational thinking. On level 3 (recommended for grade 10 or 11), students should gain understanding of "fundamental ideas about the process of program design and problem solving, including style, abstraction, and initial discussions of correctness and efficiency as part of the software design process." (ACM, 2003, pp. 14) and "simple data structures and their uses" (ACM, 2003, pp. 14) address the components **data representation (DR)** and **abstraction (AB)** (in addition to other components mentioned above). On level 4 (recommended for grade 11 or 12), students attend the courses that deepen gained knowledge, abilities and skills in Computer Science. Students have the choice between an Advanced Placement (AP) Computer Science course that "emphasizes problem solving and algorithm development, and introduces elementary data structures" (ACM, 2003, pp. 18) or a project-based course, or a

vendor-supplied course. Especially, in addition to gaining knowledge, abilities and skills in Computer Science, ACM model curriculum promotes "the connection between elements of mathematics and computer science, including binary numbers, logic, sets, and functions." (level 2, ACM, 2003, pp. 15) and "topics in discrete mathematics: logic, functions, sets, and their relation to computer science". (level 3, ACM, 2003, pp. 14). These topics are considered required important in Computer Science education and for problem solving.

5. CONCLUSIONS

Table 1 shows that the federal states in Germany, that have in total more than half population of Germany, cover several components of computational thinking. The coverage of computational thinking components is heterogeneous among different federal states in Germany.

It is worth to note that **data collection (DC)**, **data analysis (DA)** are not considered in the four investigated curriculum frameworks for Computer Science in West Germany. However, these components are addressed in the state Saxony in East Germany. Considering the four states in West Germany, taking the curriculum frameworks for Physics into account, we can notice that both curriculum frameworks in Computer Science and Physics are complementary with respect to computational thinking education, since they cover most components of computational thinking (except the component **simulation SI**).

Surprisingly, the state Saxony in East Germany addresses almost all components of computational thinking except **parallelization (PA)**.

Comparing the Computer Science education in Germany and ACM model curriculum for K-12 Computer Science with respect to computational thinking education, no difference can be noted: the components **problem decomposition (PD)**, **abstraction (AB)**, **algorithms and procedures (AP)** are recommended in curriculum frameworks for Computer Science in Germany and in ACM model curriculum.

Based on the analysis results in this paper, we would recommend educators to pay more attention to the components **parallelization (PA)** and **simulation (SI)**, which are not considered in Computer Science curriculum frameworks in three of five federal states in Germany. In addition, since big data is increasingly a problem in computation, we would also recommend to embed methods of **data collection (DC)** and **data analysis (DA)** in Computer Science curricula, because these components could only be found in Physics curriculum frameworks.

6. REFERENCES

- ACM, (2003). A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee, ACM.
http://www.acm.org/education/education/curric_vols/k12final1022.pdf
- BA, (2017). LehrplanPLUS, Informatik, Bayern.
<http://www.lehrplanplus.bayern.de>

- BA-Ph, (2017). LehrplanPLUS, Physik, Bayern.
<http://www.lehrplanplus.bayern.de/fachprofil/gymnasium/physik/11>
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2 (1), 48–54. doi: 10.1145/1929887.1929905 .
- BW, (2004). Bildungsstandard Informatik, Gymnasium - Kursstufe, Baden-Württemberg, http://www.bildungsstaerkt-menschen.de/service/downloads/Bildungsstandards/Gym/Gym_Inf_wb_bs.pdf
- BW-Ph, (2016). Bildungsplan des Gymnasiums. Ministerium für Kultus, Jugend und Sport, Baden-Württemberg http://www.bildungsplaene-bw.de/site/bildungsplan/get/documents/lsbw/export-pdf/depot-pdf/ALLG/BP2016BW_ALLG_GYM_PH.pdf
- Denning, P. J. (2009). The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52 (6), 28–30. doi: 10.1145/1516046.1516054 .
- NI, (2014). Kerncurriculum für die Schulformen des Sekundarbereichs I, Schuljahrgänge 5 – 10, Informatik. Herausgeber: Niedersächsisches Kultusministerium
http://db2.nibis.de/1db/cuvo/datei/kc_informatik_sek_i.pdf
- NI-Ph, (2009). Kerncurriculum für das Gymnasium – gymnasiale Oberstufe die Gesamtschule – gymnasiale Oberstufe das Fachgymnasium das Abendgymnasium das Kolleg, Physik. Herausgeber: Niedersächsisches Kultusministerium
http://db2.nibis.de/1db/cuvo/datei/kc_physik_go_i_2009.pdf
- NW, (2014). Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen
http://www.schulentwicklung.nrw.de/lehrplaene/uplod/klp_SII/if/KLP_GOST_Informatik.pdf
- NW-Ph, (2014). Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen – Physik.
http://www.schulentwicklung.nrw.de/lehrplaene/uplod/klp_SII/ph/KLP_GOST_Physik.pdf
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 1–11). Norwood: Ablex.
- Polya, G. (1945). *How to solve it; A new aspect of mathematical method*. Princeton: Princeton University Press.
- SA, (2011). Lehrplan Gymnasium Informatik, Sächsisches Staatsministerium für Kultus und Sport
https://www.schule.sachsen.de/lpdb/web/downloads/lp_gy_informatik_2011.pdf?v2
- SA-Ph, (2011). Lehrplan Gymnasium Physik, Sächsisches Staatsministerium für Kultus und Sport
http://www.schule.sachsen.de/lpdb/web/downloads/lp_gy_physik_2011.pdf?v2
- Statista, 2015. Das Statistik-Portal.
<https://de.statista.com/statistik/daten/studie/71085/umfrage/verteilung-der-einwohnerzahl-nach-bundeslaendern/>
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366 (1881), 3717–3725. doi: 10.1098/rsta.2008.0118 .
- Yadav, A., Good, J., Voogt, J., Fisser, P. (2017). Computational Thinking as an Emerging Competence Domain. In, M. Mulder (ed.), *Competence-based Vocational and Professional Education*, pp 1051-1067, Springer Verlag. Doi:10.1007/978-3-319-41713-4_49.

Computational Thinking and Subject Learning and Teaching in K-12

Gamified Mathematics practice: Designing with e-commerce and computational concepts

Chien-sing LEE, Jing-wen WONG, Peh-yenc EE,

Department of Computing and Information Systems,
Sunway University, Malaysia.

chiensingl@sunway.edu.my, jing.w8@imail.sunway.edu.my, peh.e@imail.sunway.edu.my

ABSTRACT

This paper addresses two problems which usually occur in learning Mathematics: first, students who face difficulty understanding and are too shy to participate in discussions and subsequently do not manage to resolve their doubts, and second, dull e-learning websites. The many rules in Mathematics compounds the problem further. We thus aim to address these problems through a gamified e-commerce-oriented Mathematics learning practice system, Alzebra, for informal learning. Focusing on principles of Information Systems Analysis and Design, e-commerce-oriented computational concepts are embedded in the game to motivate online practice. The system concept, design methodology and user testing outcomes are presented. Significance lies in deriving perception towards gamification and components which users liked or disliked and the efficacy of our hybrid approach in systems development.

KEYWORDS

Design; gamification; Mathematics practice; e-commerce; computational concepts.

1. INTRODUCTION

Blended learning is increasingly popular. However, educators may not be available face-to-face at all times to help students with their problems. Hence, two problems need to be addressed (Chen & Jones, 2007; Li, 2016). First, students who have difficulties grasping concepts in class and who are shy. They tend not to participate in the activities or interact with their peers in class even though they do not understand what they are learning in class. Instead, they would be forced to revise topics on their own. The second problem arises if the e-Learning platforms are dull and mostly text-based or unexciting.

In the learning of Mathematics (MVID, 2016), the enormous number of rules that need to be followed often makes understanding complex Mathematics frustrating. These pose challenges to motivate students to access online materials to carry out self-study and to keep them engaged throughout their online learning process. Hence, we aim to develop a gamified computer-aided learning system, Alzebra, to carry out revision and reinforcement outside the classroom.

Bearing in mind several learning strategies, our objectives are to:

- a) assess the improvements that can be made to existing related systems and choose the best features that can be adopted;
- b) explore the possibilities of gamified learning in online education.

2. RELATED WORK

2.1 Learning difficulties faced by students in Mathematics

Other than the small number of students who have been identified as having dyscalculia (Mathematics learning disability), there are a few reasons why students face difficulties grasping concepts in Mathematics (Taylor & Galligan, 2006; MVID, 2016):

- a) students who experience this problem often possess characteristics such as lack of confidence due to constant failure, do not activate prior knowledge to solve problems, have trouble memorizing basic Math functions, have problems focusing when facing questions involving multiple steps, lack of cognitive thinking skills, and afraid of being wrong..
- b) there are also a few teacher-related variables, which cause students to have problems in understanding concepts in Mathematics.
- c) research has also suggested that curriculum-related variables such as spiralling curriculum causes students to experience significant problems learning and applying Mathematics concepts. This may be due to cognitive overload.

2.2 Mathematics Learning Strategies

Many Mathematics strategies have been around and are used by educational institutions. Some of the approaches available are classroom-based techniques such as metacognitive strategies, cognitive strategies, and social or affective strategies while others are software-based approaches where educational technology, is used as one of the teaching strategies (Taylor & Galligan, 2006; Yang, Chang, Cheng, & Chan, 2016; Centre for Advanced Research on Language Acquisition, 2016; MVID, 2016). Due to the fact that both means of instructional delivery methods are diverse, the outcomes from both approaches in relation to students' performance may however, also differ.

2.3 Gamification in Educational Websites

Gamified learning is a term used to describe the integration of game mechanics in learning the process to make instructions more engaging and fun. It has the potential to help the way students need to feel engaged when learning, that is, through growth and advancement, recognition and rewards, a higher goal to pursue, and a sense of teamwork

Kapp's (2012) study states that there are researchers who suggest that gamification can be used as a tool in education to spark interest in students to learn. Moreover, students who have used a gamified e-Learning platform produce higher practical test scores compared to those who use the non-gamified version.

Furthermore, according to the Gamification Survey carried out by Talent LMS (2016), 79% of the participants have shown a positive attitude towards the integration of gamification in their university or institution. Out of 75%, the participants are already gamers themselves whereas 50% of them play casually and 27% of them moderately to fairly often. In addition, over 60% of the participants would be motivated by leader boards and increased competition between students and 89% would be more engaged with an e-learning application if it had a point system.

Based on this, it can be concluded that the strong interests of the participants in game may indicate that implementation of gamification in educational websites can be accomplished. An example of existing systems implementing gamification in education is the Khan Academy.

3. SIGNIFICANCE

There are several contributions from this study:

- a) This study contributes to how Information Systems Analysis and Design principles and computational components integrated with e-commerce and gamification can be used to design applications which have the potential to motivate online practice. The application of computational concepts to the real-world corresponds with computational thinking (Wing, 2006).
- b) A deeper understanding of the perceptions of the student community needs to be first identified and designed for if gamification is to work well. This finding supports that of an earlier paper (Wong & Lee, 2016).
- c) Consistent with (TalentLMS, 2016), prior user gaming experience influences acceptance of gamified applications.
- d) Object-oriented design is cost-effective and sustainable.

4. METHODOLOGY

4.1 Sample

The sample students are 10 students who are weak in Mathematics studying at the pre-university level. Learning

Mathematics online is foreign to them though they know that these systems exist. The testing period is one week each (initial survey and user-testing).

4.2 Procedure

Adopting agile methodology, rapid prototyping and design thinking, two phases are carried out, involving two iterations in each phase. The first phase involves the initial survey and the second phase the beta testing. These are elaborated on below.

First phase:

An evaluation of existing e-learning websites (objective a above) based on Nielsen's criteria: Website Content, Website Interface, and Website Functionality is carried out to determine improvements which can be made and opportunities for developing systems meeting our objectives.

Subsequently, for design and development, the first iteration includes the basic content management features. The second iteration includes the add questions page, practice page, show hint and check answer section in practice page, quiz page, and view result page. Next, a survey is carried out to determine students' attitude towards the use of technology in learning Mathematics online.

Second phase:

The first iteration involves the point-accumulating function in the prototype, user garden page, marketplace page, leaderboard section, and FAQ page. Within the user garden, e-commerce-oriented activities are introduced to motivate practice. The second iteration includes a comment section in all topic pages.

5. SYSTEM CONCEPT

This Website is developed to integrate the concept of gamified learning into an online educational website. On registering to become a member, users will get their very own garden which they can visit through the link located at the User Login Information dropdown list.

The system works like a normal online educational website which enables students to learn on topics, do practices, attempt quizzes, and view their results. Other than those minimal requirements for an educational website, an extra enhancement is incorporated into the system, that is, a point-accumulating system.

The main idea of this point accumulating system is to encourage students to revise topics by attempting practices and quizzes and keep them engaged when they are on the website. For every correctly-answered question, students get to earn points. Points will also be given when students have completed a quiz.

On registering, each student will have their own page called "My Garden", this is where they have plants they need to nurture in order to gain more points. The way they cultivate their plants is by buying materials from a page called

“Market Place”. There are four items that need to be used on each seed in order for it to be fully grown. Once the plant is fully grown, it can be sold to earn points. The names of students with the highest points accumulated will be shown on a leaderboard at the home page.

The final system’s use case diagram (Figure 1) and user interfaces (Figures 2, 3, 4 and 5) are presented below.

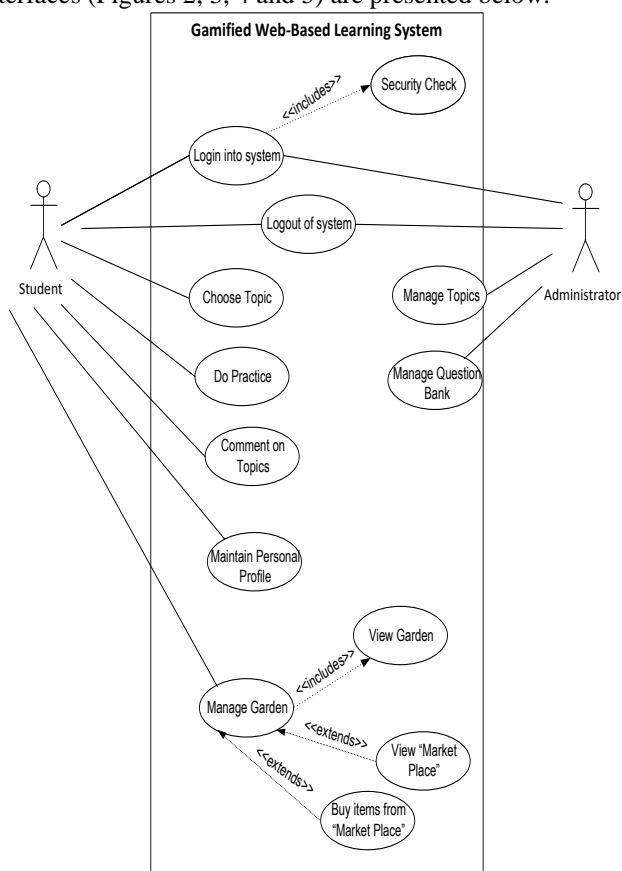


Figure 1. Use Case Diagram

Figure 3 shows the layout for the User Garden Page. Computational concepts are used here to design the game story for the topic trigonometry. There are altogether ten slots in the garden. Five of the slots are open, while the other five of them are locked. These open slots will be where the seeds received by users are planted. The locked slots need to be purchased for 150 points each to get more space for users to plant their seedlings. Upon registering, each user will be given a seed. Each seed has to be watered, weeded, fertilized, and cleared of pest once respectively to be completely grown. To grow the seeds, users have to visit the market place to buy the materials needed.

Grown plants can be sold by users to earn more points. A mysterious seed will be given for free to users every time their accumulated points have reached 100 points. Each time users get a seed, it will be automatically be planted in one of the open slots in their garden. If users do not have any open slots left, the seed will be discarded. Users will be competing with other members on the system to get the highest ranking on the leaderboard based on the points they have accumulated. To earn points, users must do practices. With each question correctly answered, users will get five points. Besides that, users can also gain points by doing quizzes.

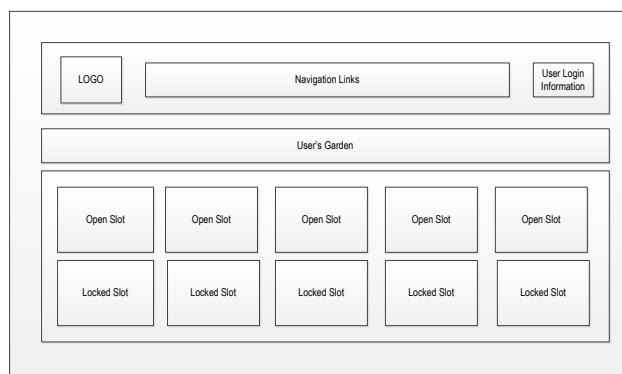


Figure 3. User's Garden

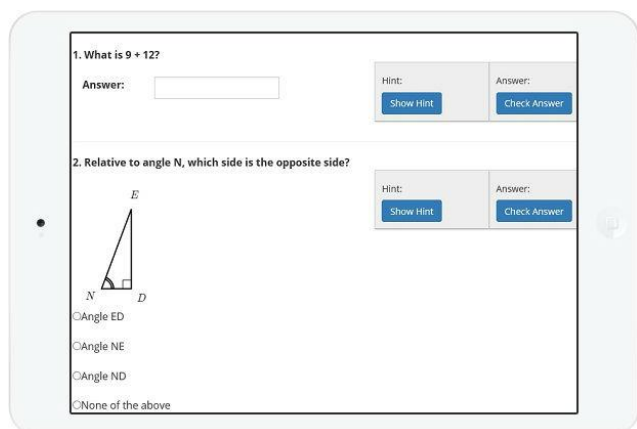


Figure 2. Practice Page Screenshot

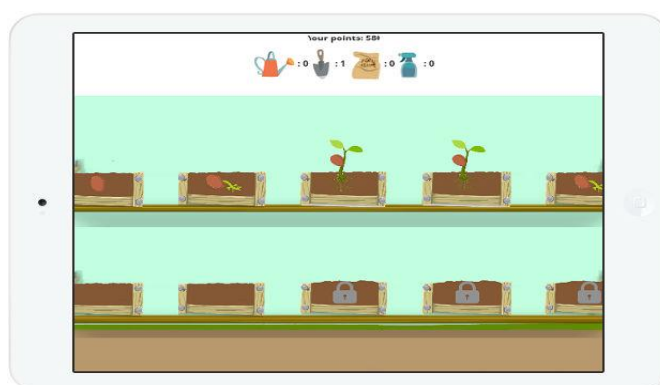


Figure 4. User's Garden Page Screenshot



Figure 5. Market Place Screenshot

6. FINDINGS

6.1 Findings from the initial survey

Findings from the initial survey involving 30 users are as follow:

- 73% of the participants think that learning Mathematics will be useful for them in their future. 20% of them do not think that learning Mathematics will be important to them. The rest have a neutral attitude towards learning Mathematics.
- 61% of the participants have a great experience learning. 23% of them do not have a positive attitude towards classroom experience while 16% of them have a neutral attitude towards the classroom experience. Those who have a positive attitude towards the learning experience mostly understand what they have learned in class and have friends who can help them when they face problems understanding Mathematics.
- 73% of the participants think that they can do well in Mathematics. 13.5% of them think that they are not good in Mathematics and another 13.5% of the participants have a neutral attitude towards their self-confidence in Mathematics.
- 47% of the participants find learning Mathematics through technology easier to understand. 23% of them find it uncomfortable learning Mathematics through technology while 30% of them have a neutral attitude towards the use of technology in learning Mathematics.

Subsequently, based on the result of this survey, a basic gamified Web-based learning platform was developed.

6.2 Findings from beta testing

Beta testing involves 10 students. Findings based on the Technology Acceptance Model indicates that overall,

Alzebra has received positive response from the ten participants. Furthermore, it is observed that:

- All of the participants managed to use the website without any difficulties (ease of use). 80% of the students think that the design and layout of the system are acceptable. They can navigate through the site easily. 20% of the students find the layout of the website can be made more interesting.
- 70% of the participants have a positive attitude towards the concept of game in educational website. They are able to accept gamification in education while 30% of the students prefer the normal web-based learning system with no gamified concept included.
- Similarly, 60% of the students think that online competition such as leader board is challenging and fun while the rest think that it is annoying.
- 60% of the students will use the comment section provided to interact with other members online when they are facing problems understanding the concepts of the topic while 40% of them think it is unnecessary.
- A majority agree that they can do better if the website is incorporated as part of the Mathematics subject.
- Three of the suggestions made are to improve the gamification portion in the website. The point-accumulating system can be motivating as the majority (6 out of 10) finds online competition stimulating while the rest of the participants think that the reward provided is gimmicky.

7. CONCLUSION

From these results, students appear to prefer attractive websites and prefer not having their performance or comments displayed publicly. The latter is typical of more conservative Asian culture and the influence of prior gaming experience towards acceptance of gamification in e-learning. Furthermore, there is improvement in acceptance towards such learning environments compared to the initial survey. This finding supports that of two other related projects, i.e., on teaching augmented reality to youths and e-crafting (Wong & Lee, 2016; Low & Lee, 2016). Noting the comments and suggestions above, to meet the needs of a majority of the users who are not gamers, we need to improve on our design with game mechanics which matter to the users.

This is a course assignment. The sample size is small and findings are not generalizable. Nevertheless, we hope that eventually, this e-Learning platform will provide a better user experience for students, hence keeping them enthused to carry on their self-studies outside of a classroom.

8. REFERENCES

- Andriotis, N. & Panagiotis, Z. (2014). *Gamification Survey Results*. *TalentLMS Blog*. Retrieved March 01, Centre for Advanced Research on Language Acquisition. Learning Strategies for Mathematics (nd). Retrieved March 01 2016, from http://carla.umn.edu/cobalt/modules/strategies/strategies/CALLA_Table10-3.pdf
- Chen, C. C. & Jones, K. T. (2007). Blended Learning vs. Traditional Classroom Settings: Assessing Effectiveness and Student Perceptions in an MBA Accounting Course. *Journal of Educators Online*, 4 (1). Retrieved February 29, 2016, from <http://files.eric.ed.gov/fulltext/EJ907743.pdf>
- Dominguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fernández-Sanz, L., Pagés, C. & Martínez-Herráiz, J. Gamifying learning experiences: Practical implications and outcomes. *Computers & Education*. 63, 380-392. Retrieved February 29, 2016, from <http://thinkspace.csu.edu.au/itc510amandaforde/files/2014/07/Gamifyinglearningexperiences-1z3dgt7.pdf>
- Li, W. (2015). Is your eLearning boring? Spice it up With These 3 Innovative eLearning Ideas. *eLearning Industry*. Retrieved January 22, 2016, from <http://elearningindustry.com/is-elearning-boring-3-innovative-elearning-ideas>
- Kapp, K. M. (2012). *The gamification of learning and instruction game-based methods and strategies for training and education*. San Francisco: Pfeiffer.
- 2016, from <http://www.talentlms.com/blog/gamification-survey-results/>
- Khan Academy. <https://www.khanacademy.org/>.
- Low, H. S. & Lee, C. S. (2016). *e-Crafting*. Capstone project, Sunway University, Malaysia.
- MVid. Understanding Math Learning Problems. (n.d.). Retrieved January 18, 2016, from <http://www.coedu.usf.edu/main/departments/sped/mathvids/understanding/understanding.html>
- Taylor, J., & Galligan, L. (2006). Mathematics for Maths anxious tertiary students: Integrating the cognitive and affective domains using interactive multimedia. *Literacy & Numeracy Studies*, 15(1), 23-44.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Wong, C. K. & Lee, C. S. (2016). A better understanding of how gamification can help improve digital lifestyles,” *International Conference on Virtual Systems and Multimedia*, Kuala Lumpur, Malaysia.
- Yang, E. F. Y., Chang, B., Cheng, H. N. H. & Chan, T. W. (2016). Improving pupils’ Mathematical communication abilities through computer-supported reciprocal peer tutoring. *Educational Technology & Society*, 19(3), 157-169.

How Computer Scientists and Computing Teachers Think Differently in the Concepts to be Included in a Secondary School Computing Curriculum

Chiu-fan HU, Cheng-chih WU*, Yu-tzu LIN, An-tsu WANG

National Taiwan Normal University, Taiwan

chiufan@ntnu.edu.tw, chihwu@ntnu.edu.tw, linyt@ntnu.edu.tw, atwang710@gmail.com

ABSTRACT

The new K-12 computing curriculum draft for Taiwan secondary schools was designed to launch in 2018 but the draft only outlined themes and contents for students to learn, without further details on key concepts to be covered in the contents. Therefore, in 2016, a Delphi study was conducted to survey the opinions about what “key learning concepts” should be included for implementation at the secondary level based on the draft. By adopting the Delphi method, different viewpoints from computer scientists and secondary school computing teachers were collected to build consensus of key concepts through a series of convergence. Based on the research results, we found the computer scientists and computing teachers had opposing opinions about whether the secondary school students should learn the advanced concepts. The purpose of this study was to understand the different views on learning concepts of the draft between two groups. The data analyzed in this study were based on the Delphi survey in 2016. This study found computer scientists tended to be more conservative about this issue, therefore they suggested that the advanced and theoretical concepts are not essential at the secondary level, e.g., recursion, searching, sorting, data compression, data conversion, and divide and conquer. This was because the computer scientists considered these concepts as what they had studied in college. Rather, computing teachers knew how to simplify these concepts for teaching at the secondary level. The research findings can serve as useful references for revising and implementing the computing curriculum in the future.

KEYWORDS

Computing curriculum, Delphi survey, Computational thinking, K-12 education

1. INTRODUCTION

International Society for Technology in Education [ISTE] (2014) and Computer Science Teachers Association [CSTA] believed CT (Computational Thinking) is essential for students, so collaborated on a project to prepare students to become computational thinkers who should understand how digital tools could help them solve problems. In fact, in 2011, CSTA has issued a revised K-12 computer science curriculum standard (CSTA, 2011) that addresses the importance of computer science in concept, practice and the application of cross-discipline and outlines five strands of the curriculum standards (CT, collaboration, computing practice and programming, computers and communication devices, and community,

global, and ethical impacts). In this curriculum standards, CT is regarded as an important concept to enable students to apply appropriate strategies and tools to solve complex problems effectively in the real world. Department for education of England [DOE] (2013) issued a national curriculum that renamed the subject name ICT into Computing. The statutory programmes of study clearly stated that “A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world.” Students in England are taught to become digitally literate—able to use, and express themselves, solve problems and develop ideas. Australian Curriculum, Assessment and Reporting Authority [ACARA] (2013) also published a new the Foundation to Year 12 Australian Curriculum. The Technologies learning area draws two subjects, which are “Design and Technology” and “Digital Technologies”. The goal of Digital Technologies is to enable students to define, design and implement digital solutions. The learning strands include: (1) knowledge and understanding; students learn digital systems and representation of data; (2) processes and production skills: students can collect, manage and analyze data, and create digital solutions by certain skills (investigating and defining; generating and designing; producing and implementing; evaluating; and collaborating and managing). Throughout the learning contents of the Foundation to Year 12 Australian Curriculum, one of vital aims is to develop understanding the skills in computational thinking in F-Y12.

According to the development of ICT (Information and Communication Technology) curriculum standards, planned by the ISTE, CSTA, Australian and England, it's obvious that computer science has become an important field in K-12 schools and the concept of CT (Computational Thinking) is the essence of the recent curriculum development. Meantime, ISTE, CSTA and related organizations in Australian and England outlined the learning contents of curriculum standards, built up a glossary to define the words and phrases of the curriculum standard and provided teaching guidelines, examples and the portfolio of students' work. Those resources could equip teacher for their future instruction under the new curriculum standards.

In 2014, the ministry of education [MOE] in Taiwan announced the Grades 1-12 Curriculum Guidelines. In the Grades 1-12 Curriculum Guidelines, a new learning area - Technology which includes Living Technology and Information Technology, is added and will be launched in 2018 academic year. Computational thinking is the center theme of the new K-12 computing curriculum draft in

Taiwan. The purpose is to develop students' ability of computational thinking and to enable them to formulate abilities of problem solving, team collaboration, creativity, and communicating and expressing. It is also meant to make students' great attitude in information society and habits of utilizing information technologies.

In elementary school, the instruction focuses on the practice and application to cultivate students' ability of utilizing information technologies in daily life. In Grades 7-9, the instruction emphasizes to utilize information technologies and computational thinking to solve problems. In Grades 10-12, by exploring computer science, the instruction enables students to realize the principle of computational thinking and it puts emphasis on the integration and application.

A required course, named Information Technology, will require students to take one hour per week or equivalent time in Grade 7 through Grade 9 and two credits at Grades 10 to 12. The learning performance of Information Technology includes (1) computational thinking and problem solving, (2) information and collaborative creation, (3) ICT and communication and expression, and (4) using attitude of ICT. The learning contents includes six categories, which are (1) programming, (2) algorithm design, (3) system platform, (4) data representation, processing and analysis, (5) application of ICT and (6) ICT and social, legal and ethical issue. In fact, the curriculum draft only outlines the key themes in each content category. (see Appendix A for details).

Three selective courses are planned to implement at Grades 10-12, including Advanced Programing, Projects in ICT application, and Robotic Programming. Students could learn more professional knowledge and skills needed for future career and learning performances which include expression and sequence of operation, ICT creation and ICT attitude.

To provide teachers with great flexibility, the K-12 computing curriculum draft only outlines themes and topics for students to learn, without further details on key concepts to be covered. It will certainly be a challenge to the textbooks writers and the computing teachers to decide which concepts to teach and which to skip.

We conducted a Delphi technique study to give suggestions for the "key learning concepts" to be included in the K-12 computing curriculum draft in 2016. The first draft of Delphi survey questionnaire consists of six themes (programming; algorithm design; system platform; data representation, processing and analysis; application of ICT; ICT and social, legal and ethical issue) and 117 learning concepts developed from the computing curriculum. After three-rounds of survey, the expert panel derived 92 key learning concepts. Concepts which are not yet get consensus from the experts had been examined and provided recommendations for when included in learning. The results serve as useful references for computing teachers and textbook authors when implementing the new curriculum. At the same time, the study found the computer scientists and computing teachers had different points. Computer scientists put great attention on the depth, breadth and prior knowledge of learning contents,

and meanwhile computing teachers expressed their views by teaching experiences. Therefore, we believed there were some different points on those learning concepts between these two groups and it is worth discussing.

So, the purpose of this study was to understand the different views on learning concepts of the k-12 computing curriculum draft between computer scientists and computing teachers.

2. METHOD

The data analyzed in this study were based on the Delphi survey which investigating the key concepts recommended for a proposed national K-12 computing curriculum in Taiwan. The Delphi survey was to derive consensus from a panel of twenty-one experts, including nine computer scientists and twelve secondary school computing teachers. Computer scientists were college professors from CS related fields, who had research or educational experiences in secondary school CS education; whereas, computing teachers were certified secondary Computing teachers who owned either a CS related master or PhD degree.

This study was to analyze the results of the last survey. After three rounds of surveys, all of experts should know each other's opinions, and the results of this phase present their thoughtful views. In the results and discussions section, we looked into the experts' views from surveys and round-table discussion to express the different points in the two groups.

3. RESULTS

This section describes the results from the Delphi study following the six categories of learning contents: (1) programming; (2) algorithm design; (3) system platform; (4) data representation, processing and analysis; (5) application of ICT; (6) ICT and social, legal and ethical issue. Based upon the results, the different views on the secondary level learning concepts of computing curriculum draft between computer scientists and computing teachers had been depicted.

3.1. Programming

Experts from the two groups had different views about what "key learning concepts" in three learning contents.

(1) "1.1 Basic concepts of programming languages." Computing teachers thought students should learn the types of programming languages to grasp the concepts about the categories of programming paradigms. However, in many of computer scientists' opinions, the programming paradigm was too abstract for students to learn before they learn this programming language. Computer scientists also argued that such learning concepts would cause the students to learn in rote because they could not understand about the paradigm exactly.

(2) "1.3 Implementation of arrays." The main difference was on the concepts of time complexity. Computing teachers thought this was an essential concept that students should know; however, in computer scientists' opinion it was too difficult for secondary school students to learn.

(3) “1.6 Implementation of fundamental algorithms-recursion, searching, sorting, and divide and conquer.” The difference between the two groups was the learning of recursion. Lots of computer scientists thought recursion was too difficult, but in computing teachers’ opinion, the concept of recursion was related to mathematical induction taught in mathematics class, therefore it would not be too difficult for students to understand, in addition, this concept was important for realizing the computing power by programming.

3.2. Algorithm design

Experts from the two groups had different views about what “key learning concepts” in four learning contents.

(1) “2.1 Basic concepts of algorithms-problem decomposition and flow control.” The key difference was in what grade should this content be taught. The content was planned to taught at grade 7. From some computer scientists’ perspective, teaching of this concept should focus on expressing the problem-solving process by flowcharts or pseudocodes rather than introducing the algorithm to the grade 7 students.

(2) “2.4 concepts and application of fundamental data structures- tree and graph.” Both groups agree that it was important. However, due to the limited instruction time, experts from the two groups had different opinions about what concepts or skills about data structures should be included, e.g., concepts and application of tree traversal.

(3) “2.5 concepts and application of fundamental algorithms—recursion and divide and conquer.” Computer scientists thought recursion was too difficult.

(4) “2.6 performance analysis of algorithms.” Computer scientists suggested that at secondary level, the content of performance analysis of algorithms should be focused on observing the programming efficiency rather than deriving the time/space complexity, which was too theoretical. Students only need to learn simple tools and methods of performance analysis and its concepts of program optimization.

3.3. System platform

In this topic, experts from the two groups had different view about what “key learning concepts” in one learning contents.

“3.1 The development and evolution of system platforms.” Both groups, especially the computer scientists, thought it was not essential to learn the evolution and personage of the computer science. They did not think those concepts are not required for understanding computing, and they were afraid that students would only memorize the knowledge. The other reason was that there are debates on representative personages.

3.4. Data representation, processing and analysis

Experts from the two groups had different views about what “key learning concepts” in the learning contents.

“4.3 Concepts and methods of data processing-data consolidation, data compression, data conversion.” Since

the computer scientists worried that the topic would be taught as the “data mining” class in college, they argued that this topic should only include fundamental concepts and methods, e.g. the principles and importance of data conversion, rather than theoretical parts of the data conversion algorithms. More practical examples and hands-on activities should also be included in instruction. But the computing teachers did not mention about this.

3.5. Application of ICT

Experts from the two groups had consensus in all of learning contents, “5.1 Data processing projects-data searching, data organization and representation, data computing and analysis”, “5.2 Information technology projects-multimedia applications, programming applications”, and “5.3 Concepts and tool use in collaborative digital creation”. Because the 5.1 and 5.2 were planned at grades 7 to 9 in the computing curriculum draft, some computer scientists believed that only senior high school students had enough abilities to conduct projects.

3.6. ICT and social, legal and ethical issues

Experts from the two groups had different views about what “key learning concepts” in one learning contents.

“6.3 Information security, ethics, and legislation.” Although the opinion was not obviously disagreed, some computing teachers pointed out that due to students have limited knowledge about advanced techniques in computer science, the learning concepts of information security should be emphasized on its importance but not the algorithms.

4. DISCUSSIONS

In summary, the major differences between the computer scientists’ and the computing teachers’ opinions on learning concepts are as the following:

(1) Projecting the curriculum from the learning contents of different learners

Computer scientists in our study were college professors in CS related fields, and they tended to view the k-12 computing curriculum draft on the basis of courses in universities, especially when the same terminologies were adopted in the curriculum draft. Besides, some learning contents outlined in the curriculum draft are either well-known courses in college computing curriculum or matured fields in computing discipline, e.g., algorithm, machine learning, and big data. Based on the computer scientists’ past learning and teaching experiences in universities, computer scientists tended to disagree to teach secondary school students with similar concepts taught in the college, especially those theoretical or difficult concepts for college students. These differences could be found in learning concepts of programming, algorithm design and data representation, processing and analysis. Computer scientists were opposed to including theoretical or advanced concepts such as recursion, searching and sorting, data compression and data conversion.

Computing teachers, on the other hands, had practical experiences in teaching computing in secondary schools. They knew how to explain the core aspects of these concepts by demonstrating relevant examples and supporting students' learning with appropriate tools (for example, visualization programming platforms), therefore were more inclined to include the theoretical contents.

(2) Philosophy on how CS should be taught in schools

It is common in Taiwan that computing subject at secondary level is either application-oriented, in which application software (such as Microsoft Offices or Photoshop) are delivered and the students only learn low-level skills to use computer software/hardware; or knowledge-oriented, which focuses more on theories and factual knowledge in computer science. It is a general impression that computing subjects in the schools are either too trivial or too theoretical.

Computer scientists in this study, in general, hoped to inspire students' interests in learning computing and foster their problem-solving ability by applying computing skills for everyday life or careers, rather than to learn factual knowledge. They thought secondary school students should learn by hands-on experiences and develop their interests in computing fields, especially in programming and algorithm design. They also suggested that the history of computer science is not required in learning computing, because they were afraid that would become rote learning. The points could be found from their disagreement with teaching the development and evolution of programming language and system platform.

(3) Different professional background in computing science

Some learning contents in the k-12 computing curriculum draft are new ones which were not addressed in the previous K-12 computing curriculum (e.g., divide and conquer). Computing teachers were not familiar with those contents, therefore had a conservative attitude toward inclusion of these concepts.

As mentioned above, computing teachers might associate some concepts listed in the k-12 computing curriculum draft with computer application software (e.g., excel and photoimpact) in present teaching experiences and as a result it could affect their opinions of learning concepts. This point could be found in the concepts of data compression and data cleaning included in 'Data representation, Processing and analysis.' Those are complex and professional fields in the computing discipline, and therefore computer scientists delivered different opinions.

5. CONCLUSIONS

The purpose of the study was to explore the differences of opinions about the learning concepts of the K-12 computing curriculum draft between computer scientists and computing teachers, and discuss possible reasons.

Based upon the results, it was revealed that computer scientists had more conservative attitude toward including advanced learning concepts. They suggested that some

advanced and theoretical concepts should not be included at the secondary level, e.g., recursion, searching, sorting, data compression, data conversion, and divide and conquer. This might due to the lack of teaching experiences in secondary schools, or the past learning or academic experiences in these topics. Generally, computer scientists seemed not to believe that abstruse theories could be simplified and taught at secondary level by applying examples or learning tools.

The experts in this study were selected with the consideration of their professional knowledge, teaching experiences, and familiarity of the K-12 computing curriculum draft. On the basis of the results the study suggests that, in the related researches, the selected experts could have all of the specific knowledge or be fully debriefed the rationale of the K-12 computing curriculum draft prior the research. As a result, they can consider computing learning concepts from both sides of theory and practice.

Based on the research findings, the researchers in the future can develop the materials and tools or conduct experiments in secondary schools to test the different views which have found from the study (e.g., recursion, searching and sorting, data compression, data conversion and divide and conquer). Furthermore, the results can serve as useful references for revising the computing curriculum in the future.

6. ACKNOWLEDGMENT

This research is supported by the Ministry of Science and Technology, Taiwan, R.O.C. under Grant no. NSC 103-2511-S-003-023.

7. REFERENCES

- Australian Curriculum, Assessment and Reporting Authority (ACARA). (2013). *The Foundation to Year 12 Australian Curriculum*.
- Computer Science Teachers Association (CSTA). (2011). *CSTA K-12 Computer Science Standards*.
- Department for Education, England. (2013). *National curriculum in England: computing programmes of study*.
- International Society for Technology in Education (ISTE). (2014). *Computational Thinking for all*.
- Ministry of Education, Taiwan. (2014). *The Grades 1-12 Curriculum Guidelines*. Taipei: Author.
- Ministry of Education, Taiwan. (2016). *The k-12 computing curriculum draft*.

Appendix A: The Learning Contents for the Draft Computing Curriculum

1. Programming
 - 1.1 Basic concepts of programming languages (G7)
 - 1.2 Structured programming-conditional structures and loops (G7)
 - 1.3 Implementation of arrays (G8, G10-G12)

- 1.4 Concepts of modular programming (G8)
- 1.5 Implementation of modular programming (G8)
- 1.6 Implementation of fundamental algorithms—recursion, searching, sorting, and divide and conquer (G10-G12)
2. Algorithm Design
 - 2.1 Basic concepts of algorithms—problem decomposition, flow control (G7)
 - 2.2 Concepts and application of arrays (G8)
 - 2.3 Introduction to basic algorithms—searching and sorting (G8)
 - 2.4 Concepts and application of fundamental data structures—tree and graph (G10-G12)
 - 2.5 Concepts and application of fundamental algorithms—recursion and divide and conquer (G10-G12)
 - 2.6 Performance analysis of algorithms (G10-G12)
3. System Platform
 - 3.1 The development and evolution of system platforms (G9)
 - 3.2 The architecture and operations of system platforms (G9)
 - 3.3 Concepts of networking techniques (G9)
 - 3.4 Concepts of network applications (G9)
 - 3.5 Task management and resources allocation, distributed system, routing (G10-G12)
 - 3.6 The future trends of system platforms (G10-G12)
4. Data Representation, Process, and Analysis
 - 4.1 Principles and methods of data digitalization (G9)
 - 4.2 Methods of digital data representation (G9)
 - 4.3 Concepts and methods of data processing—data consolidation, data compression, data conversion (G9)
 - 4.4 Basic concepts of big data (G10-G12)
 - 4.5 Basic concepts of data mining and machine learning (G10-G12)
5. Application of ICT
 - 5.1 Data processing projects—data searching, data organization and representation, data computing and analysis (G7)
 - 5.2 Information technology projects—multimedia applications, programming applications (G9)
 - 5.3 Concepts and tool use in collaborative digital creation (G10-G12)
6. ICT and social, legal and ethical issues
 - 6.1 Future study and career development of information technology related areas (G7, G9, G10-G12)
 - 6.2 Impacts of information technology on society and human life (G8, G9, G10-G12)
 - 6.3 Information security, ethics, and legislation (G7, G8, G10-G12)
 - 6.4 Fair-use doctrine for information technology (G7, G10-G12)

Teaching Computational Thinking by Gamification of K-12 Mathematics: Mobile App Math Games in Mathematics and Computer Science Tournament

Chee-wei TAN*, Pei-duo YU, Ling LIN, Chung-kit FUNG, Chun-kiu LAI, Yanru CHENG

Department of Computer Science, City University of Hong Kong
cheewtan@cityu.edu.hk, peiduoyu2-c@my.cityu.edu.hk, hkalexling@gmail.com,
sa9510@gmail.com, cklai24-c@my.cityu.edu.hk, 1754103129@qq.com

ABSTRACT

Teaching computational thinking can be viewed as cultivating the capacity for logical thinking and problem-solving skills applied to foundational subjects such as mathematics. We report a pilot study on how carefully-designed mobile app games that gamify elementary algebra learning are used in an annual computer science tournament and also at an annual mathematics festival in Hong Kong. We define mathematics gamification as the process of embedding mathematical concepts and their logical manipulations in a puzzle game-like setting aided by computing technologies. We have evaluated the learning efficacy of our mobile app games to gain numeracy proficiency in an annual computer science tournament for middle school students in Hong Kong.

KEYWORDS

Mathematics education, mathematics gamification, mobile app games, pedagogy, K-12 mathematics.

1. INTRODUCTION

Marvin Minsky, in his 1970 Turing Award Lecture, asserted that, “*The computer scientist thus has a responsibility to education...how to help the children to debug their own problem-solving processes.*” [1]. Minsky pointed out that cultivating the capacity for logical thinking and problem-solving skills of students, while they are young, to learn foundational subjects such as mathematics is of the essence. The emphasis is on the tools and motivations for students to acquire problem-solving skills in lifelong learning of mathematics. Computer science and its software technologies might just offer an intriguing way for students to persist and persevere in learning mathematics. We describe a pilot pedagogical study on learning K-12 mathematics through mathematics gamification ideas and tested at a computer science tournament in Hong Kong.

We define mathematics gamification as the process of embedding mathematical concepts into puzzle game-like instantiations that are aided by computing technologies. We focus on the software development for typical computing technologies run on a mobile device of the learner. Game playing is essentially the manipulative of mathematical objects or structures in a logical manner such to acquire useful mathematical insights that otherwise are not obvious or taught in traditional classrooms. Also, the engaging game-like nature can potentially motivate students and serve as instructional

tools for regular practice to gain proficiency in mathematics and numeracy.

2. ALGEBRA GAMIFICATION

Elementary algebra—the first cornerstone of K-12 mathematics—has been highlighted by the National Academy of Engineering in [3] as a critical area to improve in K-12 mathematics learning (in fact touted as an *Algebra Challenge*). How should the *Algebra Challenge* be addressed from teaching computational thinking skills with an aim to underpin the foundation of learning mathematics? Can this complement traditional classroom learning? It has been recently recognized (among them are mathematicians like Keith Devlin from Stanford University) that game-playing activities allow players to grasp mathematical concepts and foster a sense of motivation that leads to numeracy proficiency especially when the game is designed to embed abstracted mathematical subjects [4-8].

Algebra gamification is a pedagogical approach to learning elementary algebra. This can be especially useful when used at an early stage of a K-12 education to give students a heads up with learning an advanced topic that might only be encountered later in classroom teaching. In this paper, we report on how this idea of mathematics gamification can be designed as mobile game apps that are suitable for middle school students when the mobile apps are deployed in mathematics-related game tournaments and then to analyze preliminary efficacy of learning behavior based on collected data. Put simply, this teaches students *how to think (about learning mathematics) at multiple levels of abstraction – the goal of teaching computational thinking* [2]. The particular instance of gamifying algebra in this paper is due to Terence Tao, a mathematician at the University of California, Los Angeles, who remarked in his online blog article [9] on “Gamifying Algebra” that:

The set of problem-solving skills needed to solve algebra problems (and, to some extent, calculus problems also) is somewhat similar to the set of skills needed to solve puzzle type computer games, in which a certain limited set of moves must be applied in a certain order to achieve a desired result one could then try to teach the strategy component of algebraic problem-solving via such a game, which could automate mechanical tasks such as gathering terms and performing arithmetic in order to reduce some of the more frustrating aspects of algebra ... Here, the focus is not so much on being able to supply the correct answer, but on being able to select an effective problem-solving strategy.

Tao's insightful remarks aptly highlight two key facts, namely that (i) certain kinds of K-12 mathematics are amenable to game design that can motivate student to learn, and (ii) problem-solving skills can be cultivated through this gamifying process as a means to learning the mathematical subject. In other words, there are several ways to solve elementary algebra — strategizing moves in a mathematical puzzle game is one of them. With the aid of computing technologies, this introduces novel perspectives to learn elementary algebra for young students. Also, in [10], Tao developed a software mock-up of the game as shown in Figure 1.

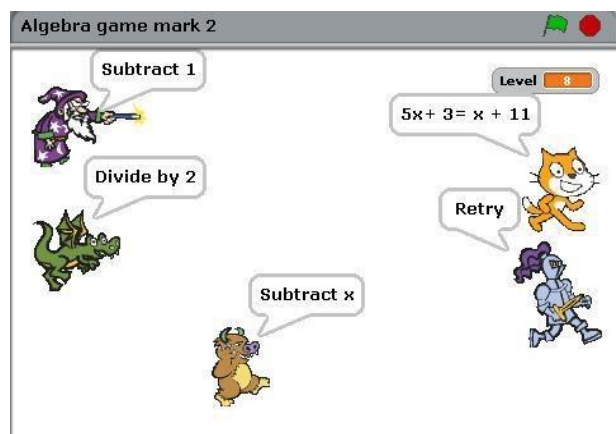


Figure 1. Terence Tao's software mock-up.

The idea of Tao's algebra game is to reduce a given linear algebra equation to a form with only " x " and a numerical value on the lefthand and righthand side respectively through a selection of a finite number of given clues. Let's give an example using a screenshot of the game as shown in Figure 1. Initially, the puzzle state is the algebra equation " $5x + 3 = x + 11$ " and the given clues are the three possibilities "Subtract 1", "Divide by 2" and "Subtract x ". The player chooses one of the three possibilities by clicking on the avatar icon. Say, suppose the player chooses "Subtract 1", the algebra equation (correspondingly, the puzzle state) then changes to " $5x + 2 = x + 10$ " (since both sides of the original equation " $5x + 3 = x + 11$ " get subtracted by one).

One possible "solution" to the puzzle given in Figure 1 is the sequence of "Subtract 1" then "Subtract x " then "Divide by 2" then "Subtract 1" and then finally "Divide by 2" to yield " $x = 2$ ". This requires a total of five moves to reach the desired state. It is important to note that what matters is not the final value of x , but it is rather *the inquisitive problem-solving process while playing that is valuable*.

The benefit to computational thinking is obvious: students learn a foundational subject (e.g., mastering algebra) while playing. There are several intriguing questions: first, how to engineer the difficulty level of the game automatically? Second, how does a computer (not human player) solve a given puzzle efficiently, i.e., with the fewest number of moves? And, third, how to engage the human players in an entertaining manner so that they keep on playing it and, unknowingly, develop a better number sense or

mathematical intuition and that such an improvement can be measured? These questions were explored in The Algebra Game Project founded by the first author [11], and detailed answers to these questions along with the software development will be published in other venues.

3. MOBILE APP GAMES WITH MATHEMATICS GAMIFICATION

In this section, we briefly describe the mathematics gamification building on Tao's algebra game in [9] and the software implementation in two mobile apps. One mobile app is Algebra Maze and the other is Algebra Game, and both mobile apps are freely-available for download at the Apple iOS Store or Google Play Store [11].

In Algebra Maze, we combine maze solving and linear algebra equation solving together as shown in Figure 2, which is the game play screen shot of Algebra Maze. The goal is to move the purple avatar toward the treasure (i.e., equivalently solving the linear equation). Each movement of the avatar corresponds to a mathematical operation on the equation given below the maze. For example, the button " $+1x$ " corresponds to the avatar moving upward one unit, and the button " $+2$ " corresponds to the avatar moving leftward two units. Hence, the operation on x is an up-down movement and the operation on the constant is a left-right movement of the avatar. With the rules above, we can deduce that the position of the avatar also has algebraic meaning, i.e., each position in the maze represents a different equation with the same solution but with different coefficients or constants.

In the initial levels, the treasure is visible, and in subsequent higher levels, the treasure is rendered invisible, i.e., hidden from the player as shown in the right hand side of Figure 2. Hence, the player needs to make use of the "information" in the given equation to deduce the location of the treasure. In some levels, the player has to first get a key, which is in a certain position of the maze, before opening a locked door located nearby to the treasure. This setting is equivalent to asking the player to reach to a certain equation first before they solve this equation. Finally, when the avatar locates the (potentially hidden) treasure, the algebra equation will be in the desired form " $x = \text{numerical_solution}$ ", i.e., the puzzle is solved.

In Algebra Game, we split the clues into two parts, one is the operator " $+$, $-$, $*$, \div " and the other one is the operand such as the x -term or the number as in Figure 3. Hence, the combination of the clues is more general than the original design. The goal in Algebra Game is the same as Tao's algebra game: To reduce a given linear algebra equation to " $x = \text{numerical_solution}$ " through a selection of a finite number of given clues. As shown in the right hand side of Figure 3, if the player "drag" the button " \div " and "drop" it on the button " 2 " then the equation will be divided by two on both side. Algebra Game is not only about solving a linear equation, it also contains other mathematical insights. For example, consider an equation " $x - 23 = 2$ " with the clues " $+$, $-$ " and " 2 , 3 ", then this is equivalent to ask if we are able to use 2 and 3 to construct

23. The players can develop their number sense through playing this game. Let us use another example, consider an equation “ $24x=48$ ” with the clues “ $*$, \div ” and “2, 3”, then this is equivalent to asking the players to factorize 24 by using 2 and 3 (prime numbers). Other than the factorization concept, there are many instances of mathematical manipulations that can be embedded in the Algebra Game such as the Frobenius’s problem (also known as the coin problem) in the form of making up a number from two given clues. In essence, given the available clues at each level, the player can only perform a limited number of operations, and this restriction helps to stimulate computational thinking in finding a sequence of moves to solve the problem. Coupled with the mathematical analysis underpinning the difficulty level design in the Algebra Game, players can further develop intuitions to develop mathematical insights and intuition while playing the Algebra Game.

The mathematics gamification process also requires analyzing the scoring at each puzzle game level that can be evaluated according to different reasonable design criteria. For example, scoring can be evaluated in terms of the number of moves needed or the speed to solve each level in the case of the Algebra Game and the number of “redo” on hitting obstacles or the speed to locate the hidden treasures in the case of the Algebra Maze. Furthermore, concrete mathematics can be purposefully interleaved at certain levels of the games. For example, after a consecutive sequence of games involving factorization in the Algebra Game, the mathematical statement of *The Fundamental Theorem of Arithmetic* (stating that all natural numbers are uniquely composed of prime numbers) can be displayed to the player in order to highlight game features (e.g., the prime numbers as clues) with the mathematical rudiment. In this way, the players learn about fundamental mathematical knowledge (such as *The Fundamental Theorem of Arithmetic* in Euclid’s Elements that is not typically taught in classroom). In summary, we find that the Algebra Maze and the Algebra Game can provide players with new perspectives to gaining new mathematical insights while training their individual number sense and problem-solving skills that are critical to develop their capacity to view mathematics at multiple abstract levels.

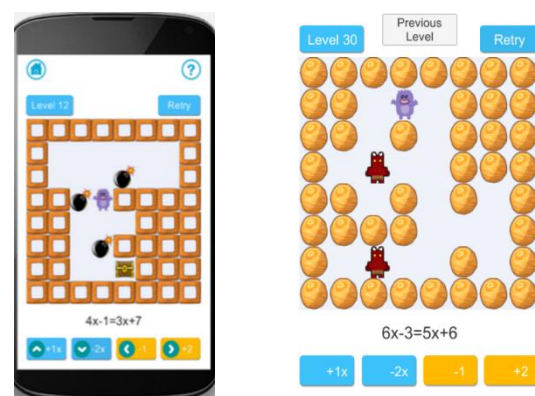


Figure 2. Algebra Maze mobile app game with maze-like gamification design and freely-available for download at iTunes App Store and Google Play Store.

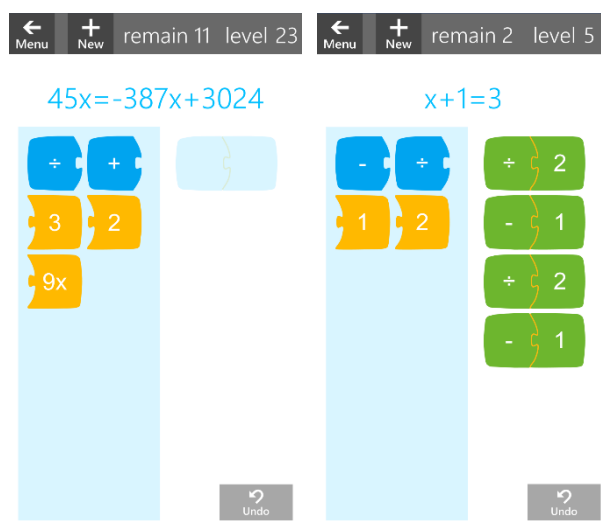


Figure 3. Algebra Game mobile app game with the selected choice displayed at the right-hand side and freely-available for download at iTunes App Store and Google Play Store.

4. CASE STUDY OF COMPUTER SCIENCE CHALLENGE TOURNAMENT IN HONG KONG

The Computer Science Challenge (CS Challenge) was a tournament organized by the Department of Computer Science at City University of Hong Kong on 21 May 2016 for both primary and secondary school students in Hong Kong [12]. A pair of students forms a team, and there were altogether 32 teams from 19 primary schools, and 53 teams from 33 secondary schools, making a total of 170 students. One of the tasks in the CS Challenge was called the *Algebra Game Challenge*, in which the Algebra Maze and Algebra Game are used for the primary school students (as shown in Figure 4) and the secondary school students (as shown in Figure 5) respectively. Each of these two tasks lasts for a fixed duration of twenty minutes. We experiment with a pedagogical initiative of teaching computational thinking to the participants as follows: a workshop for all participants was held a month before the CS Challenge, whereby participants were introduced to basic computer science knowledge and the mathematics

behind the games. On the day of the CS Challenge, however, participants used the mobile app software described in Section 3 that allow more diverse game-playing dynamics and also enable the use of data collection and data analytics to capture the users' game-playing behavior. The mobile apps in [11] were not available to the participants as they were posted online after the CS Challenge was over.

We describe in the following how the first task of Algebra Game and Algebra Maze based on data analytics of the data collected in the tournament. We analyze the performance evaluation of learning efficacy based on the time spent at each level, each move that a user has taken, and the number of "redo" times at each level. The difficulty at each level is calibrated based on our mathematical analysis of the game (from easy to hard), and we expect to have a reasonable difficulty curve so that players gain confidence instead of frustration at early levels. Let us evaluate Algebra Game first. In Figure 6, we see that the number of student drops sharply, about twenty percent, from Level 9 to Level 10 which can also be observed in Figure 7, the time spent in Level 10 almost doubled in Level 9. In fact, the number of moves needed in Level 10 is also almost double that needed in Level 9 as shown in Table 1. We conclude that the total number of moves needed at each level is a crucial factor in the difficulty-level calibration design of the game.

Interestingly, the average number of moves needed at Level 12 is around 8.8, and yet the time spent in Level 12 is the highest. This implies that the total number of moves needed is not the only factor that may affect the difficulty of the game for human players. Finally, the reason for the longer time spent in Level 1 is that students are initially warming up (as they get familiar with the game interface and rules). If we omit the information in Level 1 and proceed to compute the correlation coefficient between the time spent and the average number of moves taken, then we have a correlation coefficient that is 0.807 which reveals that they are highly correlated for the twelve levels being analyzed.

Table 1. Table of average moves taken by players in each level of the Algebra Game.

Level	1	2	3	4	5	6
Average Moves	3	2.65	3.84	3	5	5
Level	7	8	9	10	11	12
Average Moves	4	5	4.77	10.1	20.1	8.8



Figure 4. Primary school student tournament of Algebra Maze at the Computer Science Challenge on May 2016.



Figure 5. Secondary school student tournament of Algebra Game at the Computer Science Challenge on May 2016.

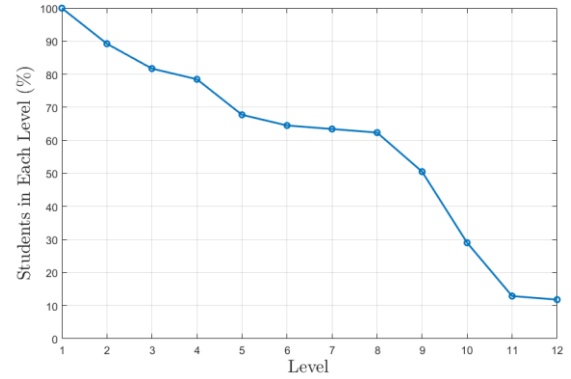


Figure 6. Percentage of number of students vs. the total number of completed levels of Algebra Game during the twenty-minute duration. An unlimited number of levels were designed in the Algebra Game.

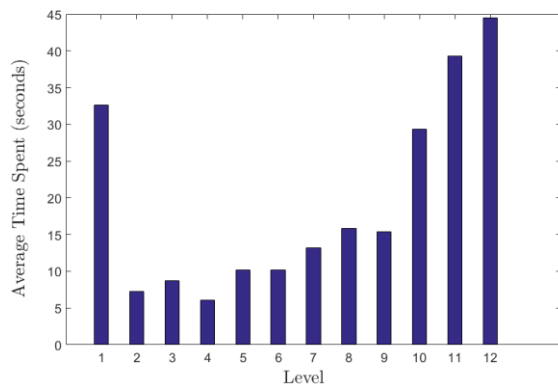


Figure 7. Average time a player spent at each level of the Algebra Game.

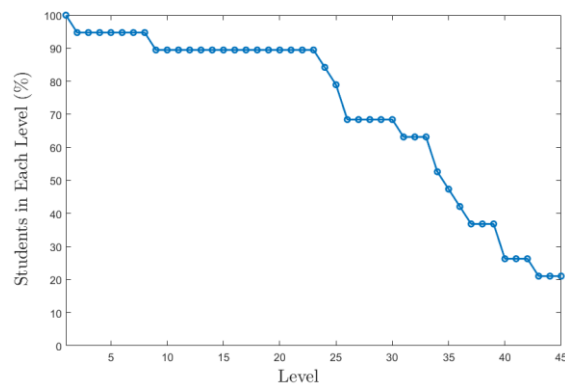


Figure 8. Percentage of number of students vs. the total number of completed levels of Algebra Maze during the twenty-minute duration. A total of forty-five levels were designed in Algebra Maze.

5. CONCLUSIONS

We described a pilot study on teaching computational thinking by cultivating the capacity for logical thinking and problem-solving skills of students using carefully-designed mobile app games based on the mathematics gamification of elementary algebra learning. Through the aid of mobile computing technologies, the logical manipulatives of the mathematical Algebra Game and Algebra Maze are embedded within puzzle game-like instantiations in a logical flow to catalyze the development of mathematical intuitions and insights. Through competitive game-playing in a Computer Science Challenge tournament, we studied the learning efficacy of the game software. We have also recently deployed the game software for non-competitive learning at the Julia Robinson Mathematics Festival in Hong Kong [13] with the aim to share the mathematics behind the Algebra Game and to evaluate the efficacy of these software tools to learning advanced mathematics, the topic of which will be reported elsewhere.

6. ACKNOWLEDGEMENT

We thank Sui-Yan Kwok, Yilan He, Zheng Zeng, Yun-Fung Tin for assistance in software development.

7. REFERENCES

- [1] Minsky, M. (1970). *Form and Content in Computer Science – 1970 ACM Turing Award Lecture*. Journal of the Association for Computing Machinery, Vol.17, No. 2, 1970.
- [2] Wing, J. M. (2006). *Computational Thinking*. Communications of the ACM, Vol. 49, No. 3, pp. 33-35.
- [3] Lavernia, E. J. and VanderGheynst, J. S. (2013). *The Algebra Challenge*. The Bridge, Vol. 43, No. 2, The United States of America National Academy of Engineering: <https://www.nae.edu/File.aspx?id=88638>
- [4] Devlin K., *Mathematics Education for a New Era: Video Games as a Medium for Learning*. A. K. Peters, Ltd. Natick, MA, USA, 1st Edition, 2011
- [5] Pope, H., & Mangram, C. (2015). *Wuzzit Trouble: The Influence of Math Game on Student Number Sense*. International Journal of Serious Games, 2(4), 5. Retrieved from <http://documents.brainquake.com/backed-by-science/Stanford-Pope-Mangram.pdf>
- [6] Shapiro, J. *Video Games Are The Perfect Way to Teach Math, Says Stanford Mathematician*, Forbes Magazine. 2013 Aug.29. Retrieved from <https://www.forbes.com/sites/jordanshapiro/2013/08/29/video-games-are-the-perfect-way-to-teach-math-says-stanford-mathematician/#7e90367a385b>
- [7] Mackay, R. F. (2013, March 1). *Playing to learn: Can gaming transform education?*. Graduate School of Education, Stanford University. Retrieved from <https://ed.stanford.edu/news/playing-learn-can-gaming-transform-education>
- [8] Novotney, A. (2015). *Gaming to learn*. Monitor on Psychology, 46(4), p. 46, Retrieved from <http://www.apa.org/monitor/2015/04/gaming.aspx>
- [9] Terence Tao, *Gamifying Algebra*, Terence Tao's Wordpress blog article, 2012: <https://terrytao.wordpress.com/2012/04/15/gamifying-algebra>
- [10] Terence Tao, *Software Mockup of Algebra Game*, 2012: <https://scratch.mit.edu/projects/2477436>
- [11] The Algebra Game Project: <http://AlgebraGamification.com>
- [12] Computer Science Challenge Game Tournament in Hong Kong: <http://CSChallenge.io>
- [13] The Julia Robinson Mathematics Festival in Hong Kong: <http://www.AlgebraGamification.com/JRMF>

Profile of a CT Integration Specialist

Joyce MALYN-SMITH¹, Irene A. LEE², Joseph IPPOLITO¹

¹ Education Development Center

² Massachusetts Institute of Technology

jmsmith@edc.org, ialee@mit.edu, jippolito@edc.org

ABSTRACT

This paper describes the findings from the Education Development Center's (EDC) project on Computational Thinking (CT) called "Broadening Participation of Elementary School Teachers and Students in Computer Science through STEM Integration and Statewide Collaboration." It presents the process used to define the primary job functions and work tasks of a CT Integration Specialist in today's education settings. Authors describe how the requisite knowledge, skills and practices of the CT integration specialist were assembled and vetted. The article presents ways this profile can be used to guide elementary school teachers in integrating CT into their classrooms and as a framework to guide the development of CT learning activities and assessments, then sets the directions for future work.

KEYWORDS

Computational thinking, elementary education, integration, skill standards, workforce development.

1. INTRODUCTION

There has been much recent debate about the definitions of CT and the relative merits of different definitions (Wing, 2006; Committee for the Workshops on Computational Thinking, 2010; Cuny, Snyder & Wing, 2010; Barr & Stephenson, 2011; Lee et al., 2011; Grover & Pea, 2013; Engelmann, 2014; Voogt et al., 2015; Weintrop et al., 2016). This debate has created confusion within education circles. EDC was funded by the National Science Foundation to develop curricular modules "iMods" that integrate CT into science and math lessons at the 1st -6th grade levels. To build a better understanding of CT integration within elementary schools, the project spearheaded an effort to research what Kindergarten-8th grade (K-8) teachers need to know and be able to do to successfully integrate CT into classroom lessons. The project team conducted a modified DACUM (Developing a Curriculum) process, a well-researched methodology used to develop curriculum designed to prepare people for career success, to create the profile. The resulting profile includes the universe of work tasks associated with a specific job and the skills, knowledge and behaviors needed to conduct those work tasks successfully.

This paper describes the research process and findings resulting from the application of a modified DACUM process to describe the work activities of the classroom teacher who integrates CT into disciplines they teach and/or the specialist who assist them in CT integration. At the time this research was conducted, no such job as the "CT Integration Specialist" existed.

2. BACKGROUND

Building on a legacy of experience in developing national skill standards (Leff & Aring, 1995; Norton, 1997; Dahms & Leff, 2002; Education Development Center, 2012; Ippolito, Latcovich, Malyn-Smith, 2008) EDC's STEM+C project produced an occupational profile that defined, in concrete terms, the work of a "Computational Thinking Integration Specialist"; then identified and validated with expert CT educators the "computational thinking" skills and competencies that are used by CT integration specialists. An occupational profile of the type produced by EDC presents a detailed synopsis of what a particular professional does, as well as the skills, knowledge and behaviors that enable him/her to succeed in the workplace. These occupational profiles provide important information to educators who use them to guide the design of pre-service and in-service curriculum and training programs. The process employed was one that had been used successfully for decades to develop curriculum for technical occupations (Big Data) and national skill standards for emerging industries (Biosciences) and industries undergoing substantive changes in professional and technical job responsibilities (Human Services). This research is not intended to contribute to or address ongoing discussions related to transfer. It is meant to help describe what classroom teachers and/or CS support specialists need to know and be able to do when integrating CT into various disciplines.

3. DEVELOPMENT PROCESS

A modified DACUM process (Developing A CUrriculUM) (Norton, 1997) was used to produce the profile of the CT Integration Specialist. DACUM is an internationally known methodology used by expert practitioners in an occupational field to identify the major areas of work and the constituent tasks that define successful job performance. The DACUM method has been used internationally for more than half a century to develop curricula based on identified core workforce competencies. This process rests upon three basic principles:

- Expert workers can describe and define their jobs more accurately than anyone else.
- An effective way to define a job is to precisely describe the tasks that expert workers perform.
- All tasks, in order to be performed correctly, demand certain knowledge, skills, resources, and behaviors.

Traditional DACUM analyses invite expert practitioners representing a single occupation. The "modified"

DACUM approach used successfully by EDC engages expert workers from a range of related occupations who share a common core of work tasks, knowledge, and skills.

The work process involved three distinct steps: building a team, defining a learning occupation, developing a profile of the CT integration specialist.

Building a team: EDC assembled a project team that included 3 highly qualified skill standards developers experienced in conducting occupational analyses. Their first and perhaps most important task was to recruit individuals recognized by their peers as experts in integrating CT into K-8 curricula to serve on a national panel to define the work of a CT Integration Specialist. The DACUM process requires the following criteria be used to select panelists: 1) recognition by peers as experts in their field (integrating CT into disciplinary lessons), and 2) a minimum of 2 years experience in performing the work described in the profile. In addition every effort was made to include gender, geographic, and cultural diversity among the panelists. The 11 panelists represented a range of subject areas, occupational levels, and work settings (elementary and middle school educators, technology specialists and computer science educators). For the purposes of this research we dubbed these individuals “CT Integration Specialists” and framed the profile around their definition of that role. The panel was convened by the National Science Foundation funded STEM+C project entitled “Broadening Participation in Elementary School Teachers and Students in Computer Science through STEM Integration and Statewide Collaboration.” The panel’s work sessions were held at Education Development Center’s world headquarters in Waltham, MA in August 2016.

Defining a learning occupation: The panel’s first task was to come to agreement on the learning occupation defining the CT Integration Specialist. A “learning occupation”, adapted from best practices in Germany and other countries (Leff & Aring, 1995) is an invented construct used to describe a set of cross cutting tasks, skills, knowledge and attributes required to perform a range of job functions conducted in a group of related real-life occupations. In this regard the learning occupation of CT Integration Specialist was meant to help define the work of both teachers at various grade levels, and also technology and computer science specialists who support classroom teachers as they integrate CT into their curricula. For the purpose of developing curricula, the profile identifies the universe of work functions and tasks that a CT Integration Specialist *might* be called upon to perform. The Learning Occupation provides a framework for development of courses/professional development. It is not meant to be a “job description” performed by a single individual.

The first task undertaken by this panel of experts was to discuss and refine the proposed Learning Occupation so that it captured the essence and commonalities of their

own work. Panelists came to consensus around the following definition that set the boundaries for the occupational profile. “The CT Integration Specialist recognizing that CT is integral to learning, is a teacher who models and integrates CT across academic disciplines and/or /out-of-school activities by establishing an inclusive culture while using, modifying and creating CT activities and assessments of student learning.”

Developing a profile of the CT Integration Specialist: Once the learning occupation was defined and agreed upon, the expert panel developed a profile of the CT integration.

The profile development work session involved panel members in a guided dialogue that includes brainstorming, identifying and organizing work responsibilities, revisiting and refining those work responsibilities until consensus was reached. The ensuing guided dialogue provided descriptions of concrete, observable activities for which the panelists use CT and that met the definition of the Learning Occupation. The work session yielded the first draft of a profile of the CT Integration Specialist. Subsequent to the two-day work session, the expert panel members reviewed and commented on the draft profile.

The panel identified 6 large functional groupings or “job functions” described as follows: “A Computational Thinking Integration Specialist ...”:

1. Establishes a CT learning environment in the classroom
2. Creates lesson plans that integrate CT with all subjects
3. Facilitates student learning
4. Engages stakeholders in support of CT learning
5. Teaches students to apply CT concepts and practices
6. Engages in professional learning/development in support of CT and content areas

The panelists identified 68 activities/work tasks performed by CT Integration Specialists described in the learning occupation. Each of the 68 tasks was grouped under the job function (or duty) category to which it best corresponded. In addition, the panelists developed lists of the Skills, Knowledge and Behaviors of CT Integration Specialist as well as selected Equipment/Tools and Supplies used as they are engaged in those activities listed (see Table 1 below for the duties and tasks of a Computational Thinking Integration Specialist).

4. NEXT STEPS: Although the profile identified the work tasks in which CT Integration Specialists engage, concrete examples that describe what this work “looks like in action” are needed to build a strong dialog between CT Integration Specialists and non-computer science educators who are struggling to

understand CT and connect it to learning objectives in their classes. In summer and fall of 2017 EDC will host a CT Workshop for a small group of NSF ITEST and STEM+C grantees whose work focuses on integrating CT into various disciplines. Workshop participants will develop a framework that describes what CT looks like “in action” at various grade levels and along learning progressions related to the concept/constructs that undergird their CT work. The profile of the CT Integration Specialist, along with other CT resources will inform the development of that framework.

Profiles such as these have many uses. The profile can be used:

- by teachers learning how to integrate computational thinking into their classes and as a professional development resource;
- by faculty at the post-secondary and secondary levels to design or modify programs and/or courses;
- by school superintendents and other employers creating job descriptions and interview questions for hiring; and
- by job seekers developing their resumes and preparing for interviews.

Authors hope that this work pushes the field forward in thinking about what it takes to integrate computational thinking into the disciplines; and helps to clarify the emerging skill sets needed for teachers seeking to become CT Integration Specialists.

Table 1: Duties and tasks of the CT integration specialist (excerpted from EDC’s Profile of a Computational Thinking (CT) Integration Specialist, 2016.)

Duty 1: ESTABLISHES A CT LEARNING ENVIRONMENT IN THE CLASSROOM
Task 1A. Creates student-centered spaces that accommodate their needs.
Task 1B. Obtains physical resources (e.g., technology).
Task 1C. Establishes expectations and procedures.
Task 1D. Establishes systems for management of resources (e.g., equipment).
Task 1E. Creates an environment respectful of divergent ideas and abilities.
Task 1F. Promotes student dispositions conducive to CT (e.g., celebrates failure as a first attempt in learning, encourages persistence when setbacks occur, develops iterative refinement of initial ideas).
Task 1G. Fosters collaboration.
Task 1H. Promotes student leadership (e.g. engages mentors).
Task 1I. Promotes ethical use of resources.
Task 1J. Encourages multiple solutions to the same problem.
Duty 2: CREATES LESSON PLANS THAT INTEGRATE CT WITH ALL SUBJECTS
Task 2A. Determines CT outcomes as manifested within the subject matter.
Task 2B. Researches lesson plans that lend themselves to CT.
Task 2C. Collaborates with peers to identify how to integrate CT (e.g., vertical alignment, cross curricular connections).
Task 2D. Aligns lessons to standards (e.g., NGSS, Common Core, Mathematics, CSTA, State CS standards).
Task 2E. Develops a course outline that indicates location and allocation of time for CT integration.
Task 2F. Identifies students’ prior knowledge and interests.
Task 2G. Creates Kinesthetic computer based learning activities that

infuse CT (e.g., algorithms, data, modeling/simulation, programming).
Task 2H. Creates differentiated instruction to accommodate different learners (e.g., remediation & enrichment activities, IEP, ESL).
Task 2I. Provides modifications and accommodations for students with special needs.
Task 2J. Provides supports to address common misconceptions about CT and the discipline.
Task 2K. Procures materials and resources.
Task 2L. Creates contingency plan (e.g., technology failure).
Task 2M. Creates an assessment rubric.
Duty 3: FACILITATES STUDENT LEARNING
Task 3A. Builds CT vocabulary.
Task 3B. Uses CT technical language (e.g., algorithm, abstraction, function, debugging) in a consistent manner.
Task 3C. Uses models to simulate real world phenomena and processes. (e.g., makes connections between physical models of real world phenomena and computer models of the same phenomena).
Task 3D. Provides examples of CT ranging from concrete to abstract (e.g., links common practices from the classroom to how they pertain to computers using CT).
Task 3E. Provides open-ended guiding questions that promote CT (e.g., provides opportunities for open ended artifact construction that engage students in abstraction and automation).
Task 3F. Calls out CT throughout the day (e.g., through thinking aloud, identifying when algorithms are being used).
Task 3G. Exposes students to artifacts that use CT to solve real world problems.
Task 3H. Empowers students to take ownership of learning through the Use, Modify and Create process.
Task 3I. Manages student groups to promote collaborative learning (e.g., pair programming, gender balance).
Task 3J. Provides opportunities to practice CT within subject matter content (e.g., identifies patterns in nature and man made phenomena, decomposes problems into sub-problems, develops algorithms).
Task 3K. Provides opportunities to collect, use and represent authentic data for storage, manipulation, and analysis on a computer.
Task 3L. Provides access to people who use CT in their professional work (e.g., via teleconferencing, field trips, speakers).
Task 3M. Provides time for iteration design and development cycles.
Task 3N. Provides opportunities for students to share their understanding of CT.
Task 3O. Guides students through self- evaluation and reflection.
Task 3P. Provides opportunities to program at various levels of difficulty within subject matter content.
Task 3Q. Organizes students into teams based upon interests/ programming platforms.
Duty 4: ENGAGES STAKEHOLDERS IN SUPPORT OF CT LEARNING
Task 4A. Hosts events that promote CT (e.g., model CT using technology, Hour of Code, robotics competitions).
Task 4B. Communicates importance of CT and demonstrates activities related to CT to community.
Task 4C. Advocates for CT integration at various venues (e.g., discipline specific conferences).
Task 4D. Shares CT standards and CT information resources.
Task 4E. Works with school system administration to support CT (e.g. attends board meetings, invites administrators to observe).
Task 4F. Seeks support from professionals who use CT in their everyday work.
Task 4G. Promotes CT through political avenues.
Task 4H. Partners with businesses for CT support (e.g., funding, speakers, field trips).
Duty 5: TEACHES STUDENTS TO APPLY CT CONCEPTS AND PRACTICES
Task 5A. Teaches how to compare and contrast human and computer intelligences (in terms of power of and limits of each).
Task 5B. Teaches how to assess the difficulty of a problem from a human and a computer perspective.
Task 5C. Teaches user interface of tool or environment for creating an artifact (e.g., program, model, animation, mobile app.).

Task 5D. Teaches Computer Science concepts (e.g., instructions, sequences, expressions and evaluation, booleans, variables, and control mechanisms such as loops, conditionals, randomness) and their applications.
Task 5E. Teaches iterative development.
Task 5F. Teaches debugging techniques.
Task 5G. Engages students in CT practices. (e.g., collecting and analyzing data, using models, giving instructions, decomposition).
Task 5H. Engages students in analyzing artifacts made by others using CT.
Task 5I. Teaches how to modify artifacts to address a new problem.
Task 5J. Teaches how to create their own artifacts using abstraction and automation.
Task 5K. Teaches how to use an artifact to study or solve a real-world problem.
Task 5L. Teaches how to determine whether the artifact has met its intended purpose.
Task 5M. Assesses student learning of CT concepts (e.g., CSTA, ISTE, Common Core).
Duty 6: ENGAGES IN PROFESSIONAL LEARNING/ DEVELOPMENT IN SUPPORT OF CT AND CONTENT AREAS
Task 6A. Stays current with emerging technologies, methods, tools, standards, curriculum, programming languages / theory.
Task 6B. Maintains professional qualifications.
Task 6C. Seeks out mentors.
Task 6D. Mentors others.
Task 6E. Attends relevant conferences.
Task 6F. Engages in cross-discipline training.
Task 6G. Participates in professional organizations.
Task 6H. Develops ambassadors to promote CT.
Task 6I. Participates in a personal learning network (e.g. Twitter, Google docs).
Task 6J. Shares best practices and resources.

5. REFERENCES

- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54.
- Committee for the Workshops on Computational Thinking, 2010. *Report of a Workshop on the Scope and Nature of Computational Thinking*, National Research Council, Washington, D.C., National Academies Press.
- Cuny, J.E., Snyder, L., Wing, J.M., 2010. *Computational Thinking: A Definition*. Unpublished manuscript.
- Engelmann, C. A. (2014). *Final evaluation report:*

Computational Thinking Symposium held on Saturday, December 7, 2013 at the Santa Fe Institute. Omaha, NE: Einstein Evaluation Group.

Dahms, A.S., Leff, J.A. Industry Expectations for Entry-Level Technical Workers, *Biochemistry and Molecular Biology Education*, 30, 4, (2002) 260-264.

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, 42(1), 38–43.

Integrating IT skills in law, public safety, corrections and security career programs, second ed. (2008). Education Development Center, Inc., Newton, MA (Accessed 1/11/12 from <http://itac.edc.org/>)

Ippolito, J. Latcovich, M., Malyn-Smith, J. (2008). *In fulfillment of their mission: The duties and tasks of a Roman Catholic priest*, NCEA Publication, NY, NY.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Mayln-Smith, J., and Werner, L. (2011). Computational thinking for youth in practice, *ACM Inroads* Vol. 2 No.1.

Leff, J., & Aring, M. (1995). *Gateway to the Future: Skill Standards for the Bioscience Industry*. Newton, MA: Education Development Center, Inc., 27.

Norton, R. (1997). *DACUM Handbook*. Columbus, OH: The Ohio State University.

Profile of a Computational Thinking (CT) Integration Specialist. (2016). Education Development Center, Inc., Newton, MA

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715–728. Retrieved from <http://link.springer.com/article/10.1007/s10639-015-9412-6>

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, 25(1), 127–147.

Wing, J. (2006, March). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Enhancing the Link between Parent-Child in Learning Computational Thinking

Jane Yat-ching WONG^{1*}, Pam Hau-yung WONG¹, Robert Kwok-yiu LI^{1,3}, Chee-wei TAN^{1,2},

¹ CoolThink@JC Office

² Department of Computer Science

³ Department of Physics and Materials Science

wong.jane@cityu.edu.hk, pam.wong@cityu.edu.hk, Robert.Li@cityu.edu.hk, cheewtan@cityu.edu.hk

ABSTRACT

To foster new generations becoming creators of technology, CoolThink@JC (“Computational Thinking Education,” n.d.), a four-year project sponsored by Jockey Club Charities Trust in Hong Kong, aims to advocating that “computational thinking is a fundamental skill for everyone” (Wing, 2006, p.33). The project targets to upper primary school students, and in that parent education is one of the components. Particularly, parent education focuses on parent-child relationship in learning computational thinking. Thus, we propose several approaches involving coding hands-on workshop, instructional video learning and unplugged activities to enhance the parent-child mode of learning in a large-scale project involving 32 primary schools in Hong Kong.

KEYWORDS

Computational thinking, pedagogical challenges related to parent-child education, CoolThink@JC

1. INTRODUCTION

CoolThink@JC (“Computational Thinking Education,” n.d.) is a four-year project created and funded by the Jockey Club Charities Trust, with support from the Education University of Hong Kong (EdUHK), the Massachusetts Institute of Technology (MIT), and the City University of Hong Kong (CityU).

Parent education is one of the highlights of the project. Its aim is to foster parents’ understanding of computational thinking through activity-based workshops which involve adult-youth partnership.

Based on our observation and ongoing experience of organizing parent education workshop in primary schools, we put forward pedagogical challenges and propose ideas in the following sections, for enhancing the link between parents and children in learning computational thinking in Hong Kong.

2. PEDAGOGICAL CHALLENGES

Here we list down several pedagogical challenges:

- 1) What are effective methodologies under which parent and child can learn basic computational thinking knowledge?
- 2) Can parent-child pair learning complement traditional classroom learning?

- 3) Parents come from diverse background and have different opinions on education. Can computational thinking education play a definite role to convert parents’ value in computational thinking?
- 4) How to instill the role of game playing to enhance parent-child communications? For example, it has been recently recognized (among them are mathematicians like Keith Devlin from Stanford University) that game-playing activities allow players to grasp mathematical concepts and foster a sense of fun leading to proficiency in foundational subjects related to Science-Technology-Engineering-Mathematics (STEM) (Mackay, 2013).
- 5) Can we scale up the parent-child learning activities to accommodate a large number of learners?

These challenges may call for the need to trade-off between learning efficacy at an early stage of a K-12 STEM education and the implementation complexity of the learning tasks. The effectiveness of parent-child learning should then be evaluated based on a long-term study with rigorous statistical evaluation.

3. PROPOSED APPROACHES

Here we list down several approaches that will be carried out under Coolthink@JC (“Computational Thinking Education,” n.d.) and subject to pedagogical efficacy evaluation.

- 1) Small Workshop: Coding experience workshops that involve hands-on activities for creative applications by both parents and children. It will be run once per school annually in a 2-hour workshop format, and the venue will be held at the school compound. For each of these workshops, we accommodate 20-30 parent-child pairs and the content in these workshops follow closely the progress of the computational thinking curriculum being taught in the specific school. These small workshops serve to inform parents of a common language in learning computational thinking.
- 2) Large Workshop: These coding experience workshops typically involve around 100 parent-child pairs. The materials in these workshops can resonate with world-wide STEM movements

such as the MIT Scratch Day (“About Scratch Day,” n.d.) and the Hour of Code (“The Hour of Code,” n.d.). The workshop can be instrumented with a competitive nature to allow parents to work with their child to collaborate on building a working system and to self-assess on their learning of computational thinking and how it is relevant to the broader theme of STEM subjects in schools. This means that the contents for these workshops ought to be more experimental, even allowing for trial-and-error form of learning that encourages parent-child pair to put into practice what they have learnt in schools. For example, this might be using MIT Scratch or the MIT App Inventor software to control hardware or to configure a robot to navigate a maze. Trial-and-error form of learning is the main focus.

- 3) Instructional Video: To complement the above workshop, we will put online electronic instructional videos in the form of Massive Open Online Courses (MOOC) that are comprised of easy-to-follow short video clips and to deliver the content to parents. In this way, parents who are unavailable to attend the previously-mentioned workshops or who wish to refresh their memory after attending these workshops can benefit from these instructional digital materials.
- 4) Unplugged Activities: This involves the design and delivery of activities that do not require a computer and focus on fundamental ideas behind computer science such as algorithm and its connections to mathematics (“Unplugged Activities in Code.org,” n.d.). The idea is to create several tables each having its own computer science-related theme and allow the parent-child pair to work through problem sets that are facilitated by personnel trained in computer science (“Unplugged Activities in Teaching London Computing,” n.d.). The problems are designed in such a way that it starts easy (potentially involving playing digital games) and progressively become more challenging in the latter part, whereupon the role of parents in guiding the child becomes more apparent. Note that the definition of digital games means any form of games that involve discrete mathematics. A collaborative form of learning is the main focus in these unplugged activities.

In above approaches, due to the different number of parent-child pair and the environment under which the activities that are carried out, it is necessary to carefully study how the learning contents, platforms and technologies can be leveraged for effective delivery of computational thinking education. We will study some of these issues to address the learning efficacy and the effectiveness of technologies to address the challenges listed on Section 2 and report them in a longer paper.

4. CONCLUSION

This paper presents five pedagogical challenges in developing activities for parent education. A number of approaches, some have already been tried out in small class section in primary schools, some are under planning and to be launched in large class sections (100 parent-child pairs) to be offered in the coming summer. It is hoped that these activities can support the parent education in CoolThink@JC (“Computational Thinking Education,” n.d.).

5. REFERENCES

- “About Scratch Day” (n.d.). Retrieved March 8, 2017, from <https://day.scratch.mit.edu/>
- “Computational Thinking Education” (n.d.). Retrieved March 7, 2017, from <http://www.coolthink.hk/en/ct/>
- Mackay, R. F. (2013). *Playing to learn: can gaming transform education?*. Graduate School of Education, Stanford University. Retrieved from <https://ed.stanford.edu/news/playing-learn-can-gaming-transform-education>
- “The Hour of Code” (n.d.). Retrieved March 8, 2017, from <https://hourofcode.com>
- “Unplugged Activities in Code.org” (n.d.). Retrieved March 8, 2017, from <https://code.org/curriculum/unplugged>
- “Unplugged Activities in Teaching London Computing” (n.d.). Retrieved March 8, 2017, from <https://teachinglondoncomputing.org/resources/in-spiring-unplugged-classroom-activities/>
- Wing, J. M. (2006). Computation thinking. *Communications of the ACM*, 49, 33-35. doi: 0001-0782/06/0300

Computational Thinking and Teacher Development

Teaching Computational Thinking with Electronic Textiles:

High School Teachers' Contextualizing Strategies in *Exploring Computer Science*

Deborah A. FIELDS^{1*}, Debora LUI², Yasmin B. KAFAI²

¹Utah State University

²University of Pennsylvania

deborah.fields@usu.edu, deblui@upenn.edu, kafai@upenn.edu

ABSTRACT

Understanding how teachers promote students' computational thinking in computer science classes addresses a critical need. We report on how high school teachers implemented a 30-40 hour electronic textile unit in which students designed different wearables with the LilyPad Arduino as part of the *Exploring Computer Science* curriculum in two classrooms. Our analysis focused on how teachers brought out computational thinking through students' interactions and projects in three key areas: strategic problem solving, iteration, and interfacing between abstract and tangible computation. In the discussion, we address what we learned about teachers' pedagogical content knowledge to make computational thinking tangible to students.

KEYWORDS

Electronic textiles, computational thinking, computer science education, teacher practices, pedagogical content knowledge.

1. INTRODUCTION

The introduction of computational thinking into the K-12 curriculum has become a global effort. Computational thinking (CT) was defined by Wing (2006) as a way of approaching and conceptualizing problems, which draws upon concepts fundamental to computer science such as abstraction, recursion, or algorithms. Early work in this area primarily focused on defining computational thinking, specifically its cognitive and educational implications as well as highlighting existing contexts for teaching computational thinking (e.g., NRC, 2011). While much subsequent work has focused on the development of different environments and tools for CT, as well as curricular initiatives in the K-12 environment, there is growing need for more empirical work situated in actual classroom environments (Grover & Pea, 2013).

One glaring absence from these efforts is a lack of understanding of exactly *how* schoolteachers can incorporate CT into their existing classrooms (Barr & Stephenson, 2011). Thus far, most studies focused on CT tools and environments had researchers themselves implement projects or were situated in out-of-school contexts where youth voluntarily engaged on topics of their own choosing (e.g., Grover, Pea & Cooper, 2015; Denner, Werner & Ortiz, 2012). While these studies provided important insights about the feasibility of engaging students in CT, they could not address the

critical issue of how computer science teachers, dealing with large class sizes and curricular restrictions, can integrate CT into their classroom activities—connecting technology, content and pedagogy (Mishra & Kohler, 2006).

In this paper, we focus on the implementation of a 6-8 week (30-40 hour) electronic textiles unit within two high school classrooms situated within the *Exploring Computer Science* (ECS) curriculum (Goode, Margolis & Chapman, 2014). Electronic textiles (e-textiles), or fabric-based computing, incorporate basic electronics such as microcontrollers, actuators and sensors with textiles, conductive thread and similar “soft” materials (see Buechley, Peppler, Eisenberg, & Kafai, 2013). Two experienced ECS teachers from two separate urban schools implemented the curriculum in their classrooms during the final two months of the school year. Two researchers observed the daily implementation of the curriculum, documenting classroom activities and interactions in extensive field notes, video recordings and photos of students' work. The following research question guided our analysis “What kind of teaching strategies did the two teachers employ in supporting and situating computational thinking within the e-textile unit?” Our discussion focuses on the teachers' contextualization and personalization strategies to make computational thinking accessible in students' work.

2. BACKGROUND

While computational thinking is related to the creation of code, it is important to note how understanding programming is not the same thing as CT itself (Wing, 2006). As Wing (2006) states, “Thinking like a computer scientist means more than being able to program a computer.” In other words, it involves particular kinds of approaches to problems that exist in the world (not just on the screen). In terms of teaching programming, considerable research has focused on *content*, drawing attention to the ways in which particular programming concepts and practices, such as loops and debugging, can be taught within classrooms (e.g., Soloway & Spohrer, 1991). Here, research is driven by the need to recognize what concepts and practices are difficult to learn and how to scaffold students' learning. More recent efforts have focused on *context*, highlighting different kinds of project spaces in which learning programming can occur, whether in game design, robotics, creating apps, or constructing wearables (e.g., Kafai & Burke, 2014). Here, efforts are driven by the recognition that teaching and learning

programming needs to be contextualized in ways that engage students' existing interests.

Because teaching computational thinking is newer, research has generally focused more broadly on conceptual or hypothetical contexts (Grover & Pea, 2013). One area of work defines the actual nature of computational thinking (e.g., NRC 2011) in terms of cognition and its relationship to existing disciplines (e.g., mathematics and engineering). Another area of work has focused on developing CT-focused curricula for K-12 contexts (e.g., the AP Computer Science Principles course, Exploring Computer Science, Bootstrap). Finally, another significant area of work in this area relates back to questions of designing contexts. As with teaching programming, researchers have identified the importance of developing particular environments and tools for supporting CT, often overlapping with those that teach programming (e.g., graphical programming interfaces, digital and tangible computational construction kits). While all this work tends to focus on the potential or need to bring CT into education, what is missing are studies of how teachers actually implement these ideas in their classrooms and the particular ways in which technology content and pedagogy intersect that has been described as *technological pedagogical content knowledge* (Mishra & Kohler, 2006).

The research on actual computer science teaching, in particular with a focus on CT, has focused for the most part on pre-service teachers and ways to integrate CT in classrooms (e.g., Yadav, Mayfield, Zhou, Hambrusch & Korb, 2014). Case studies have been developed to examine the strategies used by teachers to address CT in their classrooms (Griffin et al, 2016). The area that overlaps most with CT is focused on algorithmic thinking (Ragonis, 2012). Our work contributes to this emerging body of knowledge by examining how experienced computer science teachers teach CT using electronic textiles. Much early research using e-textiles has focused on broadening participation in areas of computing and engineering by reshaping students' perspectives of and interests in those fields (e.g., Buchholz, Shively, Peppler & Wohlwend, 2014; Kafai, Fields, & Searle, 2014). One study (Kafai et al, 2014) identified several CT concepts, practices and perspectives that students learned while making an e-textiles human sensor project—a precursor to one of the projects in the curriculum discussed in this paper.

Using e-textiles affords different opportunities to observe teaching strategies because they (1) integrate CT within engineering (i.e., circuit design) and coding (i.e., software design) and can illustrate how teachers make connections between them; (2) are hybrid nature in nature (i.e., as textual code on the screen and as physical circuits on the textile) and can make visible how teachers navigate between different modalities; and (3) allow for creative expression and aesthetics through personalized projects and can demonstrate how teachers respond to and are supportive of student interest. Focusing on two classrooms from the *Exploring Computer Science* (ECS) program (Goode, Margolis & Chapman, 2014), we examined what

strategies these experienced ECS teachers used in their implementation of the new e-textiles curriculum unit.

3. METHODS

3.1 Context

The Exploring Computer Science (ECS) initiative comprises a one-year introductory computer science curriculum with a two-year professional development sequence. The curriculum consists of six units: Human-Computer Interaction, Problem-Solving, Web Design, Introduction to Programming (Scratch), Computing and Data Analysis, and Robotics (Lego Mindstorms) (Goode & Margolis, 2013). The instructional design of the curriculum adopts inquiry-based teaching practices so that all students are given opportunities to explore and design investigations, think critically and test solutions, and solve real problems. ECS has successfully increased *diversity* to representative rates in Los Angeles and has subsequently scaled nationwide to other large urban districts and regions, now with over 500 teachers nationwide.

Within this successfully implemented, inquiry-based curriculum, we noted an opportunity to bring creative making in the form of e-textiles into computer science classrooms. The curriculum was co-developed by e-textiles and ECS experts to combine best practices of teaching and crafting e-textiles based on a constructionist philosophy alongside ECS principles, style, and writing. The curriculum contains big ideas and recommended lesson plans, with much room for teachers to interpret and bring in their own style. The e-textiles unit consists of six projects, each increasing in difficulty and creative freedom, that introduced concepts and skills including conductive sewing and sensor design; simple, parallel, and computational circuits (independently programmable); programming sequences, loops, conditionals, and Boolean logic; and data from various inputs (switches and sensors). As an example, the fifth project of the curriculum is a “banner” project in which students worked in pairs to create a letter in a classroom banner. Each letter includes two switches used to generate four lighting pattern effects with 4-5 individually programmable LEDs. The final project consists of a personalized textile artifact that incorporates a handmade human sensor created from two aluminum foil conductive patches that when squeezed generate a range of data. In this study, students used this data to program different lighting effects so that the lights changed based on how hard a user squeezed their project. Student artifacts included stuffed animals, paper cranes, and wearable shirts or hoodies, all augmented with the sensors and actuators.

3.2 Data Collection & Analysis

In Spring 2016 two high school teachers, each with 8-12 years of computer science classroom teaching experience, piloted the unit in their ECS classes with 24 and 35 students in two urban schools in a major city in the western United States. During the implementation, two researchers visited the classroom four days a week, documenting teaching with detailed field notes and pictures of student work supplemented by pre- and post-interviews with the teachers, video recordings, and daily reflections by the teachers.

For this paper, we conducted analysis of field notes taken from the two classrooms. Out of a total of 25-27 days of data for each classroom, we selected a set of six field notes for each classroom. These focused on key lessons in the curriculum, including: learning to sew using a stitchcard (Project 2), working with a preprogrammed microcontroller (Project 3), collaboratively programming a microcontroller as part of a classroom banner (Project 5); creating and programming an individual human sensor project (Project 6). Based on the existing framework for the AP Computer Science Principles course (CollegeBoard, 2016), we developed a preliminary coding scheme looking at how teachers incorporated key computational thinking principles into their classrooms. Two researchers (Authors 1 and 2) entered into an iterative cycle of coding the field notes, comparing their analysis, and refining the coding scheme. Throughout three cycles, we began to identify what was unique in e-textiles that related to the core content and practices identified in AP CS Principles.

4. FINDINGS

Within the e-textiles unit, we saw evidence of many ways in which teachers brought out computational thinking through students' interactions and projects. Three of the most prominent aspects include: 1) strategic problem solving, 2) iteration, and 3) interfacing between abstract and tangible computation. In order to clarify how the teachers translated and implemented the curriculum within their classrooms, we highlight the intersections between how the curriculum promoted the particular CT element and then how teachers expanded upon these elements within their classrooms.

4.1. Strategic Problem Solving

One area of rich computational thinking was in strategic problem solving, the deconstruction of problems into a sequence of steps or "rules" with which to approach problems, sometimes referred to as algorithms (Ragonis, 2014). We found that teachers developed means of strengthening strategic problem solving *across* the intersecting domains of code, circuitry, and craft in e-textiles. In regard to coding, one of the teachers enabled students to develop multiple ways of structuring code to solve a problem. By not dictating to students a single way to approach a problem, students themselves came up with different approaches. For instance in the collaborative banner project that required students to use two switches as inputs to create four lighting patterns, student groups used either nested conditionals or "and" statements as a means for solving that challenge, the latter of which ("and" statements) the teacher had not planned on introducing. By subsequently highlighting the two main approaches used by students, the teacher supported their formalization as problem solving strategies that could be (and were) applied to other problems as well.

In terms of circuitry, teachers helped students develop and apply rules for connecting components in a functional circuit, for instance considering polarity ("plus to plus, minus to minus"), using a common ground line to connect negative pins, and making circuitry efficient. Related to the latter, one student drew a parallel between the

"traveling salesman" problem that he had encountered earlier in the ECS curriculum with designing efficient circuitry in e-textiles: "you have limited amount... of string [conductive thread] so you have to kind of connect all your lights, get it how you want in the cheapest way possible." This approach to finding the simplest circuitry path without creating a short circuit was a type of strategic problem solving that shows computational thinking off the computer screen.

Likewise, with crafting, students learned other sorts of algorithms for the physical construction of the project. For instance, one teacher developed a simple means of expressing a way to tackle the problem of sewing a two-dimensional circuit on a three-dimensional space (i.e., a stuffed animal): "fileting" the animal. This involved sewing two identical cutouts of the stuffed animal (a front and a back) together on part of one side—a "filet"—in order to sew the circuitry before connecting the edges and stuffing the creation. While the approach was not new (outlined in Buechley, Qiu, & de Boer, 2013), the teacher named the approach (a "filet") and presented it in such a way that students formalized it as a strategy for approaching the creation of light-up stuffed animals. In this type of situation, when the teacher models rules for problem solving and creates an environment where students can contribute their own problem solving approaches, it may be argued that students not only learned how to *create algorithms*, but also practiced how to *approach problems algorithmically*, or develop an *"algorithmic stance"* toward problem solving.

4.2. Iteration

Within the world of software design, iteration—or the process of continual repetition and revision—is essential for the completion and refinement of different algorithms and programs. Iterative design, or the cycle of prototyping, testing, and revision, is also key to engineering production. Within the e-textiles unit, students were engaged with iteration on both levels. Within the unit, the production of individualized projects drove the process of iterative design, that is, fixing and resolving mistakes throughout design, visible in changed circuit diagrams, several different versions of code, and constantly refined projects.

Beyond the curriculum, teachers actively incorporated this ethos of iteration into the classroom through several strategies. While the curriculum outlined the basic guidelines for projects, students were encouraged by teachers to come up with original designs. As such, problems and issues that individual students faced often did not fit into an existing template of construction. Both teachers therefore actively addressed the process of dealing with mistakes and making revisions throughout the unit in several ways. First, they created supports for dealing with mistakes, for example, providing tools explicitly designed for iteration (seam rippers), and sharing concrete 'tips and tricks' of e-textiles construction (e.g., how they dealt with broken thread). Second, teachers explicitly valued the process of iteration, regularly sharing students' unfinished projects, including their mistakes, to highlight something the creator had learned that was of

value to the class as a whole. In doing so, they stressed the importance iteration and refinement alongside the final product.

Teachers also positioned themselves as collaborators (rather than authorities) within this environment of iteration. Not only did they share the e-textiles projects that they had personally made, but they also modeled the many mistakes and revisions that they faced within their own personal journeys of creation. Additionally, because of the variety of unique issues of construction on student projects, teachers worked alongside students to troubleshoot in the moment. One teacher required students to work in pairs to check each other's circuitry diagrams and code before they could move on to later tasks. This supported peer troubleshooting and revision, allowing the teacher to come over only when both students were befuddled. From this perspective, iteration appeared both in direct project work and in the teachers' pedagogical approach: through shared problem solving, modeling projects at various stages, and building a shared knowledge base about e-textiles by compiling tips and tricks learned through mistakes.

4.3 Interfacing

As opposed to many computer science classes where programming takes place mostly on the screen, one unique and prominent affordance of using e-textiles to teach CT is how it fosters a need for students to become more familiar with the intersections between *abstract* and *tangible* forms of computation. Specifically, this results from the high level of interaction that inherently occurs when creating an e-textiles artifact: hand-crafting a functional circuit that can be controlled via purposeful planning and development of code. Through e-textiles, teachers were given a highly visible and potent context through which to teach important CT concepts including information abstraction and manipulating data. Teachers' encouragement of this intersection is discussed below, followed by a discussion of how they used this to contextualize understanding of information abstraction and manipulating data.

E-textiles is a unique educational computing context because it involves the creation of codable circuits and sensors, artifacts which are situated at the *intersection of software and hardware*. Teachers actively worked to make this connection clear for students, supporting students' fluency in moving back and forth between the realms of designing, constructing, and troubleshooting tangible circuits, while also planning, writing and troubleshooting the code to control these circuits. One specific strategy was to create conditions where student experiences with programming and software were immediately (and always) correlated with some piece of hardware; that is, no one was ever expected to code purely for the screen without a physical LilyPad Arduino circuit output. For example, even during lessons primarily focused on coding, both teachers actively pointed out the relationship between the onscreen lines of code with physical components (e.g. pins on the LilyPad for inputs and outputs) and behaviors (e.g. this line of code makes that LED turn on). Teachers further strengthened this

connection for students during the creation of their individual codeable circuit projects (specifically the collaborative banner and human sensor projects). Mostly, this occurred through individual design consultations; for instance one teacher worked with students to consider how the programmability of their project was influenced by the size of their sensor patches. As a result of this kind of activity, students not only became familiar with working on both hardware and software, but also started to develop a capacity of moving back and forth fluidly between these two domains.

Information abstraction is universally acknowledged to be an important type of computational thinking, whether dealing with binary numbers or subroutines and procedures (Grover & Pea, 2013). In terms of e-textiles, this mostly involves taking the real world phenomenon related to an artifact (e.g., touch levels, placement of LEDs) and converting it to digital representations on the computer (e.g., variables, statements). Within the curriculum, students had numerous opportunities to practice formal information abstraction such as declaring variables and incorporating Boolean logic into their programs.

However, teachers' contributions toward students' understanding for information abstraction resulted most strongly from their efforts to contextualize this concept within the process of e-textiles construction. In other words, because e-textiles creates a highly visible context for moving in between real world phenomena (e.g., physical touch, switches) and computational inputs (e.g., numbers, ranges, conditions), both teachers leveraged this as a natural context in which to instill an understanding of information abstraction. For example, one teacher worked to illustrate how variable declaration allowed the computer to make use of real world data, stating, "So depending on how hard you touch the foil, it [the variable we created] will store it there and we can use it for our If-Else Statements". Teachers furthered the back-and-forth nature of e-textiles construction through the process of troubleshooting students' individual projects. Both teachers constantly went through a cycle of testing, where students would be asked to "read" their code both in the physical format (what it actually did) and in the abstract, onscreen format (the code). While e-textiles provided a format for this work, the teachers' efforts made the move between the tangible and the abstract, the physical and the digital (e.g., onscreen) explicit for students, enabling them to develop more experience with information abstraction first hand.

Teachers similarly helped students gain experience in *reading and manipulating computational data*—that is, learning how to make sense of information gathered through computational inputs or outputs and leveraging these for a computational purposes. This skill was cultivated most actively through the creation of the human sensor project. In this project, students created personal artifacts (e.g., a pink bear, a cosmically-themed top hat, a large origami crane) that contained handmade touch sensors (aluminum patches sewn onto their projects) that were programmed to read the conductivity levels of a

person touching both patches. Teachers worked to give students many visceral opportunities to deal with real world data within the context of their human sensor project. One teacher, for instance, described the process of testing the conductivity of himself and his wife, describing how they could each get different ranges based on their different sizes. Based on this, he then required students to test out the conductivity ranges of at least four different people within the class and use these as baselines to develop more universal ranges in developing their code. This brought personalization to interfacing between digitally written data and physical touch, especially as the teacher modeled how he tested the sensor with his wife and as students tested their sensors with each other. It is impossible to tell how directly this affected students overall, but at least one student described taking his project home so he could test out his dad's conductivity range, even going so far as to use his hands while sleeping. Notably, reading data from a sensor and breaking it down into usable ranges that could be expressed mathematically (i.e., "sensorvalue < 1000 && sensorvalue > 750") was a particular challenge for students, evidenced in a related question on a post-test. In the next year we plan to bring more curricular scaffolding to this task while supporting the efforts of teachers to personalize this aspect of computational thinking for deeper learning.

5. DISCUSSION

Our paper contributes to the emerging body of research on teaching practices of computational thinking that are not just focused on coding but extend into other domains, such as physical crafting and electronics. In our analysis we focused on key aspects of computational thinking (CT)—strategic problem solving, iteration, and interfacing between abstract and tangible aspects of computing—that teachers addressed within the new electronic textiles curriculum. In the following sections, we further discuss aspects of teachers' pedagogy regarding computational thinking in their classroom activities.

Across the analyses of the various teaching strategies in their classrooms, we noted the element of personalization as a critical aspect of teaching computational thinking in this context. First, based on earlier research on the importance of aesthetics in learning with e-textiles (Kafai et al, 2014), the curriculum intentionally foregrounded the personal nature of each student project, resulting in different implementations by each student. This aspect of personalization is rarely discussed in contexts of pedagogical content knowledge but can pose challenges in teaching because of the vastly different problems that arise in individual student projects. Teachers need to be flexible in taking advantage of them to promote deeper learning. Second, in this study the teachers took personalization beyond individualized projects and worked it into their classroom practice. For instance, teachers modeled their own projects, mistakes, and design processes in ways that validated "process" alongside "product", highlighting iteration as an important aspect of computational thinking. This also provided a backdrop that facilitated sharing mistakes and processes that could help the entire class learn. It is less risky to share the mistakes you have made when your teachers have already shared their own

mistakes. Further, by sharing their projects and discussing multiple users of their projects (including one's spouse) the teachers brought out the broader usability of projects: students' projects (as their teachers' projects) could have relevance outside the classroom. These features are important because personalization has been shown to support student interest in CS classrooms (Griffin et al., 2016). Furthermore, these practices highlight strategies that can make teaching more culturally relevant, providing one means for equity to become a key part of classroom instruction with computational thinking (Goode, Margolis & Chapman, 2014).

Further, we want to consider teaching strategies that focus not just on content or projects but on the larger classroom working environment, within which the electronic textile ECS unit was implemented. The two teachers we studied engaged with vastly larger numbers of students (24 and 35) and within more restricted time constraints (the class period) and spaces (i.e., a classroom where material had to be put away everyday) than encountered in most other e-textile implementations, especially in afterschool, summer camp, or weekend workshops. The teachers used strategies such as modeling (both their own and students' in-progress projects), peer support (checking each other's work), and tips and tricks to support students' different creations. Other strategies which this paper does not have room to address deserve further attention such as the organization and management of materials and time, and validation of student work during class discussion.

By themselves the strategies discussed above are nothing new in and of themselves, having been found in much exemplary science and mathematics teaching (see Ball, Thames & Phelps, 2008). However, it is the application of these teaching strategies to computational thinking that presents a unique and promising approach to support students in this emerging field of pedagogy. These strategies are similar to what Mishra and Kohler (2006) described as technological pedagogical content knowledge (TPCK) or the unique knowledge that teachers need to develop in order to embed technology in their instructional practice to support student learning. However, the original description of technological pedagogical content knowledge focused on how to integrate different technologies such as video or games into the classroom, whereas in the ECS context, teachers focused on integrating computational thinking with content. To highlight this distinction, we should reframe TPCK as "computational pedagogical content knowledge" or (CPCK), acknowledging the specialized content knowledge (Ball, Thames & Phelps, 2008) that is emerging in relation to computational thinking in K12 education. Much more research is needed in documenting CPCK teaching practice within other curricular efforts to develop best practices that can be shared and developed.

In this paper we considered how experienced ECS classroom teachers connected computational thinking in the context of learning with e-textiles. While the unit was designed with certain goals in mind, largely related to programming, we sought to apply the AP CS Principles guidelines to identify where and how aspects of

computational thinking such as iteration, abstraction or problem solving were supported in teachers' classroom practice. We found particular affordances for learning computational thinking that e-textiles may be uniquely situated to promote, especially in regard to tangibility and personalization. Because it was the first implementation of this new ECS curriculum unit and one of the first e-textiles projects where the teachers were the main leaders in the classroom, our data collection focused primarily on the teacher modeling, leading, and discussing with students. In future iterations we plan to look more closely at how teaching strategies intersect with student learning to understand more about the depth and breadth of learning across students in each classroom and to evaluate whether equity is being reached in terms of rigorous learning.

6. REFERENCES

- Ball, D., Thames M. H., & Phelps, G. (2008). Content knowledge for teaching: What makes it special? *Journal of Teacher Education*, 59(5), 389-407.
- Barr, V. & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads* 2, 1, 48-54.
- Buchholz, B., Shively, K., Peppler, K. & Wohlwend, K. (2014). Hands on, hands off: Gendered access in sewing and electronics practices. *Mind, Culture, and Activity*, 21(4) 1-20
- Buechley, L., Peppler, K., Eisenberg, M. & Kafai, Y. (Eds) (2013). *Textile Messages: Dispatches from the World of E-Textiles and Education*. New York: Peter Lang.
- Buechley, L., Qiu, K., & de Boer, S. (2013). *Sew Electric: A Collection of DIY Projects that Combine Fabric, Electronics, and Sewing*. HLT Press: Cambridge, MA.
- CollegeBoard (2016). *AP Computer Science Principles: Course and Exam Description Effective Fall 2016*. CollegeBoard: New York, NY. Retrieved from: <https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf>
- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58, 240-249.
- Griffin, J, Pirman, T. & Gray, B. (2016). Two teachers, two perspectives on CS principles. In *Proceedings of SIGCSE'16* (pp. 461-466). New York, NY: ACM.
- Goode, J., & Margolis, J. (2011). Exploring Computer Science: A Case Study of School Reform. *ACM Transactions on Computing Education*, 11(2), 12.
- Goode, J., Margolis, J. & Chapman, G. (2014). Curriculum is not enough: The educational theory and research foundation of the Exploring Computer Science professional development model. In *Proceedings of SIGCSE'14* (pp. 493-498). New York, NY: ACM.
- Grover, S. & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(2),: 59-69.
- Grover, S., Pea, R. & Cooper, S. (2015). Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students. *Computer Science Education*, 25(2), 199-237.
- Kafai, Y. B. & Burke, Q. (2014). *Connected Code: Why Children Need to learn programming*. Cambridge, MA: MIT Press.
- Kafai, Y. B., Fields, D. A., & Searle, K. A., (2014). Electronic Textiles as Disruptive Designs: Supporting and Challenging Maker Activities in Schools. *Harvard Educational Review*, 84(4), 532-556.
- Kafai, Y. B., Lee, E., Searle, K. S., Fields, D. A., Kaplan, E., & Lui, D. (2014). A crafts-oriented approach to computing in high school. *ACM Transactions of Computing Education*, 14(1). 1-20.
- Mishra, P. & Kohler, M. J.(2006). Technological pedagogical content knowledge: A new framework for teacher knowledge. *Teachers College Record*, 108(6), 1017-1054.
- National Research Council (2011). *Report of a Workshop on Pedagogical Aspects of Computational Thinking*. Washington, DC: National Academy Press.
- Ragonis, N. (2012). Integrating the teaching of algorithmic patterns into computer science teacher preparation programs. In *Proceedings of ITiCSE'12* (pp. 339-344). New York, NY: ACM.
- Wing, J (2006). Computational thinking. *Communications of the ACM*, 49, 33-35.
- Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education*, 14(1), 1-1.

Application of the Four Phases of Computational Thinking and Integration of Blocky Programming in a Sixth-Grade Mathematics Course

Ting-chia HSU^{1*}, Hsin-chung HU²

¹ National Taiwan Normal University

² Er-Cheng Elementary School

ckhsu@ntnu.edu.tw, az8312@tmail.ilc.edu.tw

ABSTRACT

This study put four steps, problem decomposition, pattern recognition, abstraction, and algorithm, into practice by integrating the blocky programming language, Scratch, into a mathematics course. The teacher guided the sixth-graders to apply the four steps of computational thinking to writing a blocky program to solve daily-life equality axiom mathematics problems. The results showed that the method was beneficial for promoting the learning effectiveness of mathematics, and also found that there was a significantly positive correlation between the performance of blocky programming and the mathematics post-test. There was no significant correlation between creative tendency and self-efficacy after the experiment. Self-efficacy had a positive correlation to learning motivation both before and after the experiment.

KEYWORDS

Scratch, Computational Thinking, Mathematics, Self-efficacy, Learning motivation

1. INTRODUCTION

Computational thinking has a broad definition which refers to the thoughts and plans or comprehension process when students are confronted with any uncertain factor or new issue. Computational thinking also has a narrower definition which means the basic concepts and processes used for solving problems in the computer science domain. The term was officially proposed in 2006 [12], and was later divided into four phases [13], described as follows.

The first phase of the computational thinking process is to decompose the problem so that one problem can be analyzed and divided into several smaller questions. This is called the “problem decomposition” phase. Then, the second phase is to identify the patterns in the data representation or data structure. In other words, if the students observe any repeated presentation of data or method, they can identify their similarities, regularities, or commonalities. Therefore, the students do not need to spend time on the repeated work when they write the problem. The third phase is to generalize or abstract the principles or factors to become a formula or the corresponding programming language rules. The students have to try to model the patterns they found in the previous step. After testing, the students identify the key factor presenting the model in this step. Finally, they design the algorithm in the fourth phase, ensuring that they include all the steps for solving the problem systematically.

Although computational thinking is not equal to programming, the blocky programming languages, such as Scratch, mBlock and so on, are good tools for developing the capabilities of students’ computational thinking. The current study not only employed Scratch to learn computational thinking, but also used it to implement a problem the students confronted in their Mathematics course. One main purpose of these programming languages is to solve computation problems. Scratch is a visual programming tool and is suitable to be used in different subjects such as games, science, music and so on [5, 6].

Scratch has been introduced to young students from eight to eighteen years old [4], and they have been found to be highly motivated to write programs. Another study found that fifth and sixth graders perceived usefulness, high motivation, and positive attitudes toward Scratch [4, 5, 9]. Ke (2014) applied Scratch for secondary school students to design mathematics games, and found that the integration of the blocky programming and Mathematics game design could promote the potential of the students to learn Mathematics, and made the students have significantly more positive attitudes toward the development of Mathematics [2]. Furthermore, this method was beneficial for activating the students’ reflection on their daily-life mathematical experiences. The mathematics concepts and blocky programming were integrated when the students solved the problems or created the games. They not only took part in achieving the learning target of mathematics, but also carried out computational thinking, and transferred the reasoning process into an abstract program. It has been found that using Scratch in computer science can promote the cognitive level and self-efficacy of the students, but it does not result in high learning anxiety, and the students spend less time learning and creating a new program [1]. The blocky programming (e.g., Scratch, App Inventor) did retain the learning motivation and interests of the students [7]. The scholars employed another blocky programming environment, Code Club, into elementary schools, and found that it could motivate the creative digital design of the young students [10].

Therefore, the current study also integrated the mathematics course with the blocky programming software, Scratch, and applied the four phases of computational thinking to solve mathematics problems. The purpose of the study was to explore the correlations between self-efficacy and learning motivation, and between self-efficacy and creative tendency. From the

results, the critical factor correlated with self-efficacy could be identified when the students were involved in the proposed treatments. In addition, this study also aimed to confirm whether the students made significant progress in Mathematics and in problem solving by using the blocky programming. Therefore, the research problems are listed as follows:

- (1) After the treatment, was the students' learning effectiveness of mathematics significantly promoted?
- (2) Was there a significant correlation between the performance of blocky programming with the learning effectiveness of mathematics?
- (3) Was there a significant correlation between self-efficacy with creative tendency and learning motivation before and after the treatments?

2. METHOD

2.1. Participants

The subjects included one class of sixth graders of an elementary school in Taiwan. A total of 20 students participated in the study. They were taught by the same instructor who had taught that mathematics course and Scratch for more than ten years. The average age of the students was 12.

2.2. Measuring tools

The research tools in this study included the pre-test and post-test of the mathematics learning achievements, the post-test of Scratch Programming implementation, and the questionnaire for measuring the students' learning motivation, creative tendency, and self-efficacy.

The test sheets of mathematics were developed by two experienced teachers. The pre-test consisted of 10 calculation questions about the prior knowledge of learning the course unit "equality axiom," with a perfect score of 100. The post-test consisted of 10 calculation questions for assessing the students' knowledge in the equality axiom unit, with a perfect score of 100. The items in the pre-test are different from the items in the post-test, but they had the same difficulty degree.

In terms of the post-test of programming performance, there were totally five situated problems for the students to solve using the blocky programming with the four phases of computational thinking. Each programming problem was scored as 20 points, including 5 points for assessing whether the students employed proper blocks, 5 points for checking the usage of variances, 5 points for evaluating the formula transferred from the meaning of the problem by the students in the program, and 5 points for confirming if the output was correct or not. Consequently, five programming problems were worth a total of 100 points.

The questionnaire of learning motivation was modified from the measure published by Hwang, Yang, and Wang (2013) [3]. It consisted of seven items (e.g., "It is important for me to learn what is being taught in this class") with a 5-point rating scheme. The Cronbach's alpha value of the questionnaire was 0.823.

The self-efficacy questionnaire originates from the questionnaire developed by Pintrich, Smith, Garcia and McKeachie (1991) [8]. It consists of 8 items with a 5-point Likert rating scheme. The Cronbach's alpha value was 0.894. The Creativity Assessment Packet (CAP) was revised from Williams (1991) [11], including the scales of imagination, curiosity, and so on.

2.3. Experimental procedure

Before the experiment, the students were given time to get used to the blocky programming environment. Figure 1 shows the flow chart of the experiment. Each period in the mathematics class is 40 minutes in elementary schools in Taiwan. At the beginning, the instructor spent eight weeks (i.e., once period a week, and totally 8 periods) teaching the students to become familiar with the blocky programming environment.

Before the learning activity of applying the four phases of computational thinking systematically, the students completed the Creativity Assessment Packet measure, took the pre-test, and completed the learning motivation and self-efficacy questionnaires.

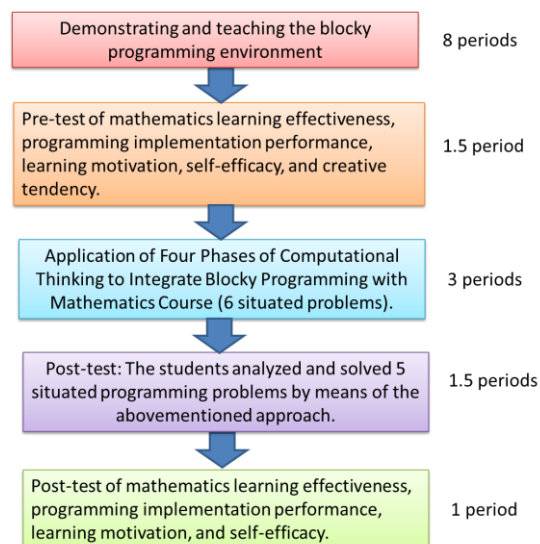


Figure 1. Experimental procedure

Thereafter, the study spent three weeks (i.e., once period a week, and totally three periods) on the enhancement of applying the four phases of computational thinking and integration of blocky programming in a sixth-grade Mathematics course. In other words, during the learning activity, the teacher guided the students in how to employ the four phases of computational thinking for analyzing the situated problems in the mathematics course, and finally how to write blocky programs for calculating the results of each problem. The teacher explained how to employ the four phases of computational thinking, step by step. From the first phase, the students tried to analyze and decompose the situated problem which the teacher designed for demonstration. Secondly, the students were guided to find out whether there was a pattern or similar situation based on the results of the analysis, that is, pattern recognition. Thirdly, they had to conclude or transfer the analysis to a formula or programming presentation. Finally, they found the limited problem

solving steps, or algorithm, so that the program could be written based on the steps.

The students practiced this method six times, each time taking half of a period. Therefore, there were totally six situated examples implemented during the three periods of the mathematics course. At the same time, the students learned mathematics from solving the blocky programming problems through the four computational thinking phases.



Figure 2. Example of an elementary school student's solution

After the learning activity, there were totally five programming problems for evaluating the blocky programming performance of the students, with the four computational thinking phases involved in both the blocky programming and the mathematics problems. The test took 1.5 periods.

Finally, they also spent one period on the post-test of the pen-and-paper-based mathematics test for measuring their learning achievements. They also answered the motivation and self-efficacy questionnaires so as to identify whether any changes had occurred in their learning motivation and self-efficacy after the different learning method was employed for learning mathematics. There were totally 15 periods spent on the experiment, which lasted for a total of around three-fourth semester (i.e., 15 weeks). The experimental treatment after pretest was five weeks.

2.4. Data analysis

The pre- and post-test were compared via a paired-sample t-test. Therefore, whether the students made progress or not was assessed. The same analysis method was also employed for comparing the students' learning motivation and self-efficacy before and after the learning activities.

Their blocky programming performance was also assessed by the teacher. Correlation analysis was performed to identify the relationship between the students' blocky programming performance and their post-test results.

Correlation analysis was utilized for checking the correlation among the students' learning motivation, self-efficacy and creative tendency when they were learning mathematics by means of conventional instruction. Moreover, the same method was used for checking the correlation among the learning motivation, self-efficacy and creative tendency after the students learned

mathematics from the Application of the Four Phases of Computational Thinking to Integrate Blocky Programming into the Mathematics Course.

3. Results

3.1. The paired sample t-test on the pre-test and post-test in mathematics

The research design hypothesized that the students would make progress in the learning objectives of the mathematics unit. Therefore, a paired-sample t-test was performed on the pre-test and post-test in the mathematics unit.

The students did not use conventional instruction to learn mathematics; rather, the four phases of computational thinking were applied to integrate blocky programming into the mathematics course. Table 1 shows that this approach did indeed contribute to the learning effectiveness of the students. They made significant progress in the mathematics unit of equality axiom after the experimental treatment ($t=2.72^*$; $p<0.05$).

Table 1. paired sample t-test on the pre- and post-test

	N	Mean	SD	t
Post-test	20	86.35	17.60	2.72*
Pre-test	20	80.75	16.66	

* $p<0.05$

3.2. The correlation between the performance of blocky programming and the mathematics post-test

In this study, we attempted to verify the correlation between the performance of blocky programming and the mathematics post-test. The results showed that they did have a significantly positive correlation (Pearson=0.673**, $p<0.01$), as shown in Table 2. When the students had better performance on applying the four phases of computational thinking to write a blocky program which solved the situated problems of the equality axiom mathematics unit, they also had better learning outcomes on the post-test of the conventional pen-and-paper-based mathematics test.

3.3. The correlation between students' self-efficacy and their creative tendency and learning motivation

The self-efficacy of the students applying the four phases of computational thinking to integrate blocky programming into the mathematics course was significantly correlated with their learning motivation (Spearman correlation value=0.623**, $p<0.01$), but was not noticeably related to their creative tendency (Spearman correlation value=0.232; $p>0.05$), shown as Table 2.

Table 2. Correlation between self-efficacy and creative tendency and learning motivation (N=20)

Spearman correlation coefficient	Motivation	Self-efficacy	Creative tendency
Motivation	1	0.623**	0.189
Self-efficacy	0.623**	1	0.232
Creative tendency	0.189	0.232	1

** $p<0.01$

4. DISCUSSION AND CONCLUSIONS

This study not only put the four phases of computational thinking into practice, but also applied it to solve mathematics problems with Scratch, one of the blocky programming languages. The results indicate that the implementation of the programming was effective; in addition, the students' learning effectiveness, and their results in the mathematics concepts post-test both improved remarkably in comparison with the pre-test of the same mathematics unit.

The implementation of the programming had a significantly positive correlation with the learning effectiveness of mathematics, implying that the students who had better Scratch scores outperformed the other students in the mathematics concepts post-test. The students' self-efficacy was correlated with their learning motivation, but not with their creative tendency. In other words, the students who had higher learning motivation possessed higher self-efficacy. In future studies, teachers could try to design mathematics games for students to design programs and learn mathematics at the same time, as the teachers in this study only designed daily-life situated mathematics problems related to the mathematics learning unit for the students to apply computational thinking to solve the problems. Future studies could also integrate different subjects to learn computational thinking, programming, and certain subject knowledge (e.g., physics, mathematics) at the same time.

ACKNOWLEDGEMENTS

This study is supported in part by the Ministry of Science and Technology in Taiwan under contract number: MOST 105-2628-S-003-002-MY3. This study was conducted in an elementary school, and the authors are grateful to the YiLan County Government Education Department.

REFERENCES

- [1] Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, 14(4), 25.
- [2] Ke, F. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, 73, 26-39.
- [3] Hwang, G. J., Yang, L. H., & Wang, S. Y. (2013). A concept map-embedded educational computer game for improving students' learning performance in natural science courses, *Computers & Education*, 69, 121-130.
- [4] Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. *ACM*, 40(1), 367-371.
- [5] Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4), 16.
- [6] Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 15(46). Retrieved http://www.um.es/ead/red/46/moreno_robles.pdf on December 12, 2016.
- [7] Nikou, S. A., & Economides, A. A. (2014, April). *Transition in student motivation during a scratch and an app inventor course*. In 2014 IEEE Global Engineering Education Conference (EDUCON) (pp. 1042-1045). IEEE.
- [8] Pintrich, P.R., Smith, D.A.F., Garcia, T., & McKeachie, W.J. (1991). *A manual for the use of the motivated strategies for learning questionnaire (MSLQ)*. MI: National Center for Research to Improve Postsecondary Teaching and Learning. (ERIC Document Reproduction Service No. ED 338122).
- [9] Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, 97, 129-141.
- [10] Smith, N., Sutcliffe, C., & Sandvik, L. (2014). *Code club: bringing programming to UK primary schools through scratch*. In Proceedings of the 45th ACM technical symposium on Computer science education (pp. 517-522). ACM.
- [11] Williams, F. E. (1991). Creativity assessment packet: Test manual. Austin, TX: Pro-Ed.
- [12] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- [13] Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881), 3717-3725.
- [14]

The Design and Evaluation of a Teacher Development Programme in Computational Thinking Education

Siu-cheung KONG 1*, Ming LAI 1, Josh SHELDON 2, Mike TISSENBAUM 2

¹The Education University of Hong Kong

²Massachusetts Institute of Technology

sckong@eduhk.hk, mlai@eduhk.hk, jsheldon@mit.edu, mtissen@mit.edu

ABSTRACT

This article documents the design and evaluation of a teacher development programme in computational thinking (CT) education. The results suggested that after taking a teacher development course (TDC), teachers enhanced their CT content knowledge; however, some teachers still did not have sufficient confidence in teaching CT in their classrooms. Subsequent modification to address this lack of confidence among this cohort of teachers is discussed.

KEYWORDS

teacher professional development, computational thinking education, evaluation, teacher development

1. INTRODUCTION

In her influential article, Wing (2006) convincingly argued that computational thinking (CT) is “a fundamental skill for everyone, not just for computer scientists” (p.33). Every child should possess not only the abilities of reading, writing, and arithmetic, but also the analytical skills involved in CT. CT can be considered as the thought process involved in effectively formulating problems and their solutions through a computational or digital means (Cuny, Snyder, & Wing, 2010). After Wing’s (2006) article, CT has been incorporated into K-12 education around the world (Grover & Pea, 2013; Voogt et al., 2015). Block-based programming environments such as Scratch (Resnick et al., 2009) and App Inventor (Wolber, Abelson, Spertus, & Looney, 2015) are one approach used in K-12 education for facilitating students’ CT development (Lye & Koh, 2014).

While CT education is being emphasized in many different countries (Voogt et al., 2015), an important challenge in its implementation in K-12 education is the shortage of teachers capable in delivering CT education (Menekse, 2015). It was reported that a large number of teachers taking professional development programmes in CT or computer science are new to computer science (Century et al., 2013). Even teachers who majored in computer science in their undergraduate studies may not be familiar with block-based programming environments. Besides the subject content of CT or computer science, the implementation of CT education requires teachers to have the relevant pedagogical content knowledge as well (Saeli, Perrenet, Jochems, & Zwaneveld, 2012). This knowledge involves the understanding of the content of coding, and the pedagogy of delivering the content, which is not an easy task (Grover & Pea, 2013). Therefore, it is important

to offer quality teacher development programmes (Yoon, Anderson, et al. 2016) for building the capacity and confidence of teachers in delivering CT education. This article reports on the design, implementation, and evaluation of a teacher development programme in CT education in the context of the CoolThink@JC initiative (CoolThink).

1.1. Effective Teacher Development Programmes

The literature on teacher professional development has identified several key elements for the capacity building of teachers. First of all, knowing the content knowledge of the subject is not enough for teaching the subject, as a teacher also need to know the most appropriate pedagogy for teaching the content to students, and this kind of knowledge is referred to as pedagogical content knowledge (PCK) (Shulman, 1986). Previous studies suggest that effective professional development for teachers usually involves a sustained period of time, with participants actively engaged, with opportunities to practice and reflect, and situated in a community of practice (CoP) (e.g., Borko, Jacobs, & Koeliner, 2010). CoP is not a stable and short term working session, but an interactive and recursive continuum with complex reactions among multiple factors (Clark & Hollingsworth, 2002). In a CoP, knowledge is situated in the daily experience of community members, and learning is a social process which involves participation and interaction (Lave & Wenger, 1991). A CoP model of teacher development allows participants to discuss, peer assess, and self-reflect on their teaching and learning (Scribner, Cockrell, Cockrell, & Valentine, 1999).

2. THE PROGRAMME

2.1. CoolThink@JC Initiative

The background of this study is a four-year initiative of CoolThink@JC (<http://www.coolthink.hk/en/>) aimed at promoting CT education among primary schools in Hong Kong beginning from 2016. A total of 32 primary schools have been recruited as the Network Schools in this initiative. Among these 32 schools, 12 are Cohort-1 schools, which started their CT education programme in the academic year of 2016/17. The remaining 20 are Cohort-2 schools, will start their CT education programme on year later. Three teachers from each school are selected to participate in the teacher development programme and subsequently teach the CoolThink curriculum.

A 3-level curriculum for Hong Kong grade primary 4 (Level 1) to primary 6 (Level 3) was developed as a

collaboration between The Education University of Hong Kong (EdUHK) and the Massachusetts Institute of Technology (MIT). Another partner in the CoolThink initiative is City University of Hong Kong (CityU), which provides parent education on CT, and in-school co-teaching support by recruiting and training undergraduate students from universities in Hong Kong as teaching assistants. The curriculum aims to foster students' CT concepts, practices, and perspectives (Brennan & Resnick, 2012). Each grade level has 10 units of activities and one or two final project(s) for students to develop CT through programming activities in the environments of Scratch and App Inventor. In addition to the formal curriculum, two sets of co-curricular activities, one on interacting with physical objects through coding, another on solving community problems using computational thinking, were developed by EdUHK for the participation of P5 and P6 students respectively. EdUHK, with the expertise in teacher education, and MIT, with the expertise in the development of programming environments and experience in training teachers, are responsible for designing and implementing the teacher development programme. A total of two TDCs, of 39 hours each, are offered for each cohort of teachers.

2.2. Teacher Development Course 1

TDC 1, mainly designed and delivered by staff from MIT, aims to enrich the content knowledge of participants on CT and to allow them to have some initial thoughts on how to deliver CT education in senior primary schools. The major component of TDC 1 was a 5-day (6 hours per day) intensive training conducted by MIT staff over one week. In addition, a 3-hour pre-MIT session and two post-MIT sessions (each with 3 hours) for consolidation and project presentation respectively were conducted by EdUHK staff. The pre-MIT session aimed to lay a foundation for teachers to get ready for the training afterwards. Schools were paired up starting in this session so that they could provide support and feedback to one another throughout the programme.

In the 5-day training, the teachers were guided through the Level 1 formal curriculum that had been developed by EdUHK and MIT, so that they could be more familiar with the content knowledge to be delivered, and have a basic idea of how to implement CT education in their primary classrooms. Besides coding tasks that had to be finished with the computers, unplugged activities (Bell, Alexander, Freeman, & Grimley, 2009) for deepening their conceptual understanding of CT were also included, which was similar to the design of the formal curriculum. The teachers also engaged in pair programming (Denner, Werner, Campe, & Ortiz, 2014) by working in pairs to finish a mini Scratch and a mini App Inventor project, which were also features of the Level 1 curriculum for primary students.

Throughout the five days, teachers took part in group work both with other teachers from their own school, and with teachers from their partner school, to receive feedback in using MIT App Inventor to design and develop a mobile app to address a teaching or classroom need. To allow sufficient time to design and build the app, the

presentation of the app was scheduled in the second post-MIT session.

The first post-MIT session aimed to review what the teachers had learned in the 5-day training and to prepare the teachers to finish and present their mobile app. In the second post-MIT session, each school's group of three teachers presented their app as a group, and received feedback from classmates, instructors, and guests who were experienced teachers in CT education. TDC 1, from the first 3-hour pre-MIT session, through the intensive MIT-led week, to the two 3-hour post-MIT sessions, lasted for about one month. This article aims to report the evaluation of TDC 1, especially on the enhancement of knowledge and confidence of teachers in teaching CT, and how it affects the subsequent design in TDC2.

2.3. Teacher Development Course 2

While TDC 1 emphasized content knowledge of CT and basic ideas for CT pedagogy, TDC 2 had a greater focus on pedagogy. As effective teacher development requires a sustained period of time, within which the participants can be actively engaged and have the opportunity to practice and reflect (Borko et al., 2010), TDC 2 lasted for a total of 13 weeks, with weekly 3-hour sessions. The teachers had the opportunities to teach the Level 1 curriculum in their schools, and shared their teaching experience with one another for feedback and reflection. The final project in TDC 2 required groups of 3 teachers from each school to design and present a complete unit for CT education, which enabled them to think more deeply about what they had learned in the programme and how to deliver CT education.

TDC 2's content included discussion of appropriate pedagogies for CT education; analysis of video clips taken in primary classes implementing CT; and examination of student created artifacts and discussion on how to use those artifacts to assess student learning. To enable teachers to capably conduct the co-curricular activities, there were sessions on interacting with physical objects through coding as well as solving community problems with CT. Guest instructors, including experienced teachers in CT, school principals, and practitioners in coding education with frontline experience in facilitating students to apply coding knowledge to solve community problems, were invited to share their experiences and deliver the lessons.

A CoP approach was employed in TDC 2. Following the collaborative activities within and between schools in TDC 1, there were opportunities for teachers in the same school to work together to solve learning tasks and to discuss with teachers of the partner schools for feedback. To facilitate discussion among teachers, a WhatsApp group for the whole class, and small WhatsApp groups for each pair of partner schools were created for sharing reflections and ideas on coding knowledge and teaching methods. We anticipated that participants with less experience in coding and CT education could benefit from interaction with more experienced counterparts. Also, all teachers could learn from one another and elaborate their understandings of content knowledge and PCK of CT.

2.4. Participants

Participants in this study were 36 primary school teachers from 12 Cohort-1 schools of CoolThink, and four staff members from CityU who are responsible for the training of teaching assistants. Among the 12 schools, 10 are with Chinese as the medium of instruction while 2 are with English as the medium of instruction. Among the 36 teachers, 25 are male and 11 are female. Their average years of teaching experience are 12.5, and average years of experience teaching the subject of Computer or Information and Communication Technology are 7.9.

2.5. Instrument

To evaluate TDC 1, and see whether any refinement was needed for TDC 2, a survey based on the standard course evaluation instrument in the institution was implemented at the end of TDC 1. The survey has three main parts: 1) Likert-scale questions about the teaching; 2) Likert-scale questions about the course design; and 3) Open-ended questions on the most useful aspects of the course and how the course could be changed to help the participants learn. Each Likert-scale question is based on a 4-point scale.

3. RESULTS

The evaluation results related to the teaching of the course and the course itself are as presented in Tables 1 and 2 respectively. It can be seen that in general, the participants were satisfied with the teaching of the course, as the scores were above 3 on a 4-point scale as indicated in Table 1. The participants particularly agreed that the instructors encouraged exchange of ideas among participants in their learning (mean=3.34). They also agreed that the overall teaching was of high quality (3.13).

For the evaluation about the course (Table 2), while most participants regarded the course enhanced their knowledge and skills in developing block-based programs with Scratch (3.21) and App Inventor (3.16), some of them indicated that they did not have sufficient confidence in applying the knowledge in their teaching (2.87) and to equip their students with CT capabilities (2.79).

Table 1. Evaluation on the teaching of the course.

		Mean	SD
Q1	Delivering the course in an organized way.	3.03	0.54
Q2	Inspiring students to think and learn.	3.08	0.49
Q3	Providing appropriate feedback to enhance student learning.	3.05	0.46
Q4	Encouraging exchange of ideas among students in their learning.	3.34	0.53
Q5	Providing opportunities for students to learning from a variety of ways.	3.11	0.51
Q6	Guiding students to think from different perspectives.	3.05	0.4

Q7	Encouraging students to proactively engage in their own learning.	3.18	0.56
Q8	Being enthusiastic in teaching.	3.11	0.56
Q9	The overall teaching was of high quality.	3.13	0.53

Table 2. Evaluation on the course.

		Mean	SD
Q10	The learning activities of the course stimulated my interest in the subject.	3.03	0.59
Q11	The course enhanced my knowledge and skills in developing block-based programs with Scratch.	3.21	0.7
Q12	The course enhanced my knowledge and skills in developing block-based programs with App Inventor.	3.16	0.75
Q13	I have acquired sufficient knowledge and skills of computational thinking for my teaching.	2.87	0.53
Q14	I am confident in equipping students with computational thinking capabilities through what I have learnt from the course.	2.79	0.62
Q15	The course was valuable to my development.	3.16	0.59

3.1. Qualitative Feedback

Overall, in terms of the qualitative feedback, the participants recognized that block-based programming environments were effective for encouraging beginners to step out of their comfort zone to learn coding, and the interface was user-friendly. However, they expressed that the linkage between acquisition of CT concepts and practices, and the learning of coding needed elaboration. In real classrooms, they would have to explicitly illustrate what CT concepts and practices students were learning when finishing a project. They also appreciated the arrangement of unplugged activities and design tasks which enabled them to better comprehend the CT concepts and aroused their interest in learning deeper. They also agreed that the mini projects were useful for them to explore and develop what they were interested in freely and creatively.

4. DISCUSSION & CONCLUSION

The results suggested that after TDC 1, the teachers agreed that their content knowledge of CT had been enhanced. However, some teachers were not highly confident in their ability to deliver CT education in their classrooms. This makes sense, as TDC 1 mainly focuses on content knowledge of CT, while TDC 2 on pedagogy. In particular, as we found that the teachers were not confident

enough in teaching their students to use App Inventor to build mobile apps, we have modified the design of TDC 2 to include coding tasks in the first few lessons to further their understanding in the use of App Inventor for creating mobile apps and how to connect it with the development of CT. And as the qualitative results suggested that the linkage between programming activities and the acquisition of CT concepts and practices needs to be more explicit, we have articulated more on the linkage in TDC2. The evaluation on TDC2 will be conducted to see whether teachers' PCK and confidence in delivering CT education have been improved.

5. REFERENCES

- Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1), 20–29.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *2012 Annual Meeting of the American Educational Research Association (AERA '12)*, Canada.
- Borko, H., Jacobs, J., & Koeliner, K. (2010). Contemporary approaches to teacher professional development. *International Encyclopedia of Education (3rd ed)*, 548-555.
- Century, J., Lach, M., King, H., Rand, S., Heppner, C., Franke, B., & Westrick, J. (2013). *Building an operating system for computer science*. Chicago, IL: CEMSE, University of Chicago with UEL, University of Chicago. Retrieved from <http://outlier.uchicago.edu/computerscience/OS4CS/>
- Clarke, D., & Hollingsworth, H. (2002). Elaborating a model of teacher professional growth. *Teaching and Teacher Education*, 18(8), 947-967.
- Cuny, J., Snyder, L., & Wing, J.M. (2010). *Demystifying computational thinking for noncomputer scientists*. <http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>
- Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, 46(3), 277-296.
- Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38-43.
- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge: Cambridge University Press.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51-61.
- Menekse, M. (2015). Computer science teacher professional development in the United States: A review of studies published between 2004 and 2014. *Computer Science Education*, 25(4), 325-350.
- Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60-67.
- Saeli, M. Perrenet, J., Jochems, W. M. G., & Zwaneveld, B. (2012). Programming: Teachers and pedagogical content knowledge in the Netherlands. *Informatics in Education*, 11(1), 81-114.
- Scribner, J. P., Cockrell, K. S., Cockrell, D. H., & Valentine, J. W. (1999). Creating professional communities in schools through organizational learning: An evaluation of a school improvement process. *Educational Administration Quarterly*, 35(1), 130-160.
- Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, 15, 4-14.
- Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: towards an agenda for research and practice. *Education and Information Technologies*, 1-14.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2015). *App Inventor 2: Create your own Android apps*. Sebastopol, CA: O'Reilly.
- Yoon, S. A., Anderson, E., Koehler-Yom, J., Evans, C... & Klopfer, E. (2016). Teaching about complex systems is no simple matter: building effective professional development for computer-supported complex systems instruction. *Instructional Science*, 1-23

Connecting Design Thinking and Computational Thinking in the Context of Korean Primary School Teacher Education

Hyungshin CHOI¹, Mi-song KIM²

¹ Chuncheon National University of Education

² University of Western Ontario

hschoi@cnue.ac.kr, misong.kim@gmail.com

ABSTRACT

This current study as part of multi-year design-based research reports our attempt to design and implement a course in teacher education in Korea. We have incorporated design thinking (DT) into the course design and investigated how primary teachers appreciate the role of DT and recognize the connection between teaching computational thinking and DT. This paper reports the course design, its progression, reflections, and learning outcomes.

KEYWORDS

Computational thinking, Design thinking, Physical computing, Teacher Education

1. INTRODUCTION & LITERATURE REVIEW

Drawing upon the power and limits of computing processes (Wing, 2006), the influence of computational thinking (CT) in the 21st century has become widely recognized in innovative educational theory and practice (Resnick & Siegel, 2015; Shodiev, 2014). It is often recognized that, however, little attention has been paid to develop teacher education or teacher professional development with regard to CT. This current study as part of multi-year design-based research reports our attempt to design and implement a course in teacher education in Korea. Specifically, we aimed to introduce design thinking (DT) into a CT course for primary school teachers who were interested in applying CT into their lesson design for primary school students. The study will uncover how primary school teachers perceive impacts of DT on integrating CT into their lesson design. This paper also reports our design which is a new graduate course titled “Creativity in the Technological Field using Design Thinking”.

2. THE STUDY & METHODOLOGY

Despite the fact that DT has been the topic for educational innovation over the last few years in many countries including Korea, there are no teacher education courses integrating DT with CT for teacher education. To respond to this challenge, as teacher educators, we decided to design and implement a graduate course in the field of computer education in Korea.

Our design-based research (DBR) (Collins, Joseph, & Bielaczyc, 2004) adopted the design thinking process to empower in-service teachers who attempted to employ CT

in their classrooms. The research questions are: (1) how are this course design experienced by in-service teachers?; and (2) what are the implications of this course design to improve teacher knowledge of computational thinking through the lens of design thinking? As Table 1 indicated, we designed the course including 12 modules, and each module takes 3 hours. The course was designed to introduce three in-service teachers the five stages of design thinking (DT) (IDEO, 2014) while making a connection with physical computing using Arduino, Lilypad, Makey Makey, and 3D printing.

Table 1. Course Modules and Related Activities

Module	Themes & Activities
1	Design Thinking Overview and Cases
2	Computational Thinking and Physical Computing
3-5	Design Thinking for Educators
6-7	Reflections on Design Thinking (while reading ‘Change by Design’ and ‘Design Thinking Lecture Note’ in Korean)
8	eCrafting with Lilypads
9	3D Printing and Physical Computing
10-11	Designing an Authentic Plan
12	Presentations and Feedback

We used both face-to-face interactions and on-line discussions using Padlet where in-service teachers were able to share reading materials, reading summaries, questions, feedback, reflection papers, and assignments throughout the course (see Figure 1).

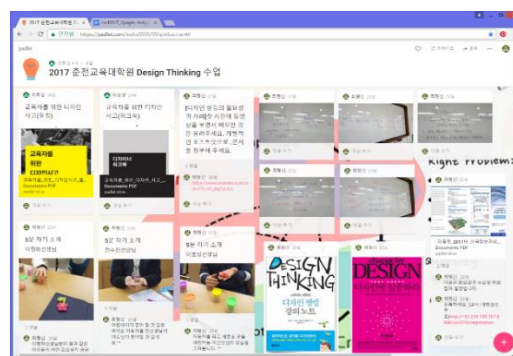


Figure 1. The Course Online Space

3. FINDINGS

3.1. In-Service Teachers' Understanding of Design Thinking Processes

Padlet was collaborative in nature and facilitated in-service teachers' design process experiences. Specifically,

it helped them collaboratively engage in design thinking processes such as the ‘define’ stage followed by the ‘ideation’ stage with either virtual or real post-it notes in Modules 3-5 (see Figure 2).

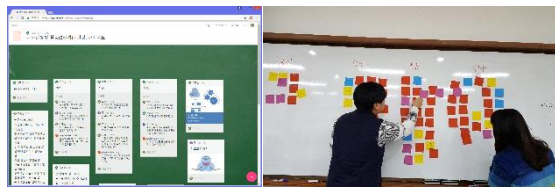


Figure 2. The ‘Define’(left) and ‘Ideation’(right) Stages

For example, drawing upon her experiences in teaching courses in software education, one teacher identified the design challenge of software education for her primary school students: “*How could I make software education space for my students?*”. Like the other in-service teachers, she became more comfortable with the design thinking processes and developed a prototype-driven solution (see Figure 3). She also gained an appreciation for making and improving prototyping by collaborating with many stakeholders (e.g., teachers, colleagues, school staffs, financial administrators, students).

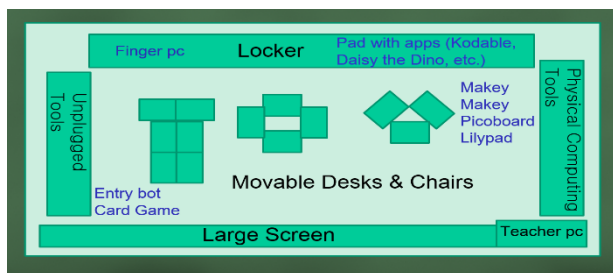


Figure 3. Design Thinking in Computer Education

3.2. Impact of a Design Thinking Course on Teacher knowledge of teaching Computational Thinking

In Modules 10 and 11, as their final projects, teachers developed either authentic lesson plans in their own educational contexts or reflection papers regarding the design thinking processes to address their own educational problems. Two participants created lesson plans and one came up with a reflection paper. One of the lesson plans is ‘a high-level lesson plan for 10 modules to integrate software education for computational thinking into design thinking’. The lesson plan has five stages (learn, ideate, design, make, share). In the ‘learn’ stage, students learn basic skills for block-based programming. In the ‘ideate’ stage, students design a project as a team. In the ‘design’ stage, students sketch their ideas visually, express the movements and complete a scenario. In the ‘make’ stage, students create programs by objects, implement various

programs to produce outputs, record the development processes, and test. Finally, in the ‘share’ stage, students upload their projects online, present them in the class, and evaluate their own learning processes. Instead of a lesson plan, one teacher decided to write a reflection paper using PPTs and developed a solution by applying the design thinking processes. Their final projects reveal that the course helped in-service teachers reflect on a way to make a connection between design thinking and computational thinking through physical computing. One teacher mentioned that “I can incorporate physical computing in my lesson as a tool to make prototypes during design thinking process.” Another teacher also noted that “[t]hey have something in common. They involve promoting students’ creativity. Physical computing helps students implement tangible objects creatively while design thinking encourages them to search for the solutions requiring creativity.” In a similar vein, the other teacher concluded that “[t]hey are connected in terms of educational effects: emphasizing collaboration and learning through failures.”

4. CONCLUSIONS & IMPLICATIONS

This design-based research reports initial design and implementation of one graduate course to connect computational thinking and design thinking for primary school teachers in Korea. As our findings indicate, in-service teachers appreciated the role of design thinking to reflect on and solve their problems collaboratively while integrating computational thinking into their lesson plans. This suggests the need to further explore teacher education by integrating design thinking processes and computational thinking. Our exploration of in-service teachers’ reflections and lesson designs also highlights several key features of physical computing such as promoting creative confidence, making, empathy, and collaboration.

5. REFERENCES

- Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *Journal of the Learning Sciences*, 13(1), 19-32.
- IDEO. (2014). *Design Thinking for Educators*. Retrieved December 10, 2016, from <http://www.designthinkingforeducators.com/toolkit/>
- Resnick, M., & Siegel, D. (2015). *A different approach to coding*. Bright/Medium.
- Shodiev, H. (2014). *Computational thinking and simulations in teaching science and mathematics*. Paper presented at the Applied Mathematics, Modelling, and Computational Science.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 19(3), 33-35

Computational Thinking and Coding Education in K-12

Curriculum Activities to Foster Primary School Students' Computational Practices in Block-Based Programming Environments

Siu-cheung KONG¹, Hal ABELSON², Josh SHELDON², Andrew LAO¹,
Mike TISSENBAUM², Ming LAI¹, Karen LANG², Natalie LAO²

¹The Education University of Hong Kong

²Massachusetts Institute of Technology

sckong@eduhk.hk, hal@mit.edu, jsheldon@mit.edu, acclao@eduhk.hk,
mtissen@mit.edu, mlai@eduhk.hk, karlang@mit.edu, natalie@csail.mit.edu

ABSTRACT

As digital technology is increasingly a part of all sectors of society, educational approaches must be developed in order to nurture students' ability to see the world through a computational lens. One way to achieve this goal is to promote Computational Thinking (CT) for young learners. The CoolThink@JC project is a four-year curriculum pilot designed to integrate CT into Hong Kong upper-primary level schools. The CoolThink framework for curriculum development is structured around computational concepts, practices and perspectives adapted from the framework of Brennan and Resnick (2012). This adapted framework motivated the choice of learning activities for CoolThink. This paper focuses on one aspect of that framework, namely computational practices. Here, we describe how activities in the CoolThink curriculum can promote the computational practices highlighted by the framework.

KEYWORDS

Computational thinking, Programming education, Curriculum activities, K-12 education

1. INTRODUCTION

Digital technologies are transforming virtually all aspects of modern living. The ability to productively engage with digital tools has become a requisite skill for empowered citizens. More and more, people need to confront and solve problems in computational terms in order to drive innovation and improve quality of life (Looi, Chan, Wu, & Chang, 2015; Yadav, Mayfield, Zhou, Hambruch, & Korb, 2014).

As we educate tomorrow's leaders, there is a growing need to nurture young people's *computational thinking* (CT) abilities in order to help prepare them to engage effectively in a digital world. CT is drawing worldwide attention from educational planners and policy makers. It is increasingly being singled out as a skill that students should acquire (Wing, 2006). Many advocates contend that more traditional subjects in K-12 education should be integrated with CT development through a curriculum where students engage in computer programming (Barr & Stephenson, 2011; Fluck et al., 2016; Grover & Pea, 2013; Kaifai & Burke, 2013; Tucker, 2003).

The CoolThink@JC Project (CoolThink) is a four-year curriculum pilot aimed at integrating CT into Hong Kong schools. The work focuses on senior primary school learners (grades 4 through 6) aged 9-12, in the belief that this is the appropriate age to spark student emerging interest in computing, whereas waiting until middle school or high school may be too late (Tai et al., 2006).

The curriculum design of CoolThink follows the framework for CT set out by Brennan & Resnick (2012), which breaks CT into three parts: *computational concepts* (CT Concepts) that designers engage with in programming, *computational practices* (CT Practices) designers exercise while programming, and *computational perspectives* (CT Perspectives) that programmers develop about themselves and the world around them.

This paper describes three programming activities that CoolThink is currently piloting at a dozen Hong Kong primary schools. We focus on activities designed to guide the development of computational practices within the CT curriculum.

2. LEARNING OUTCOMES OF COMPUTATIONAL THINKING

The term "computational thinking" in education was first used in relation to child education by Papert (1980) with reference to Logo, a computer language designed for children. As Papert wrote, "I believe that certain uses of very powerful computational technology and computational ideas can provide children with new possibilities for learning, thinking, and growing emotionally." In 1980, computers were neither powerful nor affordable enough for Papert's vision to be widely realized, but that changed over next two decades. As computers became less expensive, more powerful, and more accessible to many, the term "computational thinking" became prominent in computer science education starting with a highly influential article by Jeanette Wing (2006). She defined computational thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Cuny, Snyder, & Wing, 2010). Today, it is widely recognized that CT is a broadly applicable set of skills that can help people in fields as diverse from astronomy to zoology,

and seemingly everything in between. While CT has long been regarded as a specialized skill that only computer scientists need to develop, it is increasingly being considered an essential cognitive ability for everyone in a digital-mediated world, due to its alignment with twenty-first century skills such as problem-solving and creativity (Binkley et al., 2012).

Figure 1 shows CoolThink's framework for learning outcomes (Kong, 2016). It is structured as the foundation of the curriculum by placing emphasis on the outcomes of CT development, following the three key dimensions of CT (i.e. computational concepts, practices and perspectives) developed by Brennan and Resnick (2012).

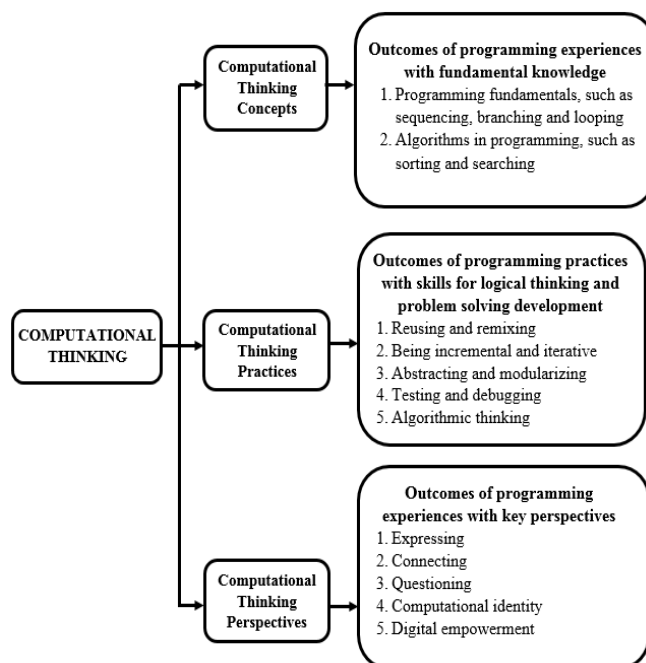


Figure 1. CoolThink framework of learning outcomes of CT adapted from Brennan and Resnick (2012).

In this paper, we focus on CT Practices. Following Brennan and Resnick, we call out the practices of reusing and remixing, being incremental and iterative, abstracting and modularizing, testing and debugging, and employing algorithmic thinking. Helping students develop these skills is a key part of a programming curriculum for computational thinking. As an example, we consider how to guide the development of CT Practices within the context of the CoolThink curriculum.

3. DESIGNING LEARNING ACTIVITIES TO SUPPORT CT PRACTICES

3.1 Making CT Learning Environments for K-12 Students

This section illustrates, with concrete examples, how learning activities for upper-level primary school learners can be designed to support the CT Practices highlighted in the CoolThink framework. While programming is challenging to learn, there have been considerable efforts to make it more accessible to novice

learners. One approach that has been proven successful in teaching novice programmers is the use of block-based programming environments (Meerbaum-Salant, Armoni & Ben-Ari, 2010). For example, Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010) and App Inventor (Wolber, Abelson, Spertus, & Looney, 2011) are two commonly used block-based programming environments for novice programmers (Price & Barnes, 2015).

First introduced in 1986 (Glinert, 1986) and again in LogoBlocks (Begel, 1996), block-based environments provide blocks that can be dragged and dropped into a scripting pane to build stacks of blocks. This allows learners to develop programs without programming syntax. The shape and the visual layout of the blocks allow learners to understand the logic flow, making programming more concrete and easier to use by young learners (Weintrop & Wilensky, 2015). Syntax errors are also reduced in block-based programming contexts as the blocks only “fit together” when the code makes sense. Therefore, the learning activities in the CoolThink curriculum are designed in the context of Scratch and App Inventor.

The CoolThink curriculum has three levels. These are intended in our pilot for Hong Kong grades 4, 5, and 6, respectively (roughly ages 9-12). The curriculum begins with a series of Scratch units in level 1, which comprise the first half of level 1. The Scratch units serve as an easily accessible introduction for learners new to programming. After the Scratch units equip learners with some programming concepts and practices, App Inventor is introduced as the environment for the rest of the curriculum, beginning with the second half of level 1 and continuing through levels 2 and 3. Taken altogether, the three levels will be completed over the course of three school years. Designed to progress as students grow in both experience and ability, the curriculum introduces more complex and authentic computational tasks as it progresses.

3.2. Providing for CT Practices in Curriculum Design

We suggest that development of CT practices through programming tasks should be a key goal of this curriculum in order to nurture and enhance learners' problem-solving ability. In the learning outcomes for the CoolThink CT framework, five sets of the CT practices are targeted in the design of the programming activities. This section demonstrates how the activities in the CoolThink curriculum can help support the CoolThink framework's CT practices.

Creation involves combining existing and new ideas (Chan, Looi, & Chang, 2015). Therefore, to nurture learners as creative problem solvers, it is important to support them in developing the CT practices of reusing and remixing. Reusing refers to recalling code student have used in previous projects and incorporating it again in new programming tasks. Remixing involves building on their work or the work of others to create new and more complex artifacts (Brennan & Resnick, 2012). For example, consider the activity “Making a Maze Game with Scratch” (Level 1, Unit 2). This activity uses

coordinates to control the movement of a sprite (the panda in Figure 2). The same code can be reused in the Scratch Mini Project (to be completed after the Scratch units that make up the first half of Level 1). For this project, students must create either a story or a game. One strategy learners may employ is to reuse game code by extracting it from their “Backpack” (a tool in the Scratch environment for storing and sharing code across programming projects) into a new programming project for their Scratch Mini Project. (see the green box in Figure 4). Students can remix the event and the “forever” loop to make the cat move continuously (see Figure 5). In order to enhance the practices of reusing and remixing, the learning activities of the curriculum should be inter-related and increasingly complex.



Figure 2. The maze game in unit 2 at level 1.

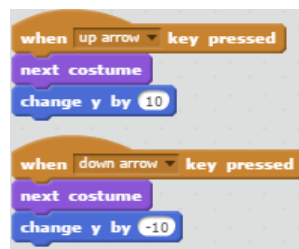


Figure 3. The use of the coordinates in the maze game.



Figure 4. Reuse the codes of the coordinates (red box), extracted from “Backpack” (green box) in Scratch Mini-Project.

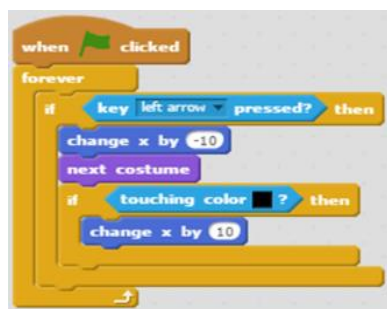


Figure 5. Remix the event and forever loop to enable the cat to move continuously.

3.3. Example: The Addition Game Activity’s Support for Multiple CT Practices

The CoolThink curriculum’s “Addition Game” (Level 1, Unit 6) supports multiple CT practices. In this App Inventor game, three numbered balls roll horizontally from left to right at a slow pace. Players must determine whether any two of the three numbers sum to 10 and press the “yes” or “no” button before the balls reach the screen edge. This Unit provides students with an opportunity to use the CT practice of developing incrementally and iteratively.

In developing the game, learners must generate three numbers in each round. Early in the unit, learners are asked to build a program that generates three random numbers (see Figure 6 for one possible solution to this challenge). Learners quickly observe that two of the numbers rarely sum to 10. This is problematic if the goal of the game is to have two numbers sum to ten before the player adds the third number. Therefore, learners need to iterate on their design to increase the probability that the sum of two of the three numbers is ten. Learners, under their teachers’ guidance, will refine the design to guarantee that two of the numbers do sum to 10. One possible approach is to generate a second number from the difference between 10 and the first random number (Figure 7).



Figure 6. Code with a very low probability of getting a sum of 10 from two of the three random numbers.

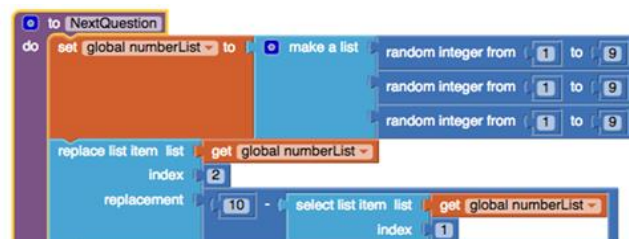


Figure 7. Modified code that always yields get a sum of 10 from two of the three numbers.

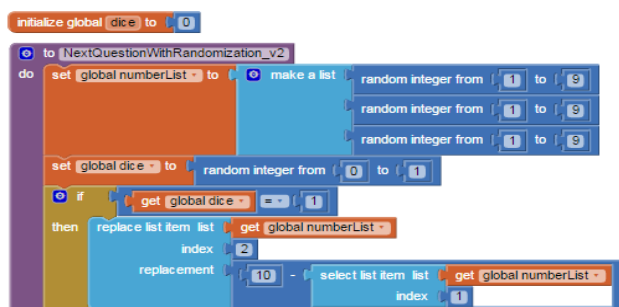


Figure 8. Another modification where there's nearly a half chance of getting a sum of 10 from two of the three random numbers.

However, this change makes the first two numbers always sum to 10 (making the game boring). In order to create a more engaging game, with teachers' guidance, learners are encouraged to refine their code such that both situations (rarely having a sum of 10 versus always having sum of 10) occur with nearly equal chance. Learners may further iterate on their design by randomizing the two situations with the use of a "global dice", as shown in Figure 8, in order to produce a roughly equal chance of having a sum of 10 and no sum of 10. By developing this solution, learners get the opportunity to experience and understand the iterative and incremental processes for constructing a computational artifact.

As computer programs become more complicated, it is useful to organize the programs by *abstracting and modularizing* code. The addition game app provides an opportunity to expose learners to abstracting and modularizing their code. Abstraction is the CT practice of defining patterns, generalizing from instances, and parameterization, and is essential to deal with complexity and scale (Wing, 2011). Modularizing is the decomposition of complex problems, which helps structure large-grain programs (Parnas, 1972). In order to enhance young learners' understanding of the CT practices, learners can be guided to modify their solution in Figure 8 into the one in Figure 9.

As shown in the version of the code Figure 9, the main program helps develop learners' abilities for high-level abstraction by calling with equal chance the module of *rarely having a sum of 10* and the module *always having a sum of 10* (i.e., when the program randomly generates either 1 or 2, using the "global dice", the corresponding action is called). For example, when the "global dice" rolls 1, the program will generate three random numbers without guaranteeing that some pair of numbers sums to 10.

Figure 9 shows a version of the program where the two actions are coded as independent modules implemented as procedures in the program. This modification helps instill in learners the importance of abstracting and modularizing when programs become more complex, thus requiring the tasks to be decomposed and tackled one-by-one.

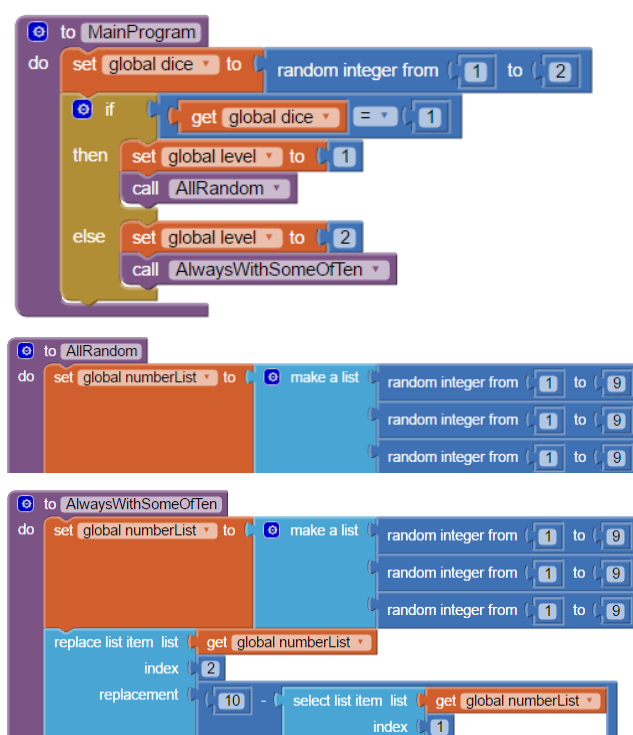


Figure 9. The main program and two procedures in the Addition Game for reinforcing the abstracting and modularizing skills of learners.

The process described above of making small, frequent refinements is an example of another of the CT practices, namely *being incremental and iterative*.

As well, the Addition Game activity reinforces the CT practices of *testing and debugging*. Students often encounter these practices through trial and error or through "support from knowledgeable others" (Brennan & Resnick, 2012). In a programming curriculum, all the learning tasks should involve the practice of testing and debugging designs as they evolve. One pedagogical technique for reinforcing testing and debugging is to provide learners with a program that involves errors in its logic. Figure 10 shows a version of the Addition Game code that contains a logical error (a self-referencing variable). The blocks could be debugged by a teacher pointing out the error directly. However, if the teacher presents this code for the class to test and repair, it can be a valuable opportunity for the whole class to collectively develop their testing and debugging skills. The intent of the design shown in Figure 10 is to randomly generate three numbers such that two of the three numbers sum to 10. It is expected that some learners will be able to immediately spot the bug in the code, realizing that none of the numbers are coupled, that the Q1 statement only changes Q1. It is not dependent on either Q2 or Q3. Other learners may need to test the code to discover that the code doesn't work as intended. Learners will then need to go through the process of revising the code, testing and debugging it until the program is bug-free (see Figure 11 for one possible solution). Through this process, learners will learn about iterative cycles of coding, testing, and debugging.



Figure 10. A logical error in the design of codes in always getting a sum of 10 from two of the three random numbers.

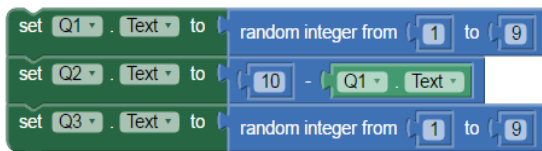


Figure 11. The correct sequence of codes in always getting a sum of 10 from two of the three random numbers.

3.3. Supporting Algorithmic Thinking

The practice of algorithmic thinking is a key element in the development of CT (Angeli et al., 2016; Barr & Stephenson, 2011; Selby & Woollard, 2013; Wing, 2006). It can equip learners with the ability to systematically process information, understand symbol systems and representations, flow of control, and conditional logic (Grover & Pea, 2013). The CoolThink curriculum's "Voting App" (Level 3, Unit 1) is an example of support for algorithmic thinking. This App Inventor activity asks students to develop an app that presents various options to be voted on and records the votes from multiple voters in a cloud database. Votes are tallied and displayed for voters on their individual devices. Figure 12 shows a high-level approach to developing this kind of app, together with the associated data flow. Asking learners to draw dataflow diagrams can reveal to what degree they understand the necessary data flow of the app and the logic they will code in order to achieve that data flow.

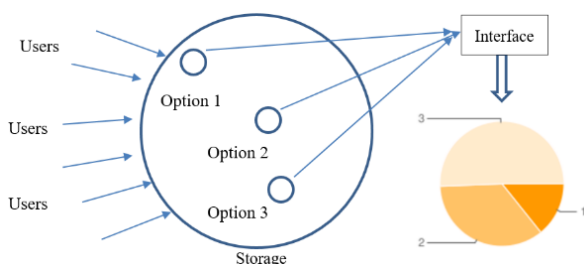


Figure 12. The data flow diagram that helps elucidate showing the learner's algorithmic thinking skills.

Following the detailed design of data flows, learners can start building the Voting App. These two stages of high level abstraction and algorithmic design can help learners' process information in a more systematic way and facilitate the development of their algorithmic thinking.

Figure 13 shows the algorithmic design of adding the count of each "vote" by one. When an option is voted on by a user, the algorithm starts by requesting the current vote counts of the voter's choice from the cloud database.

When the current vote is returned, that value is incremented by one, and sent back to be stored as the new value for that choice in the cloud database.

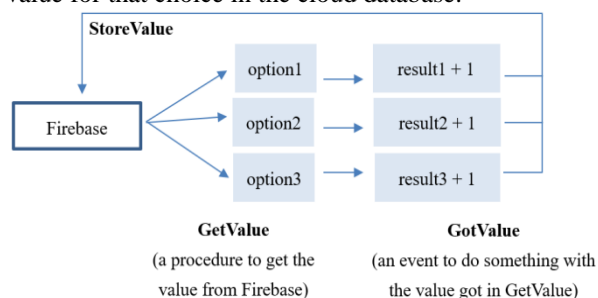


Figure 13. An algorithmic design showing the algorithmic thinking skills of learner with implementation details on the data flow.

4. CONCLUSION AND FUTURE WORK

To nurture young learners as creative problem solvers in this digital world, requires development of computational thinking early in K-12 education. This paper illustrates how the CoolThink framework incorporates learning activities aimed at developing CT Practices in a programming curriculum for upper-level primary school students. Future work will address the additional CoolThink framework elements of CT concepts and CT perspectives. Based on the example of algorithmic thinking in the CoolThink curriculum, it will be worthwhile to explore some other computational concepts such as synchronization and atomicity in multiple user databases since experience with young learners exploring these concepts in Scratch is the subject of a recent doctoral dissertation (Dasgupta, 2016). It will also be necessary to assess student learning outcomes for the CT elements and determine if the CoolThink design principles do indeed support the development of computational thinking in students in the target grades. This may be particularly challenging for CT practices, where learning manifests as activities learners perform and objects they build. In order to more accurately understand student CT learning, it seems preferable to use a breadth of tools, including not only standardized tests, but also instruments based on students' developed artifacts (e.g., design documents, written code, and final products), and even classroom observations and small group interviews.

5. REFERENCES

- Angeli, C., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 Computational Thinking Curriculum Framework: Implication for Teacher Knowledge. *Educational Technology & Society*, 19(3), 47-57.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48-54.
- Begel, A. (1996). *LogoBlocks: A Graphical Programming Language for Interacting with the World*. Cambridge: Massachusetts Institute of Technology.

- Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. In P. Griffith, B. McGaw, & E. Care (Eds.), *Assessment and teaching of 21st century skills* (pp. 17-66). Netherlands: Springer.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association meeting*, Vancouver, BC, Canada.
- Chan, T. W., Looi, C. K., & Chang, B. (2015). The IDC Theory: Creation and the Creation Loop. In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F. Qiu (Eds.), *Workshop Proceedings of the 23rd International Conference on Computers in Education* (pp. 814-820). Hangzhou, China: Asia-Pacific Society for Computers in Education.
- Coolthink (2016). "Grounded in computer science principles, computational thinking empowers students to move beyond mere technology consumption and into problem-solving, creation and innovation." <http://coolthink.hk>
- Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying Computational Thinking for Non-Computer Scientists*. Work in Progress.
- Dasgupta, Sayamindu (2016). *Children as Data Scientists: Explorations in Creating, Thinking, and Learning with Data* (Doctoral dissertation). Massachusetts Institute of Technology, September, 2016.
- Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J., & Zagami, J. (2016). Arguing for Computer Science in the School Curriculum. *Educational Technology & Society*, 19(3), 38-46.
- Glinert, E.P. (1986), Towards "Second Generation" Interactive, Graphical Programming Environments *Proceedings of the 2nd IEEE Computer Society Workshop on Visual Languages*, 61-70.
- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(1), 38-43.
- Kafai, Y., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, 95(1), 61-65.
- Kong, S. C. (2016). A framework of curriculum design for computational thinking development in K-12 education. *Journal of Computers in Education*, 3(4), 377-394.
- Looi, C. K., Chan, T.-W., Wu, L., & Chang, B. (2015). The IDC Theory: Research Agenda and Challenges. In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F. Qiu (Eds.), *Workshop Proceedings of the 23rd International Conference on Computers in Education* (pp. 796-803). Hangzhou, China: Asia-Pacific Society for Computers in Education.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4), 16.
- Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M.M. 2010. Learning computer science concepts with scratch. Proc. of the 6th Annual ICER Conference (2010), 69–76.
- Papert, S. A. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books.
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053-1058.
- Price, T. & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 91-99). Omaha, USA: ACM.
- Selby, C., & Woollard, J. (2013). Computational Thinking: The Developing Definition. In *19th Annual Conference on Innovation and Technology in Computer Science Education*. Canterbury, Great Britain.
- Tai, R. H., Liu, C. Q., Maltese, A. V., & Fan, X. (2006). Planning early for careers in science. *Life sci*, 1, 0-2.
- Tucker, A. (2003). A Model Curriculum for K-12 Computer Science. *Report of the ACM K-12 Education Task Force Computer Science Curriculum Committee – Draft*. Retrieved January 26, 2017, from <http://www.acm.org/k12/k12Draft1101.pdf>
- Weintrop, D., & Wilensky, U. (2015). To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. In M. Lee & T. Strelevitz (Eds.), *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199-208). New York, NY: ACM.
- Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35.
- Wing, J. M. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, Carnegie Mellon University, Pittsburgh. Retrieved from <http://link.cs.cmu.edu/article.php?a=600>
- Wolber, D., Abelson, H., Spertus, E., & Looney, L., *App Inventor: Create your own Android Apps*, (2011), O'Reilly Media.
- Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, 14(1), 1-16.

Emergent Roles, Collaboration and Computational Thinking in the Multi-Dimensional Problem Space of Robotics

Florence R. SULLIVAN^{1*}, P. Kevin KEITH²

¹University of Massachusetts, Amherst

²Landmark College

fsullivan@educ.umass.edu, kevinkeith@landmark.edu

ABSTRACT

This study presents a sequential analysis of the relationship of emergent roles to student collaboration and computational thinking in the multi-dimensional problem space of educational robotics. The interactions of six groups (n=17) of middle-school aged girls participating in a one-day introduction to robotics workshop were video and audio recorded. Here we analyze one group of three girls' interactions and the emergence of distinct roles that correlate with periods of collaboration and periods of parallel solo work, which, in turn, impact student's engagement in computational thinking including solution planning, algorithmic operations, and design of the robotic device. Suggestions for future research are provided.

KEYWORDS

Robotics, Collaborative Learning, Computational Thinking, K-12 Education, Roles

1. INTRODUCTION

Computational thinking is foundational to success in computer science (Wing, 2006). A current goal in the context of education in the USA, is to provide computer science instruction for all students in K-12 settings (National Science Foundation, 2016). Robotics is an activity that has the potential to stimulate students' computational thinking (Sullivan & Heffernan, 2016). Yet, there is little research devoted to this relationship. Here, we focus on how collaborative arrangements in robotics learning environments influence group participation and engagement in computational thinking for girls, with an emphasis on the impact of group roles on collaboration.

Group roles are an important element of computer supported collaborative learning (Hoadley, 2010). They help to define the expected behavior of the members of the group (Jahnke, 2010). Scripted roles are those that are assigned by a teacher to facilitate the process of collaborative learning. This is contrasted with emergent roles that "emerge spontaneously or are negotiated spontaneously by group members without interference by the teacher or researcher" (Stijbos & De Laat, 2010). Emergent roles are typical in open-ended robotics activity, such as in this study.

In discussing group work, it is important to understand when groups are working collaboratively vs. cooperatively. In cooperative group work, the task is divided among the members, knowledge building occurs through individual actions, the results of which are later

shared with the group (Dillenbourg, 1999). Successful collaborative group work requires ongoing, well coordinated group interactions (Barron, 2003), while cooperative group work only requires an initial division of the task. Arguably, collaborative learning results in greater learning outcomes for students. In our study, students were asked to collaborate, but were not assigned specific roles. Rather the students were urged to work together and take turns, hence, group roles emerged.

Robotics learning environments are multi-dimensional problem spaces which afford multiple roles that may be taken up. These problem spaces consist of a computer (programmer), a robotic device (builder), and a space to test the robot (analyst). The multiple tools in this problem space can create a situation where students vie for control of the tools through adopting certain roles (Jones & Issroff, 2005).

Computational thinking (CT) has been defined as formulating problems in ways that enable us to use a computer to solve them, and automating solutions through algorithmic thinking (Computer Science Teacher Association – CSTA, 2016). Moreover, these skills are important because they create a tolerance for ambiguity, allow for persistence in working with difficult problems, and for practicing communication in working with others to achieve a common goal (CSTA). In this study, we focus on how emergent roles in the multi-dimensional problem space of robotics relates to collaboration and to different types of computational thinking. The aim of this research is to improve robotics curriculum and teaching for students.

2. SAMPLE

This study took place at a one-day, all girls introduction to robotics event. The participants in this study included 17 girls ages 8-13 (M = 11.725). All of the participants were working with robotics for the first time. The students worked on solving robotics challenges drawn from the First Lego Leagues (2011) food factor challenge. The students were divided into six teams (five teams of 3 and one team of 2). Due to size limitations, this paper focuses on three students that comprise one team. The data set that was analyzed for this study consists of 3 hours and 11 minutes of problem solving video observations (11,516 seconds). Pseudonyms are used throughout.

3. METHODS

This study utilized the iterative sequential mixed method design and consists of three phases (Teddie & Tashakkori, 2009, p. 155). In the first phase, emergent roles were

identified and quantitative means were used to establish the amount of time that each individual team member engaged in each role as well as the time that they were acting on their own, working cooperatively or collaboratively. In the second phase, the transcript of all utterances were coded using an a priori (Teddle & Tashakkori, 2009, p. 252) coding structure of different categories of discourse, including computational thinking categories that were subsequently observed as being used by novice students in a robotics environment. In the third phase, quantitative methods were used to describe the sequence of events as they unfold in time and the likelihood of the interrelation of the roles, collaboration, and computational thinking.

3.1 Phase I – Quantitative

The first quantitative phase of this study was to record onset and offset times of certain behaviors. All 3 hours and 11 minutes of video were coded (for each student) and no overlapping of the codes occurs. The unit of analysis for this phase of coding was ‘change of focus.’ The codes for the roles were: Programmer (Active or Observer), Tester-Debugger, Builder (Active or Observer), Analyst, Other.

The codes for collaboration (Table 1) are based on Forman and Cazden's (1985) codes for participation in groups as markers of coordination.

Table 1. Collaboration Codes	
Type	Description
Parallel	Little to no focus on the group.
Cooperative	Working together, focused on own results.
Collaborative	Working together and sharing ideas
External	Focused on something outside the group.

Inter-rater reliability was assessed by training a second coder and then having them view a portion of the data. Results for inter-rater reliability for the role were $\kappa = .83$ which indicate that inter-rater reliability for this study was adequate. Results for collaboration were $\kappa = .92$.

3.2 Phase II – Qualitative

The qualitative phase focused on the transcripts of the discourse related to the robotics activity. The transcripts were coded using a-priori codes based on the work of Wing (2006) and Barr and Stephenson (2011), including analysis, algorithmic thinking, designing, non-specific test outcome, points – competition and other. These codes have been synthesized to be relevant for the activities and type of coding expected and observed for novice programmers in a robotics environment. Inter-rater reliability was calculated utilizing Krippendorff's alpha (Krippendorff, 2004). Results for inter-rater reliability for the discourse were $\alpha = .901$ which indicate that inter-rater reliability for this study was high. The coded utterances were then assigned to a time sequence in the video corresponding to when they were spoken.

3.3 Phase III Quantitative

The first step in the phase III quantitative analysis was to calculate descriptive statistics to summarize the coded observations for each individual student. The total time and relative duration were calculated for each observed timed event (role and collaboration). The second step in the quantitative analysis was to describe the joint probabilities of certain pairs of coded behaviors for each individual student. A joint probability is the probability that an event will occur given another event. When the time that the event occurred is also coded, the joint probability includes the element of time and the probability is calculated such that it is the probability that an event will occur given another event in the same time frame. This is also called Lag(0) (Bakerman & Quera, 2011) analysis since the calculation describes the co-occurrence of events in the same time frame given that the displacement in time is zero. Joint probabilities, or Lag(0) analysis, were calculated to compare role to collaboration, role to computational thinking, and collaboration to computational thinking.

4 Results

Data were analyzed with GSEQ 5.0 (Bakerman & Quera, 2011) to examine the students' behavior. GSEQ is a data analysis program designed to explore observational sequential data. This program allows for the computation of both simple statistics, such as frequencies, and contingent statistics, such as relative frequency or conditional probabilities.

4.1 Roles Exhibited by Students

To begin exploring for patterns, the total duration and relative duration that each student assumed a role was calculated and presented in Figure 1. For this analysis, duration was expressed in seconds.

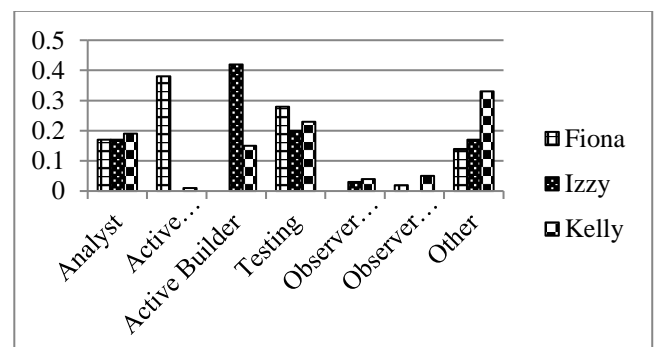


Figure 1. Relative Duration of Role

The results indicate that Fiona's primary role was that of programmer. Of the observed time, she spent 38% of her time in this role. Izzy's primary role was builder. This is evident by 42% of her time was spent in this role. The data also shows that she was never engaged with the programming of the robot. Kelly never had a primary role, and spent 33% of her time doing other tasks.

4.2 Collaboration

No episodes of cooperation, as defined by this study, were observed. Students were either working together with one set of materials (collaboration) or working alone (parallel). Duration and relative duration were calculated for each of

the students and the relative duration results are presented in Figure 2.

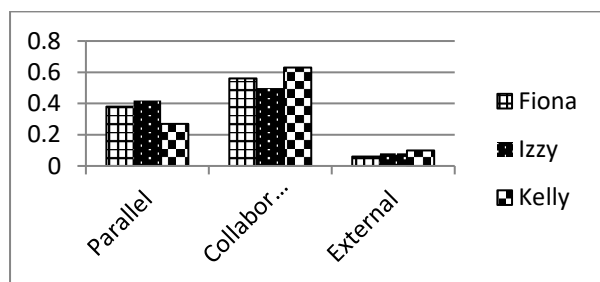


Figure 2. Relative Duration of Collaboration

The data indicates that the students spent at least half of the time jointly attentive with at least one other student. Additional analysis was done to determine how much of the time that all three students were simultaneously involved in collaboration. The data was recoded and showed that all of the students were simultaneously coded as being collaborative for only 34% (3,937 seconds) of the observed time.

4.3 Computational Thinking

As regards CT, Izzy appears to have taken the lead in the analysis ($n = 124$, 43%) which is supported by the video data. Izzy maintains control of the challenge instructions at the beginning of the work time and leads the discussion of how the team can win the challenge. However, although she had a lot to say about the initial analysis, she said very little to say about how this was carried out. This is apparent in her low percentage of algorithmic thinking ($n = 21$, 7%) and testing outcomes ($n = 29$, 10%). The rest of Izzy's discourse mostly revolved around the design of what she was building ($n = 105$, 36%).

The data also indicates some conclusions that refer directly to the questions being asked in the study regarding computational thinking. Twenty-two percent of Fiona's discourse relates to the operations of the program ($n = 58$, 22%). However, when we look at the discourse around the variable, it is almost non-existent. When it did occur, the majority of the discourse about a variable was from Fiona ($n = 13$, 5%). This is not a surprise since she took on the primary role of programmer and rarely relinquished that role. Another interesting fact that is apparent in the data is the high frequency of talk regarding the design. In order to be successful with many of the challenges, additional design was required to create an implement to be added to the robot. Izzy and Kelly both had a very high frequency ($n = 105$, $n = 123$ respectively) of discourse related to the design of the implement.

4.4 Joint Probability – Role and Collaboration

The next phase of the quantitative analysis was to begin to examine patterns between the role that the student assumed and the type of collaboration that was observed. One notable outcome from this analysis show that when a team member is involved in their primary role (e.g., programmer, builder), they perform that role in a non-collaborative way. Two roles do stand out as having a higher probability of being collaborative. One is the tester role which has a relatively high probability for all of the students. This was especially true for Kelly with a

probability of being collaborative when taking on the testing role of 95%. Kelly participated in almost every test of the robot at the challenge arena with at least one other member from the team. Fiona's calculated probability of being collaborative when testing was also high at 82%. Izzy's results are similar to Fiona's at 83% probability of being collaborative and 17% probability of not being collaborative when testing. The other role that has a high probability of being collaborative is the analyst role. For Fiona, the probability of being collaborative is 75% and for both Izzy and Kelly the probability is 89%. Most of the analysis work was done at the beginning of the challenge when all of the students were initially working together.

Joint Probability – Role and Discourse Type

The last phase was to examine the patterns between the role and the type of discourse. Joint probabilities were calculated and patterns emerged. For all three students, a majority of their discourse about the analysis of the task was done when they assumed the analyst role. The largest of these was Kelly, who exhibited a 93% probability of a joint occurrence of this discourse type and role. For Fiona the probability was 89% and for Fiona the probability was 78%. For Fiona, the discourse around computational thinking was associated with her main roles of active programmer and her secondary role of tester. Izzy's discourse related to the operations of the robot occurred mostly when she was involved in the tester role ($n = 10$, 50%). Kelly was also more likely to discuss the operations of the robot while testing ($n = 22$, 65%). Kelly had 5 instances of discourse related to the variable. Of these, 4 occurred while taking on the programmer role and 1 occurred while testing.

5 DISCUSSION

Our analysis indicates that roles play an important part in the level of collaboration that occurs within the group. The roles afforded by the environment: (a) were taken up and never relinquished, (b) influenced the type of discourse that was used to discuss the activity, and (c) affected the common understanding of the different systems in a robotics environment. The roles emerged early on in the process and were fairly stable throughout the activity. The roles may also have been partially structured by the group sharing a single technology resource (Jones & Issroff, 2005). The laptop and the robot are sized to be utilized by a single individual. The low frequency of discourse for both of the programming roles leads credence to this interpretation. The testing area was a four foot by eight-foot arena which facilitated the group coming together to discuss the outcomes of not only the programming of the robot, but also the building of the implement designed to help solve the challenge.

The analysis of the discourse shows that the type of speech used when discussing the program, mostly when testing, was very similar in structure to the type of speech used during analysis. Students continued to use words such as forward, backward, and turn but added non-specific modifiers such as: more, less, sharper. Given the nature of the programming environment and the structure of the

programs, for computational thinking to be more evident, we would have expected to hear discourse about the different types of blocks used and the values of the variables. We expect students to understand the factors that determine the robots speed, turning radius, or power rather than just expressing that the robots is going fast or slow (Barak & Zadok, 2009).

This is in sharp contrast to the type of speech used when the students were engineering and building their robot. The discourse associated with building, at times, used very specific language to indicate the part needed. The factors influencing this disparity in establishing intersubjectivity require further research. These roles were emergent roles and not structured, possibly affecting the way that the group collaborated. If the roles had been rotated by some structured means or if there had been an opportunity to create intersubjectivity early on, there may have been some increase in the ability of the group to create a shared understanding of how the robot is programmed. Likewise, if the materials were easier to share, for example if a large touchscreen computer was used to program, this too, may alter student interactions. Future research should investigate these possibilities.

6 REFERENCES

- Bakerman, R., & Quera, V. (2011). *Sequential Analysis and Observational Methods for the Behavioral Sciences*. New York: Cambridge University Press.
- Barak, M., & Zadok, Y. (2009). Robotis projects and learning concepts in science, technology, and problem solving. *International Journal of Technology and Design*, 289-307.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 48-54.
- Barron, B. (2003). When smart groups fail. *Journal of the Learning Sciences*, 12(3), 307-359.
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Education and Psychological Measurement*, 37-46.
- CSTA. (2016). *Computational Thinking*. Retrieved from <https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf>
- Dillenbourg, P. (1999). *Collaborative Learning: Cognitive and Computational Approaches*. *Advances in learning and instruction series*. New York, NY: Elsevier Science.
- Forman, E. A., & Cazden, C. B. (1985). Exploring Vygotskian perspectives in education: The cognitive value of peer interaction. In J. V. Wertsch (Ed.), *Culture, Communication, and Cognition: Vygotskian perspectives* (pp. 323-347). New York: Cambridge University Press.
- Hoadley, C. (2010). Roles, design, and the nature of CSCW. *Computers in Human Behavior*, 551-555.
- Jahnke, I. (2010). Dynamics of social roles in a knowledge management community. *Computers in Human Behavior*, 533-546.
- Jones, A., & Issroff, K. (2005). Learning technologies: Affective and social issues in computer-supported collaborative learning. *Computers & Education*, 395-408.
- Krippendorff, K. (2004). *Content analysis, and introduction to its methodology 2nd edition*. Thousand Oaks, CA: Sage Publications.
- National Science Foundation (2016). Building a foundation for CS for all. https://www.nsf.gov/news/news_summ.jsp?cntn_id=137529
- Stijbos, J. W., & De Laat, M. F. (2010). Developing the role of concept for computer-supported collaborative learning: An explorative synthesis. *Computers in Human Behavior*, 495-505.
- Sullivan, F.R. & Heffernan, J. (2016). Robotic construction kits as computational manipulatives for learning in the STEM disciplines. *Journal of Research in Technology Education*, 49(2) 105-128. DOI: 10.1080/15391523.2016.1146563
- Teddie, C., & Tashakkori, A. (2009). *Foundations of Mixed Methods Research*. Thousand Oaks, CA: Sage Publications.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 33-35.

Acknowledgements: The research reported in this manuscript was supported by a grant from the National Science Foundation DRL#1252350. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

A Framework of Computational Thinking Curriculum for K-12 with Design Thinking by App Inventor

Peng CHEN^{1,2}, Ronghuai HUANG^{1*}

¹ Smart Learning Institute, Beijing Normal University, Beijing

² Department of Educational Technology, Capital Normal University, Beijing
pengchenbnu@163.com, huangrh@bnu.edu.cn

ABSTRACT

Computational Thinking (CT) has become popular in recent years and has been recognized as an essential skill for the digital generation. Students are exposed to computational thinking when they do programming, and MIT App Inventor is currently one of the most popular block based programming environments. Meanwhile, Design thinking is considered as a creative, human-centred, participative, exploratory and problem-solving process that values different perspectives of a problem. In this study, we aim to bring the design thinking in a curriculum framework of K-12 to promote computational thinking by App Inventor. The future work is to implement and evaluate CT curriculum.

KEYWORDS

Computational Thinking, App Inventor, Design Thinking, Curriculum Design

1. INTRODUCTION

Over the past three decades, CT has gained extensive attention and become accepted as one of the skills required by those growing up in the digital era; especially after being defined by Wing in 2006. She presented that computational thinking as a way of “solving problems, designing systems and understanding human behavior by drawing on the concepts of computer science”, and she argued that CT “represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use” (Wing, 2006). After that, CT has gained a lot of attraction, and many countries and researchers have involved in this topic. Computer programming is an excellent way to develop computational thinking skills (Orr, 2009), because it involves the use of computer science concepts such as abstraction, debugging, remixing and iteration to solve problems (Brennan & Resnick, 2012; Ioannidou, 2011; Wing, 2008). MIT App Inventor is currently one of the most popular blockbased programming environments. The main goal of App Inventor is to teach computing and programming to students with limited prior programming knowledge and to democratize app creation by providing an easy-to-learn environment. It has experienced broad adoption in diverse venues, and researchers have used it in summer camps and other outreach activities for K-12 students for several years now (Ericson & McKlin, 2012; Roy, 2012; Wagner, Gray, & Wolber, 2013). Conform to the situation of China, how to develop the solution program for the different level of K-12 information technology

courses to foster computational thinking and innovation capacity is the most important thing we focus on. Our whole project aims to use the App Inventor combine with the information technology course to cultivate K-12 students' computational thinking. In this study, we describe a framework of computational thinking Curriculum for K-12 with Design Thinking by App Inventor.

2. THEORETICAL FRAMEWORK

Design thinking is considered as a creative, human-centred, participative, exploratory and problem-solving process that values different perspectives of a problem (Brown, 2008; Dunne & Martin, 2006; Melles, G. and Mistic, V., 2011). In our research, we adopted the Stanford d.school (D.school, n.d.) process of design thinking which including empathize, define, ideate, prototype, and test. The design action plan is an iterative process, and each action phase should achieve some deliverable outcomes.

(1) Empathize (to understand our users)

Activities: User interview, Observation, Immersion.
Deliverables: Empathy map, list of user feedback, problems identified.

(2) Define (to define clear project objectives)

Activities: Workshops, Stakeholder meetings.
Deliverables: Design brief, stakeholder map, context map, customer map, opportunity map.

(3) Ideate (to explore ideas and solutions)

Activities: Ideation activities, brainstorming, mindmaps, sketching/drawing. Deliverables: Ideas/concepts, sketches, prioritisation map, affinity map, idea evaluation.

(4) Prototype (to build and visualise ideas and solutions)

Activities: Space prototyping, physical prototyping, paper construction, wireframe building, storyboards, role-plays. Deliverables: Physical prototypes, wireframes, storyboards.

(5) Test (to review and decide)

Deliverables: List of user feedback, observation, evaluation

3. RESEARCH PLAN

3.1. Setting and participants

The case setting was seven schools in Beijing, Zibo and Huhehaote, which include 1 primary school, 3 junior schools and 3 high schools, and two classes in each school join the research. Students are grouped in 3-4

person, and are asked to develop a app through collaborative learning based on the design thinking process.

3.2. *Course structure and activities*

The Course consisted of three main modules, lasting 12 weeks in one semester. Each week, we are expected to expend 2 hours of effort in class. Module 1 (6 Weeks): In the beginning, the teacher introduced App Inventor. Participants learned about basic components for building apps and built practice apps. Computational thinking concepts and associated techniques are instructed. Module 2 (2 Weeks): This module introduced participants to the five steps of the design action plan, and participants discuss the topic in groups. Module 3 (4 Weeks): App design. Participants empathize, define and ideate the topic through the design action plan. They need to do some activities, complete deliverable outcome, and draw the sketches of their game. Then, Participants built one practice app, proposed an app of their own and built a working prototype or completed app.

3.3. *Data collection and analysis*

This study adopted a mixed-method approach to collect and analyze the following data: student digital artifacts, classroom observations, survey, test and individual student interviews.

4. CONCLUSION AND FUTURE WORK

The current research employed design thinking to develop a framework of the App Inventor curriculum for cultivating K-12 students' computational thinking. There are a number of future research tasks being considered in the agenda of this study. First, design and implement a K-12 programming curriculum constructed based on the framework. Second, design instruments to assess CT knowledge, skills, and perspectives of learners in the programming curriculum. Third, evaluate the design thinking framework is effective by evaluating the progression of learning outcomes of CT knowledge, skills, and perspectives which include computational identity and digital empowerment.

5. REFERENCES

- [15] Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *AERA* 2012.
- [16] Brown, T. (2008). Design Thinking. *Harvard Business Review*, 86(6), 84–92.
- [17] Dschool, 2015. (n.d.). The Design Thinking Process | ReDesigning Theater. Retrieved from <http://dschool.stanford.edu/redesigningtheater/the-design-thinking-process/>
- [18] Dunne, D., & Martin, R. (2006). Design Thinking and How It Will Change Management Education: An Interview and Discussion. *Academy of Management Learning & Education*, 5(4), 512–523. <https://doi.org/10.5465/AMLE.2006.23473212>
- [19] Ericson, B., & McKlin, T. (2012). Effective and Sustainable Computing Summer Camps. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 289–294). New York, NY, USA: ACM. <https://doi.org/10.1145/2157136.2157223>
- [20] Ioannidou, A. (2011). Computational Thinking Patterns. Online Submission. Retrieved from <https://eric.ed.gov/?id=ED520742>
- [21] Melles, G. and Mistic, V. (2011). Introducing design thinking to undergraduate students at Swinburn university. *Japanese Society for the Science of Design*, (18(1) (69)), 4–9.
- [22] Orr, G. (2009). Computational Thinking Through Programming and Algorithmic Art. In *SIGGRAPH 2009: Talks* (p. 31:1–31:1). New York, NY, USA: ACM. <https://doi.org/10.1145/1597990.1598021>
- [23] Roy, K. (2012). App Inventor for Android: Report from a Summer Camp. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 283–288). New York, NY, USA: ACM. <https://doi.org/10.1145/2157136.2157222>
- [24] Wagner, A., Gray, J., & Wolber, D. (2013). Using app inventor in a K-12 summer camp. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 621–626). <https://doi.org/http://dx.doi.org/10.1145/2445196.2445377>
- [25] Wing, J. M. (2006). Computational Thinking. *Commun. ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- [26] Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717–3725. <https://doi.org/10.1098/rsta.2008.0118>

Computational Thinking and Psychological Studies

Development and Validation of a Programming Self-Efficacy Scale for Senior Primary School Learners

Siu-cheung KONG

The Education University of Hong Kong
sckong@eduhk.hk

ABSTRACT

Programming is one of the important literacies in the digital age. The acquisition of such knowledge and skills is of vital importance to the next generation. This study aimed to develop and validate an instrument to measure programming self-efficacy of senior primary school learners (Grade 4 - Grade 6) in a block-based environment. The proposed scale consisted of two subcomponents related to learners' perceptions of their own competence in (1) programming knowledge and (2) programming skills. In order to assess the validity of the scale, online questionnaires were distributed to 106 primary school students who joined a course of a new programming curriculum. The objective of the curriculum is to nurture young learners to solve daily life problems. The reliability of the scale was good. The confirmatory factor analysis (CFA) supported the validity of the instrument. More specifically, results indicated that the hypothesized measurement model of the scale fit the data collected. It confirmed that the scale was valid and adequate for measuring programming self-efficacy of senior primary school learners. Theoretical and practical implications of this study were discussed at the end of the paper.

KEYWORDS

Programming, Programming self-efficacy, Scale development, Scale validation, Self-efficacy

1. INTRODUCTION

With the fast development of technology today, the younger generation is exposed to the digital world where they not only need to develop "the ability to chat, browse, and interact, but also the ability to design, create, and invent with new media" (Resnick, Maloney, et al., 2009, p. 62) so as to thrive for a better life. Computer scientists see programming as a new literacy that everyone has to acquire in this century, especially for the young (Hutchison, Nadolny, & Estapa, 2015; Vee, 2013). Papert (1980) also pointed out that procedural thinking of children can be fostered by learning programming. Thus, children are highly encouraged to develop programming skills to become creative problem-solvers in the digitalized world.

Evaluating learners' understanding of programming concepts and skills is a major challenge at this stage (e.g., Wang, Li, Feng, Jiang, & Liu, 2012; Yang et al., 2015). Evidence of a validated instrument regarding students' perceptions on programming is even more scarce. It is indisputable that tests of programming knowledge and

skills are fundamental in examining the learning performance (Yang et al., 2015), but it is also important to capture learners' beliefs of their own competence in programming as their performances will be influenced by their self-perceptions. Pajares (1996, p. 543) stated that "the beliefs that individuals hold about their abilities and about the outcome of their efforts powerfully influence the ways they will behave". In order to facilitate the implementation of programming education in senior primary schools, there is a pressing need for academics to design instruments to measure learners' perceptions/attitudes of programming after they are involved in some kind of learning. Based on self-efficacy theory (Bandura, 1986), we developed a programming self-efficacy scale among senior primary school students. According to the theory, when learners have similar level of domain knowledge and skills, their actual performances will be affected by their perceptions of personal efficacy (Bandura, 1986; Zimmerman, 1995). In other words, a person may gain sufficient knowledge and skills for the task, but he/she may fail to achieve desired results due to lack of confidence and motivation. Thus, this scale could predict the performance of the learner (Askar & Davenport, 2009).

2. BACKGROUND OF STUDY

2.1. Programming Self-Efficacy

A definition of programming self-efficacy might first begin with the explanation of self-efficacy. Bandura put forth the self-efficacy theory in the late 1970's. He defined it as "people's judgments of their capabilities to organize and execute courses of action required to attain designated types of performance" (Bandura, 1986, p. 391). Self-efficacy of a person can be obtained from four aspects, including "personal performance accomplishments", "vicarious experience" of observing others' behaviors, "verbal persuasion", and "state of physiological arousal" (Bandura, 1977). There is a strong relation between self-appraisals of capability and performances (Bandura & Adams, 1977; Schunk, 1981). Studies pointed out that personal efficacy influences individual's choice of activities, amount of effort invested, persistence in the face of obstacles, and performance (Bandura, 1977; Schunk, 1989). People with higher self-efficacy are more willing to invest effort to cope with challenging tasks (Bandura, 1994). Self-efficacy is not about the traits of a person, but it is a kind of self-evaluation specific to a particular domain of activities (Bandura, 2006). Therefore, a self-efficacy scale about computer programming should be developed when researchers attempt to investigate

learners' efficacy of computer programming. Based on the theory, programming self-efficacy reflects one's perception and judgment of his/her ability in solving computational problems with programming knowledge and skills. Learners with high programming self-efficacy are more willingly to apply their knowledge and utilize skills to solve computational problems.

Despite the fact that programming is a key ability and literacy that the next generation should acquire, there have been few attempts to create instruments to measure learners' self-efficacy of programming. Ramalingam and Wiedenbeck (1998) pioneered a notable self-efficacy scale for undergraduates in the context of C++ programming language. Some previous studies also adapted this scale to explore learners' self-efficacy of C++ programming (Korkmaz & Altun, 2014; Ramalingam, LaBelle, & Wiedenbeck, 2004) and Java programming (Askar & Davenport, 2009). All these scale items are positive-worded statements, which reflect learners' confidence of their capability of handling programming related concepts and skills. These studies reflect that these are the main components of the scale in measuring learner's efficacy of programming knowledge and skills. However, we need an instrument that focuses on senior primary school learners (i.e. Primary 4 to 6). Since the existing scales were designed for undergraduates, some of the programming concepts are too difficult for primary school learners. In addition, our instrument is designed for a programming curriculum aiming at developing Computational Thinking (CT) through programming where the programming languages used in this study are Scratch and App Inventor.

2.2. Dimensional Structure of a Two-Factor Model

In light of the past literature, the programming self-efficacy scale consists of two components: (1) programming knowledge and (2) programming skills. As our evaluation targets are senior primary school learners, the knowledge and skills that they should acquire are supposed to be simpler than the undergraduates. Brennan and Resnick (2012) studied the programming activities of the kids in the Scratch online community and workshops over a few years to develop a three-dimensional CT framework. Therefore, we also borrowed key ideas from Brennan and Resnick (2012) to the programming self-efficacy scale in the current study. In brief, Brennan and Resnick's framework covers CT concepts, practices, and perspectives. CT perspectives refers to learners' understanding of themselves, their relationships to others and the technology world, which goes in line with our instrument that tries to measure the perception and understanding of programming.

One dimension of programming self-efficacy is programming knowledge. It refers to the programming concepts and knowledge that learners apply in programming. Brennan and Resnick (2012) identified seven basic computational concepts that are commonly used among young novice programmers, including sequences, loops, parallelism, events, conditionals, operators, and data (i.e. variables and lists). In most block-based language assessments, the ability of dealing with loops (e.g., Ericson & McKlin, 2012; Meerbaum-Salant,

Armoni, & Ben-Ari, 2013; Zur-Bargury, Parv, & Lanzberg, 2013), conditionals (e.g., Ericson & McKlin, 2012; Seiter & Foreman, 2013; Zur-Bargury et al., 2013), and variables (e.g., Ericson & McKlin, 2012; Meerbaum-Salant et al., 2013; Seiter & Foreman, 2013) are considered as the fundamental programming knowledge for novice. Researches also emphasized the importance of handling concepts of sequences, and operators (e.g., Seiter & Foreman, 2013). The ability to apply procedure to finish programming tasks is also regarded as basic building blocks of a program. It can be used to avoid repetition of codes and duplicating commands so that novices are able to make the programs more modular and easier to test and debug (Marji, 2014). Thus, understanding of procedure is also suggested to be incorporated into the items of programming knowledge.

The other dimension is CT practices. Programming concepts and CT skills are supposed to be developed in the problem-solving process by using features of a programming environment. In other words, learners not only need to apply programming concepts but also use a variety of skills to tackle computational problems (Olson, Sheppard, & Soloway, 1987). Brennan and Resnick (2012) proposed four sets of practices that are related to the process of solving problems using a programming language, namely being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing. Debugging practice is always regarded as a crucial skill for novices (Hwang, Wang, Hwang, Huang, & Huang, 2008). Apart from these suggestions, other studies argued that planning and designing solutions before programming (e.g., Burke, 2012; Fessakis, Gouli, & Mavroudi, 2013), and algorithmic thinking (e.g., Denner, Werner, Campe, & Ortiz, 2014; Duncan & Bell, 2015; Seiter & Foreman, 2013), which means to design a solution through a series of steps, are indispensable for creating a program. The above concepts and skills are developed and deployed during the process of solving computational problems, yet there must be a problem arisen before they plan and design a solution. It is more crucial to raise questions than to solve problems (Einstein & Infeld, 1938). Consequently, problem formulation should also be perceived as an important component among the skills in solving problems using a programming language.

3. METHOD

3.1. Item Development and Validation

The programming self-efficacy scale was developed in accordance with a comprehensive literature review of programming self-efficacy, and the CT framework proposed by Brennan and Resnick in 2012. Detailed discussions were conducted with a leading research team including professors and researchers from the Education University of Hong Kong to ensure that the language used in each item is understandable to senior primary school learners. The scale consists of two components with 15 items in total (programming knowledge: 7 items; programming skills: 8 items). Item example for CT knowledge is "I have basic knowledge to finish coding tasks". Item example for CT skills is "I can build the code

in an incremental way with a number of iterations". All the items are anchored with 5-point Likert scale, from 1 "unable to master" to 5 "fully master". This study followed Brislin's (1970) suggestion for back-translation. English items were translated into Chinese, and discrepancies were discussed and carefully modified. Finally, we examined the face validity of the scale items for further adaptation into primary school settings.

3.2. Participants and Procedures

Participants of this study had some experience of programming before taking up this survey. The questionnaires were administered to learners in a primary school where the new programming curriculum has been implemented for a semester. The scale is targeted at primary 4, 5 and 6 learners. Online survey was adopted. All the participants filled in the questionnaire since they were asked to finish the survey and submit the answers during class time. In total, 42.5% of the participants were female, and 57.5% of them were male. Among all the participants, 25.5% of them were from Grade 4, 51% were from Grade 5, and 23.5% were from Grade 6. The demographics of the learners are shown in Table 1.

Table 1. The demographics of the participants of the study.

Grade			Gender	
4	5	6	Female	Male
27	54	25	45	61
25.5%	51.0%	23.5%	42.5%	57.5%

4. RESULTS

The measurement structure is confirmed with confirmatory factor analysis (CFA) using Amos 24. Maximum likelihood estimation was used in CFA. $\chi^2(df)$, CFI, TLI, RMSEA were used as the fit indices for the measurement model of programming self-efficacy construct. According to Bentler (1990), CFI and TLI which is greater than .90 suggests a good fit, and greater than .95 suggests an excellent fit. For RMSEA, a cut-off value close to .06 (Hu & Bentler, 1999) or more recently the upper limit of .08 seems to be acceptable among most researchers. In the current study, $\chi^2(87) = 184.78$ ($p < .000$), CFI = .92, TLI = .90, and RMSEA = .10. Both CFI and TLI indicate that the hypothesized measurement model is well fitted with the data collected for the scale development. Although RMSEA is not satisfying, it is probably because the sample size is small for such two-factor model. In addition, all factor loadings are ranged from .59 to .87, further confirming convergent validity of programming self-efficacy. The CFA and reliability of programming self-efficacy scale are shown in Table 2. Figure 1 demonstrates the measurement model of programming self-efficacy.

Table 2. Confirmatory factor analysis and reliability of programming self-efficacy.

Factors and Items	
Factor 1: programming knowledge, $\alpha = .88$	
1. I have basic knowledge to finish coding tasks.	.76
2. I can complete coding by identifying a series of steps in task and solve them subsequently (sequence).	.79
3. I can code with "If...then...else" (conditionals) sentence.	.68
4. I can complete coding tasks with the concept of loop, that is, repeating an action.	.66
5. I can apply variables to finish coding tasks, for example, to set "a" with a number (variable).	.78
6. I can apply operators to finish coding tasks, for example, to use operators such as > (larger than) or < (less than).	.70
7. I can apply procedures to finish coding tasks.	.59
Factor 2: programming skills, $\alpha = .93$	
1. I can overcome the difficulties in coding tasks by dividing them into multiple subtasks and solve them one-by-one.	.74
2. I can test and debug a completed program.	.76
3. I can reuse and / or remix existing codes to build up my own program.	.76
4. I can make an abstraction on a coding task.	.87
5. I can build the code in an incremental way with a number of iterations.	.85
6. I can think of solutions to a computational problem with a series of steps.	.80
7. I can formulate a computational problem from daily life.	.79
8. I can plan and design a solution from a computational problem.	.78

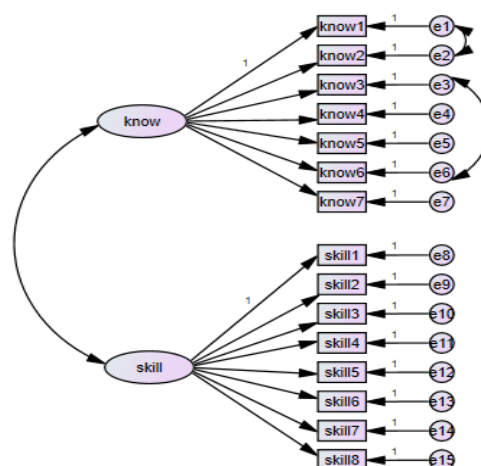


Figure 1. 2-factor model of programming self-efficacy.

In addition, the study also tried to merge the two factors into one single factor by creating a common latent variable such that all 15 items could load on it. CFA results suggested that this alternative model is not as good as the previous two-factor model ($\chi^2(88) = 192.50$ ($p < .000$), CFI = .91, TLI = .89, and RMSEA = .11). Therefore, programming self-efficacy should be better modeled as a two-factor construct as proposed based on the theoretic stance.

5. DISCUSSION

The aim of this study was to develop and validate a programming self-efficacy measure among senior primary school learners. CFA results indicated that the measurement structure has achieved good fit according to the fit indices ($\chi^2(87) = 184.78$ ($p < .000$), CFI = .92, TLI = .90, and RMSEA = .10), though the RMSEA is not that good due to the sample size. The programming self-efficacy as a two-factor model was confirmed. In addition, factor loadings are ranged from .59 to .87, demonstrating the two subcomponents are adequately measuring the latent factor of programming self-efficacy.

5.1. Theoretical and Practical Contributions, and Future Research Directions

The programming self-efficacy scale has theoretical implications for future research. As the next generation is required to acquire the ability to design and create with new media, programming is widely accepted as one of the indispensable literacies in the digital era. Young learners are highly recommended to acquire and master basic programming knowledge and skills so as to become computationally literate learners. Currently, educators examine learners' capability of programming by means of tests and examinations. However, this practice overlooks the significance of learners' perceptions of their abilities in programming. Self-efficacy theory implied that learners' persistence in the face of obstacles and actual performances would be affected by their level of efficacy (Bandura, 1977; Schunk, 1989). Yet, there is no existing scale for investigating self-efficacy of programming among primary school learners. Therefore, programming self-efficacy scale is developed in this study, which would facilitate a deep understanding on primary school learners' self-competence of programming before and after learning programming. The results of the scale might also be used as a tool to predict learners' course performance.

For practical implications, researchers found out that there was a strong correlation between learners' self-efficacy and their actual performance (Bandura & Adams, 1977; Schunk, 1981). Therefore, our future research direction is to explore the relationship between learners' programming self-efficacy and the test results of their programming knowledge and skills. Besides, self-efficacy theory stated that past experience of learning and practicing the skills would influence learners' belief of their competence (Bandura, 1977). Thus, it is expected that learners' self-efficacy will rise because of frequent exposure to the programming course (Ramalingam et al., 2004). Pre- and post-tests can be further conducted to measure learners' self-efficacy over the course. It is a reliable indicator for

researchers to understand the improvements of learners and reflect the effectiveness of the curriculum so as to improve the pedagogy if necessary.

5.2. Limitations

It is also essential to identify the limitations of the current study. According to Bandura (1977), self-efficacy judgements consist of three dimensions, namely magnitude, strength, and generality. In the context of programming, Strength of self-efficacy refers to learners' conviction and confidence of their abilities to complete the programming task. The survey design of this study merely focuses on measuring the strength of self-efficacy by the 5-point Likert scale ranging from "unable to master" to "fully master". In future research, effort have to be put on other dimensions. For instance, the magnitude of self-efficacy can be measured by asking learners to judge their capabilities of completing programming tasks in various difficulty levels (Ramalingam & Wiedenbeck, 1998). Additionally, the generality of self-efficacy can be evaluated by adding items which are about learners' self-assurance of handling tasks in different situations like with or without others' help (Ramalingam & Wiedenbeck, 1998). In our study, in-depth interviews of participants after online surveys were conducted in order to have a more accurate understanding of their attitude and perceptions on programming. Future research studies are encouraged to cover a broader scope of research methods to test/ validate the programming self-efficacy scale.

6. CONCLUSION

Programming is regarded as a new literacy in the twenty-first century. It helps equip young people with the ability of expressing themselves in the digital era (Hutchison et al., 2015). Hence, it is significant for young people to possess basic programming knowledge and skills in the digitalized world. In order to investigate learners' perception of their own competence of programming, this study developed and validated a programming self-efficacy scale for senior primary school learners. The scale is a two-factor model with 15 items in total. The subcomponents are related to learners' belief about their abilities in utilizing different kinds of programming knowledge and skills to solve computational problems. Future research will be conducted to investigate the correlation between learners' programming self-efficacy and the test results of their programming knowledge and skills.

7. REFERENCES

- Askar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for java programming among engineering students. *The Turkish Online Journal of Educational Technology*, 8(1), 23-32.
- Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191-215.
- Bandura, A., & Adams, N. E. (1977). Analysis of self-efficacy theory of behavioral change. *Cognitive Therapy and Research*, 1(4), 287-310.

- Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory*. Englewood, Cliffs, NJ: Prentice-Hall.
- Bandura, A. (1994). Self-efficacy. In V. S. Ramachaudran (Ed.), *Encyclopedia of human behavior* (pp.71-81). New York, NY: Academic Press.
- Bandura, A. (2006). Guide for constructing self-efficacy scales. In F. Pajares, & T. Urdan (Eds.), *Self-efficacy beliefs of adolescents* (pp.307-337). Greenwich, CT: Information Age Publishing.
- Bentler, P. M. (1990). Comparative fit indexes in structural models. *Psychological bulletin*, 107(2), 238.
- Brennan, K., & Resnick, M. (2012, April 13-17). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association* (25 pp.). Vancouver, Canada: American Educational Research Association.
- Brislin, R. W. (1970). Back-translation for cross-cultural research. *Journal of cross-cultural psychology*, 1(3), 185-216.
- Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education*, 4(2), 121-135.
- Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school learners? *Journal of Research on Technology in Education*, 46(3), 277-296.
- Duncan, C., & Bell, T. (2015, November 9-11). A pilot computer science and programming course for primary school students. In *Proceedings of the 10th Workshop in Primary and Secondary Computing Education* (pp. 39-48). New York, NY: ACM.
- Einstein, A., & Infeld, L. (1938). *The evolution of physics: The growth of ideas from early concepts to relativity and quanta*. New York, NY: Simon and Schuster.
- Ericson, B., & Mcklin, T. (2012, February 29-March 3). Effective and sustainable computing summer camps. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 289-294). New York, NY: ACM.
- Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, 63, 87-97.
- Hutchison, A., Nadolny, L., & Estapa, A. (2015). Using coding apps to support literacy instruction and develop coding literacy. *The Reading Teacher*, 69(5), 493-503.
- Hu, L. T., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural equation modeling: a multidisciplinary journal*, 6(1), 1-55.
- Hwang, W. Y., Wang, C. Y., Hwang, G. J., Huang, Y. M., & Huang, S. (2008). A web-based programming learning environment to support cognitive development. *Interacting with Computers*, 20(6), 524-534.
- Korkmaz, Ö, & Altun, H. (2014). Adapting computer programming self-efficacy scale and engineering students' self-efficacy perceptions. *Participatory Educational Research*, 1(1), 20-31.
- Marji, M. (2014). *Learn to program with Scratch: A visual introduction to programming with games, art, science, and math*. San Francisco, CA: No Starch Press.
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, 23(3), 239-264.
- Olson, G. M., Sheppard, S., & Soloway, E. (1987). *Empirical studies of programmers: Second workshop*. Norwood, NJ: Ablex Pub.
- Pajares, F. (1996). Self-efficacy beliefs in academic settings. *Review of Educational Research*, 66(4), 543-578.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.
- Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research*, 19(4), 367-381.
- Ramalingam, V., Labelle, D., & Wiedenbeck, S. (2004, June 28-30). Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 171-175). New York, NY: ACM.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60-67.
- Schunk, D. H. (1981). Modeling and attributional effects on children's achievement: A self-efficacy analysis. *Journal of Educational Psychology*, 73(1), 93-105.
- Schunk, D. H. (1989). Self-efficacy and cognitive achievement: Implications for students with learning problems. *Journal of Learning Disabilities*, 22, 14-22.
- Seiter, L., & Foreman, B. (2013, August 12-14). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 59-66). New York, NY: ACM.
- Ve, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, 1(2), 42-64.
- Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education*, 59, 412-422.

- Yang, T. C., Hwang, G. J., Yang, S. J., & Hwang, G. H. (2015). A two-tier test-based approach to improving students' computer-programming skills in a web-based learning environment. *Education Technology & Society*, 18(1), 198-210.
- Zimmerman, B. J. (1995). Self-efficacy and educational development. In A. Bandura (Ed.), *Self-efficacy in changing societies* (pp.202-231). New York, NY: Cambridge University Press.
- Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013, July 1-3). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 267-272). New York, NY: ACM.

Computational Thinking as a Key Competence – a Research Concept

Amelie LABUSCH¹, Birgit EICKELMANN¹

¹Paderborn University, Germany
amelie.labusch@upb.de, birgit.eickelmann@upb.de

ABSTRACT

Computational thinking is well-known in computer science and is currently entering the field of education. Due to changes in the private and professional life by modern technologies all students are with increasing relevance expected to possess sufficient knowledge in computer-related problem-solving (e.g. Fraillon et al., 2014). The acquisition of key competences related to this assumes an enhancement of knowledge in learning as well as computational thinking processes. Although many concepts for computational thinking education have been created (e.g. Barr & Stephenson, 2011; Krauss & Protsman, 2017), in fact, an evidence-based competence model is not yet available, thereby it represents a significant desideratum.

Considering these aspects, the contribution at hand aims to contribute to this and focuses on the construction and investigation of a model, taken theoretical aspects and the current state of research into account. The principle of this procedure is to break down the term and construct of 'computational thinking' to core elements by working with a literacy approach and presuppose that computational thinking can only be implemented in lessons in a competence-oriented way referring to an evidence-based approach to computational thinking as a key competence of the 21st century. Starting from this, the research presented in this paper describes and explains preliminary work in the context of the preparation of IEA-ICILS 2018 (International Computer and Information Literacy Study). In this context, the authors of this paper are involved as members of the national study center in Germany, which is among other countries taking part in this international study.

KEYWORDS

Computational thinking, ICILS 2018, evidence-based model, key competence of 21st century

1. INTRODUCTION

In the course of the second cycle of ICILS in 2018, the IEA (International Association for the Evaluation of Educational Achievement) for the first time implements the additional option 'computational thinking', by applying computer-based tests for students in Grade 8 (Fraillon et al., in press), wherefore research will be able to clearly examine the interplay between competencies in computational thinking, the students' use of ICT in and outside school as well as individual students' background characteristics (Fraillon et al., in press). In this respect, the structure of theoretical models of computational thinking

can also be reviewed by using representative data sets of Grade 8 students in a number of countries around the world. Furthermore, the study will allow to describe the relationship between students' information literacy and computational thinking and by this will contribute to the question which competencies refer to ICT-literacy (Ainley, Schulz & Fraillon, 2016).

The research presented in this paper starts from the premise that understanding the core elements of computational thinking would enable teachers to integrate it in their teaching concepts. Moreover, understanding and integrating computational thinking in curricula would allow students (K-12) for developing 'computational thinking' as a key competence (see also Barr & Stephenson, 2011) and part of every type of reasoning: "The power of computational thinking is that it applies to every other type of reasoning. It enables all kinds of things to get done: quantum physics, advanced biology, human computer systems, development of useful computational tools" (Barr & Stephenson, 2011, p. 51). This takes into account that enabling students not only to consume but also to create technology is of increasing relevance. The emerging challenge is to make educational systems to react to these new challenges and to make use of the aforementioned power in the most efficient way for all students.

Therefore, the purpose of this contribution is twofold: Firstly, to review the current state of art of conceptualizing and research towards computational thinking (section 2), theoretical aspects (section 3) and explaining a process model to understand the underpinning concept (section 4). Secondly, the paper presents and discusses a more detailed research concept (section 5).

2. CURRENT STATE OF RESEARCH

Wing (2006) stated that computational thinking "represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn" (p. 33). A first point to note is that this assumption is the reason of the great discussion about computational thinking of the last few years. Thus far, hardly any studies researching computational thinking education and measuring computational thinking competences are available. In contrast to this, literature reviews show that there are different concepts of the construct, mostly referring to the first concept, presented by Wing (2006). Barr and Stephenson (2011), for instance, have developed a concept of computational thinking, including several components, e.g. data collection, data analysis, algorithms and procedures etc. In accordance with this and also aiming for making computational thinking teachable in

schools, Krauss and Prottsman (2017) recently published a *teacher's getting-started guide*.

Since these and other concepts are not founded on a universal definition, a generally accepted definition of computational thinking has not been set and finding it “has proved difficult for the CS [computer science] education community” (Mannila et al., 2014, p. 2). However, most of the definitions include core elements or rather processes as part of computational thinking (e.g. Wing, 2008; Lee et al., 2011; Barr & Stephenson, 2011, Krauss & Prottsman, 2017). Therefore, the aim is not to find a universal definition but to create an evidence-based competence model, which clarifies the key competence of computational thinking, while embedding the model into a comprehensive theory. From the perspective of empirical educational research, such a model can be developed by using appropriate tests and develop an evidence-based competence model.

Last but not least, the primary use of Wing’s assumption in the context of this contribution is that computer-linked problem-solving can be reframed to psychological problem-solving processes. Thus, we can benefit from many studies in this area (e.g. Popper, 1994; Marshall, 1995; Robertson, 2001). Román-González, Pérez-González & Jiménez-Fernández (2016) describe “a worrying vacuum about how to measure and asses CT [computational thinking]” (p. 2), wherefore they also suggest a psychometric approach. Relating to this understanding, it can be stated that understanding the process of problem-solving also contributes to clarify computational thinking and to add to computer scientists’ research.

3. THEORETICAL ASPECTS

First, it should be noted that – in contrast to a state-of-the-art computer – a human being is able to solve problems without regulations. This is also taken into consideration when trying to reason the need for conceptualizing computational thinking in the scope of ICILS 2018: “Computers themselves cannot think: they have to be programmed before they can function” (IEA, 2016, p. 2).

From a psychological point of view, processes of computational thinking show many similarities with problem-solving processes, both of which arrive from Bandura’s theory of observational learning (Bandura, 2001). Problem-solving in comparison to performing a task with clear rules is a rather complex process and one way of thinking among others, such as conceptualization and logical reasoning.

In a problem-solving process, the point of departure is a problematic situation, which is followed by situation analysis (Edelmann, 2000). The way in which the brain works in a problem-solving process is much contested among scholars. As a result, and following very early theories, it can be stated that problem-solving is productive thinking (Duncker, 1926). In this context, the aim of problem-solving processes is to close the gap between a problematic situation and the required solution,

using operators, which partly need to be invented by heuristics (Huitt, 1992).

In this context, the use of an algorithm represents a particular form of problem-solving (Kant & Newell, 1984). Compared with humans, a computer can follow orders more quickly but in so doing, it requires algorithms. On an abstract level, the human takes over the thinking process within problem-solving and delegates tasks to a computer in the form of algorithms by analyzing the problem which is needed beforehand and which “requires thinking at multiple levels of abstraction” (Wing, 2006, p. 34).

Pointing out one of the theoretical frameworks, core elements of computational thinking, which are mentioned in different definitions, can be portrayed. The emerging challenge is – as already mentioned – to create an evidence-based model that enables researchers to systemize and based on this, teachers to systematically teach computational thinking – with as well as without applying computer systems and ICT and exploring different ways of transferring and teaching these skills to different contexts.

4. PROCESS MODEL

In the following, figure 1 shows our analytic model of computational thinking processes. Within this model, four core elements and one undefined sub-process, representing unknown sub-processes, which may arise during the research process, are focused:

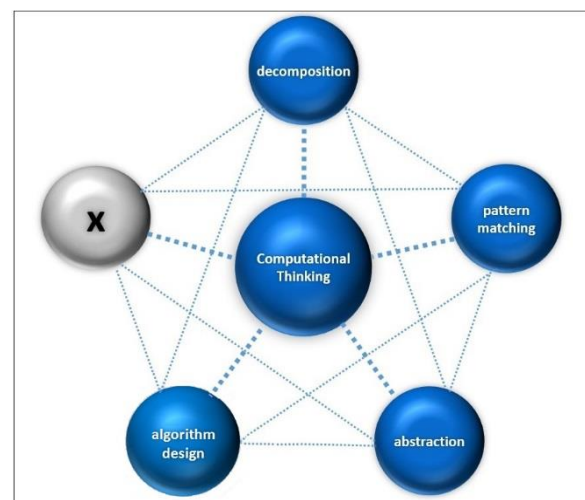


Figure 1: Model of computational thinking processes.

These determinants are assumed to have an impact on computational thinking in general. In the context of computer science, *decomposition* is the ability to subdivide a (complex) structure into fragments. In principle, this is analytic thinking during which a situation or problem is also split into fragments. Decomposition represents a latent variable, such as computational thinking.

Pattern matching also represents an ability and a latent variable. It enables a person to find common features

among and differences between fragments that have been generated by the decomposition process. In this context, Krauss and Prottsman (2017) differentiate between “the practice of finding similarities” (p. 60) (*pattern recognition*) and “the realization that something matches” (p. 60 f.) (*pattern matching*).

Abstraction refers to the ability to eliminate details, reasoning from the particular to the general, also known as induction (ibid.).

Last but not least an *algorithm* provides a guideline for action, which is usually modeled and encoded (ibid.). Producing a guideline for action does not necessarily entail mastering a programming language; it can also be expressed in another way, e.g. in a construction manual or a recipe.

The variable x in the model indicates all sub-processes which are not already mentioned within the model. These processes may include logical reasoning, object-oriented thinking and evaluation and possibly debugging of algorithmic solution. During the ensuing research, the variable x will be substantiated.

The model does not only raise the question of unknown variables but also of correlations between the mentioned sub-processes and computational thinking in general. It has both a heuristic and a didactic function: It should pave the way for further research and facilitate the transfer of knowledge and competencies. In the following, the research concept will be explained.

5. RESEARCH CONCEPT

Taking the current state of art into account, the core elements of computational thinking mentioned above can be reframed to psychological processes and afterwards transferred to competencies in order to generalize these competencies and formulate items. Against this backdrop, we focus the following research question:

1. How can computational thinking be reframed to psychological problem-solving processes and how does this contribute to the understanding of computational thinking as a key competence in the 21st century including different parts of the construct?

Thanks to the IEA, students' computational thinking competencies will be operationalized within ICILS 2018. Based on that, the aim of the research for which a starting point is presented in the scope of this paper, we will add some national extensions to the study to examine the afore-mentioned research questions. Data will be gathered from a representative sample of ICILS 2018, testing 8th grade students (in ICILS 2013, N = almost 60,000 students worldwide) and the German subsample in 2018 will comprise 4,500 students. As already described above, our method provides working at multiple levels using the afore-mentioned model (figure 1). The advantage is, however, that the underlying study ICILS represents a theoretical and empirical basis for the study and helps to develop a meaningful understanding of the interface of existing computational thinking concepts and processes related to general problem-solving skills. Referring to this,

this contribution aims to explain the research approach in a more detailed way and provide a basis for a broader discussion in the scientific community. In this context, we will also provide information to discuss hypothesis about the relationship of computer and information literacy (as conceptualized in ICILS 2018, see for Fraillon et al., 2014) and computational thinking. Following Ainley's and his colleagues' approach (Ainley et al., 2016) the paper aims to discuss how computational thinking can be summarized under the broader understanding of ICT-literacy.

Referring to the premise in the introduction, that understanding several core elements of computational thinking would enable schools and teachers to develop students' competencies in 'computational thinking', this contribution raises attention to the need of developing an evidence-based competence model taking different understandings of computational thinking into account.

To round off this picture, and to prepare for a national extension of ICILS 2018 data collection by taking more general concepts of problem-solving into account, we add a holistic theoretical model and present a research concept to investigate the underpinning theoretical structure.

6. REFERENCES

- Ainley, J., Schulz, W. & Fraillon, J. (2016). *A global measure of digital and ICT literacy skills. Background paper prepared for the 2016 Global Education Monitoring Report*. Paris: UNESCO.
- Bandura, A. (2001). Social cognitive theory: An agentic perspective. *Annual Review of Psychology*, 52(1), 1-26.
- Barr, V. & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48-54.
- Dunker, K. (1926). A qualitative (experimental and theoretical) study of productive thinking (solving of comprehensible problems). *Pedagogical Seminary*, 33(4), 624-708.
- Edelmann, Walter (2000): *Lernpsychologie* (6th ed.). Weinheim: Beltz/ Psychologie Verlags Union.
- Fraillon, J., Schulz, W., Duckworth, D. & Ainley, J. (in press). *ICILS 2018 Assessment Framework*. Amsterdam: IEA.
- Fraillon, J., Ainley, J., Schulz, W., Friedman, T. & Gebhardt, E. (2014). *Preparing for life in a digital age. The IEA International Computer and Information Literacy Study. International Report*. Amsterdam: International Association for the Evaluation of Educational Achievement (IEA).
- Huitt, W. (1992). Problem solving and decision making: Consideration of individual differences using the Myers-Briggs Type Indicator. *Journal of Psychological Type*, 24(1), 33-44.
- IEA (2016). *The IEA's International Computer and Information Literacy Study (ICILS) 2018. What's next for IEA's ICILS in 2018?* Retrieved February 6, 2017 from

- http://www.iea.nl/fileadmin/user_upload/Studies/ICILS_2018/IEA_ICILS_2018_Computational_Thinking_Leaflet.pdf.
- Kant, E. & Newell, A. (1984). Problem solving techniques for the design of algorithms. *Information Processing and Management*, 20(1), 97–118.
- Krauss, J. & Prottsman, K. (2017). *Computational Thinking and Coding for Every Student. The Teacher's Getting-Started Guide*. Corwin Press Inc.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32–37.
- Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L. & Settle, A. (2014). Computational Thinking in K-9 Education. *ACM*, 1–29.
- Marshall, S. P. (1995). *Schemas in problem solving*. Cambridge University Press.
- Popper, K. (1994). *Alles Leben ist Problemlösen. Über Erkenntnis, Geschichte und Politik*. München: Piper.
- Robertson, S. I. (2001). *Problem Solving*. Hove, UK: Psychology Press.
- Román-González, M., Pérez-González, J.-C. & Jiménez-Fernandez, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, 1-14.
- Wing, J. M. 2006 Computational thinking. *Commun. ACM*, 49(3), 33–35.

Can Music Exposure Enhance Computational Thinking? Insights from the Findings on the Music-Creativity Relations

Mei-ki CHAN¹, Wu-jing HE^{1*}, Wan-chi WONG²

¹ The Education University of Hong Kong, Hong Kong

² The Chinese University of Hong Kong, Hong Kong

meiki@eduhk.hk, mavishe@eduhk.hk*, wanchiwong@cuhk.edu.hk

ABSTRACT

This study aimed to uncover the underlying mental processes that might facilitate creative thinking after listening to a 10-min music excerpt. A qualitative component was incorporated in a quantitative study regarding the effect of music listening on creative thinking. Among 192 participants, a total of 24 college students were interviewed immediately after they listened to the 10-min music excerpts and completed some creativity tasks. The results suggested a possible facilitative role of music listening on creative thinking through optimizing individuals' arousal level, and strengthening individuals' associative abilities, holistic perception and abstraction. As these mental processes are also important attributes of computational thinking (CT), the findings may shed light on the favourable impact of music listening on CT.

KEYWORDS

Computational thinking, creative thinking, music listening, qualitative study, emotion

1. INTRODUCTION

The effect of music exposure on intellectual functioning has become an important research topic since the findings about the Mozart effect (He, Wong, & Hui, in press), which suggest that listening to music composed by Mozart leads to significant improvements in cognitive functioning (e.g., Rauscher, Shaw, & Ky, 1993). Commonly conceptualized as consisting of originality and appropriateness (Sternberg & Lubart, 1999), creativity has become one of the important foci in recent educational reforms (Hui & Lau, 2010). A better understanding on how music exposure correlates with creativity has important implications. Although there are many studies reporting a positive effect of music listening on creativity (e.g., Schellenberg, 2006), it remains unclear what underlying mental processes contribute to such a positive effect. The present study aimed to understand the impact of music exposure on creative thinking through a qualitative approach with the aims to obtain data to reveal the underlying mental processes that might contribute to the positive music-creativity relationship.

In addition to creative thinking, Computational Thinking (CT) is also an essential skill in nearly all fields (Bundy, 2007). The term CT originates from studies of computer sciences, referring to the mental processes consisting of constructing problems and solutions which can be carried out by human or machine (Wing, 2006). Although CT and creative thinking seem to relate to different disciplines, interestingly, the two terms has been suggested to contain

similar constructs, (DeSchryver & Yadav, 2015). On the ground of the explorative nature of a qualitative approach, this study also aimed to explore the possible effect of music listening on CT. It may extend our understanding on the beneficial effect of music listening to a new avenue.

2. METHOD

The present study was a subcomponent of a larger research project on music listening and thinking skills. Among the 192 college students being recruited to join the project from two universities in Hong Kong, twenty-four participants (22 females) agreed to accept a follow-up interview. All participants signed consent and took part on a voluntary basis. The age range of the interview sample was 17–27 (*Mean* = 20.75; *SD* = 2.56).

Semi-structured interviews were conducted with the participants after they listened to the 10-min music excerpts and completed two creative thinking tests, namely the The Test for Creative Thinking – Drawing Production (TCT-DP; Urban & Jellen, 1995/2010) and the Torrance Tests of Creative Thinking (TTCT; Torrance, 1974). The music excerpts were extracted from ‘Butterfly Lovers Violin Concerto’, which have been shown to be able to induce positive or negative emotions (Zhang & Chen, 2009). Each interview was tape-recorded with the permission of the participant and lasted approximately 15–20 minutes. They were asked explicitly to explain the effects of music listening on their performance on the tasks.

3. RESULTS AND DISCUSSION

Transcriptions of the interviews were analyzed following the procedures of content-analysis suggested by Graneheim and Lundman (2004). The results showed that four themes were identified, namely a) optimal arousal level, b) associative abilities, c) abstraction and d) holistic perception.

3.1. The Effect of Music Exposure on Creativity

First, in terms of optimal arousal level, the interviewees reported that in an aroused state, they could “think flexibly,” or “have more inspirations.” This is in line with previous studies, which suggest that an optimal activation level is critical in promoting creative thinking (Schellenberg, Nakata, Hunter, & Tamoto, 2007). Second, the interviewees indicated that their associative flexibility and fluency were strengthened after the intervention. The findings on the associative abilities echo the long proposed assumption of associative process playing a vital role in creativity (Mednick, 1962). Third, the interviewees also

reported that music listening was conducive to a broader attention scope and a more holistic perception. This finding is consistent with past studies that suggest a positive relationship between creativity and the breadth of attention (Kasof, 1997). Fourth, the interviewees described that their thinking became more abstract, so that it was easier for them to find commonalities between the given objects in the creativity task and those objects found in one's daily life, giving evidence to the involvement of abstraction in creativity (Welling, 2007). The results suggested that music exposure might have a favourable impact on creative thinking through four important mental processes.

3.2. The Possible Beneficiary Effects of Music on CT

It has been suggested that creative thinking and CT comprise similar mental processes (DeSchryver & Yadav, 2015). The findings of this study may shed light on the possible beneficial effect of musical listening on CT. The interview data obtained in this study suggest that music listening can facilitate creative thinking through optimal arousal, abstraction, attention scope, and holistic perception. Such mental processes might contribute to CT, which is characterized by symbol representation, parallel thinking, synthesis and abstraction (Barr, Harrison & Conery, 2011; Wing, 2006). According to Wing (2006), for instance, abstraction skills, an important component in CT, enables pattern observation, identifies core features and omits details of perceptions. On the ground of music facilitating the underlying mental processes of CT, music listening will probably be beneficiary to CT, implying the potential benefits of incorporating music into CT education.

3.3 Limitations and Merits

Some limitations should be noted. First, the exploration into the mental process subsequent to music listening is based on subjective knowledge of the participants. Second, the sample size of the interview was relatively small. Further studies should be conducted to explore if the findings can be generalized to a wider population.

Despite the mentioned limitations, this study is the first qualitative study that addresses the research question on why and how music listening can enhance creative thinking. The explorative nature of the interviews depicts the important mental processes that may facilitate creative thinking. As these mental processes are important attributes of CT, the findings of the present study extend the discussion from creativity to the possible beneficial effect of musical listening on CT. The finding extends our understanding of the impact of music exposure on creativity to revealing the thought processes involved in engaging in a creative task, as well as CT.

4. REFERENCES

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38, 20-23.

- Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1, 67-69.
- DeSchryver, M. D., & Yadav, A. (2015). Creative and computational thinking in the context of new literacies: working with teachers to scaffold complex technology-mediated approaches to teaching and learning. *Journal of Technology and Teacher Education*, 23, 411-431.
- Graneheim, U. H., & Lundman, B. (2004). Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness. *Nurse education today*, 24, 105-112.
- Hui, A. N., & Lau, S. (2010). Formulation of policy and strategy in developing creativity education in four Asian Chinese societies: A policy analysis. *The Journal of Creative Behavior*, 44, 215-235.
- Kasof, J. (1997). Creativity and breadth of attention. *Creativity Research Journal*, 10, 303-315.
- Mednick, S. A. (1962). The associative basis of the creative process. *Psychological Review*, 69, 220-232. doi:10.1037/h0048850
- Rauscher, F.H., Shaw, G.L., & Ky, K.N. (1993). Music and spatial task performance. *Nature*, 365, 611.
- Schellenberg, E. G. (2006). Long-term positive associations between music lessons and IQ. *Journal of Educational Psychology*, 98, 457.
- Schellenberg, E.G., Nakata, T., Hunter, P.G., & Tamoto, S. (2007). Exposure to music and cognitive performance: tests of children and adults. *Psychology of Music*, 35, 5-19.
- Sternberg, R. J., & Lubart, T. I. (1999). The concept of creativity: Prospects and paradigms. *Handbook of creativity*, 1, 3-15.
- Torrance, E. P. (1974). *The Torrance Tests of Creative Thinking-Norms-Technical Manual Research Edition-Verbal Tests, Forms A and B-Figural Tests, Forms A & B*. Princeton, NJ: Personnel Press.
- Urban, K. K., & Jellen, H. G. (1995/2010). *Test for Creative Thinking – Drawing Production (TCT-DP). Manual*. Frankfurt am Main, Germany: Pearson Assessment & Information GmbH.
- Welling, H. (2007). Four mental operations in creative cognition: The importance of abstraction. *Creativity Research Journal*, 19, 163-177.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49, 33-35.
- Zhao, H., & Chen, A. C. (2009). Both happy and sad melodies modulate tonic human heat pain. *The Journal of Pain*, 10(9), 953-960.

Acknowledgement: This study was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No: 28605615)

Computational Thinking and Early Childhood Development

Imagining, Playing, and Coding with KIBO: Using Robotics to Foster Computational Thinking in Young Children

Amanda A. SULLIVAN^{1*}, Marina UMASCHI BERS², Claudia MIHM³

The DevTech Research Group^{1 2 3}

Tufts University, United States^{1 2 3}

Amanda.Sullivan@tufts.edu, Marina.Bers@tufts.edu, Claudia.Mihm@tufts.edu

ABSTRACT

The KIBO robotics kit offers a playful and tangible way for young children to learn computational thinking skills by building and programming a robot. KIBO is specifically designed for children ages 4-7 years old and was developed by the DevTech research group at Tufts University through nearly a decade of research funded by the National Science Foundation. KIBO allows young children to become engineers by constructing robots using motors, sensors, and craft materials. Children also become programmers by exploring sequences, loops, and variables. Through programming KIBO, children engage with computational thinking skills and ideas including algorithms, modularity, and control structures. Unlike other programming interfaces for children, the KIBO robot is programmed to move or to respond to sensor input by using tangible programming blocks—no computer, tablet, or screen-time required. This paper provides an overview of the design features of KIBO and a synthesis of the research that has been done throughout the development of this kit. It provides examples of curriculum for playfully engaging young children with computational thinking using KIBO.

KEYWORDS

Early childhood, engineering, robotics, programming, computational thinking

1. INTRODUCTION

Early childhood is an important time for young children to grow, play, and explore the world they live in. Developmentally, it is a life stage characterized by genuine curiosity and desire for learning. In order for young children to master new knowledge about the world, they need hands-on experiences to construct their learning (Piaget, 1936). New technologies such as robotics kits and coding applications offer children a hands-on way to learn about many of the things they encounter every day but do not understand, such as sensors, batteries, and lights (Papert, 1980). Robotics is an ideal tool for early childhood because it facilitates cognitive as well as fine motor and social development (Bers, 2008; Clements, 1999; Lee, Sullivan, & Bers, 2013; Svensson, 2000). It engages children creatively, as an expressive medium, allowing young children to become engineers by playing with motors and sensors as well as storytellers by creating and sharing personally meaningful projects that react in response to their environment (Bers, 2017; Bers 2008).

When learning to build and program robots, young children are also engaging in a type of problem solving and analysis called *computational thinking*. The term “computational thinking” can be defined as solving problems algorithmically and developing a sense of technological fluency (Bers, 2017; Bers, 2010; Papert, 1980) Children as young as four years old can learn foundational computational thinking concepts (Bers, 2017; Bers, 2008) and this kind of learning can support their literacy, mathematical, and socio-emotional development (Kazakoff & Bers, 2012; Kazakoff, Sullivan, & Bers, 2013). While computational thinking is rooted in computer science, many have argued that it is a universally applicable attitude and skillset that is fundamental for everyone to master, just like reading, writing, and arithmetic (Wing, 2006).

KIBO (see Figure 1) was born out of research led by Marina Bers at the DevTech Research Group at Tufts University (Bers, 2017). The goal was to foster playful exploration of computational thinking during early childhood through tangible objects. Later on, KIBO became commercially available through KinderLab Robotics with funding from the National Science Foundation and a successful Kickstarter campaign (Bers, 2017). KIBO’s design was based on years of child development research in collaboration with teachers and early childhood experts to meet the learning needs of young children in a developmentally appropriate way (Sullivan, Elkin, & Bers, 2015; Sullivan & Bers, 2015; Kazakoff & Bers, 2014). This paper provides an introduction to the design of KIBO and presents an overview of the worldwide research conducted with KIBO for the last several years to promote computational thinking in young children.



Figure 1. KIBO robot with a sample block program, art platforms, and art supplies for decorating.

2. DESIGN FEATURES OF KIBO

KIBO is a robotics construction kit that involves both hardware (the robot itself) and software (tangible

programming blocks) used to make the robot move. The kit contains easy to connect construction materials including: wheels, motors, light output, and sensors as well as a variety of art platforms (See Figure 1 on the previous page).

KIBO is programmed using interlocking wooden programming blocks (see Figure 2). These wooden blocks contain no embedded electronics or digital components. Each wooden block has a colorful label with an icon, text and a bar code; as well as a hole on an end and a peg on the other. The KIBO robot has an embedded scanner that allows users to scan the barcodes on the programming blocks and send a program to their robot instantaneously. No computer, tablet, or other form of “screen-time” is required to learn programming with KIBO. This is aligned with the American Academy of Pediatrics’ recommendation that young children have a limited amount of screen time per day per day (American Academy of Pediatrics, 2016).



Figure 2. KIBO’s tangible programming language. Each block has a unique barcode that is scanned by the robot.

This programming language was inspired by early ideas from tangible programming beginning with Radia Perlman’s work in the mid 1970’s (Perlman, 1976) and revived by the work of Suzuki & Kato (1995) nearly two decades later. In recent years, there have been several tangible languages have been created in a number of different research labs around the world (e.g. McNeerney, 2004; Smith, 2007; Horn & Jacob, 2007).

In contrast to graphical programming, which relies on pictures and words on a computer screen, tangible programming uses physical objects to represent the same concepts (Manches & Price, 2011). Wooden programming blocks are naturally familiar and comfortable for children, in the tradition of learning manipulatives already used in early childhood classrooms to teach shapes, size, and colors (Froebel, 1826; Montessori & Gutek 2004). KIBO’s programming blocks are shared easily and manipulated by young users with limited fine motor capacity.

KIBO’s programming language is composed of over 18 individual wooden programming blocks. Some of these blocks represent simple motions for the KIBO robot such as, move Forward, Backward, Spin, and Shake. Other blocks represent complex programming concepts such as Repeat Loops and Conditional “If” statements that involve sensor input (See Figure 3).



Figure 3. This figure provides an example of a conditional statement with KIBO.

3. EARLY COMPUTATIONAL THINKING

3.1. What is Computational Thinking?

In recent years, there has been a growing focus on improving children’s technological literacy and making computational thinking a priority in early childhood school settings in the United States (e.g. U.S. Department of Education, 2010). According to Wing (2006) computational thinking is defined as, “solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (p.33). *Computational thinking* involves a set of skills that include problem-solving, design and expression, and systematic analysis (Bers, 2017; Bers, 2010). Computational thinking represents a type of analytical thinking that shares many similarities with mathematical thinking (e.g., problem solving), engineering thinking (designing and evaluating processes), and scientific thinking (systematic analysis) (Bers, 2017).

Brennan & Resnick (2013) explain that computational thinking involves the *concepts* designers engage with as they program, the *practices* designers develop as they engage with the concepts, and the *perspectives* designers form about the world around them and about themselves. Concepts may include very specific programming concepts (such as repeat loops or conditional statements), the practices may include methods of problem-solving or collaboration, and perspectives may include questioning things beyond the interface you are working with (such as questioning how other things in the world are automated, besides KIBO). Bers (2017) expands on the notion of computational thinking, describing it not only as a problem solving process, but as an *expressive process*; a skillset that allows for new ways to communicate, to tell stories and convey ideas.

It is important to note that there are many non-technical and even non-academic examples of instances that call for computational thinking skills (Wing, 2008; Yadav, 2011). These everyday activities draw on the same type of problem solving, but do not involve programming. Wing (2008) presents a series of examples including: sorting Legos (using the concept of “hashing” to sort by color, shape, and size), learning to cook a meal (using “parallel processing” to manage cooking at different temperatures for different amounts of time) and looking up your name in an alphabetical list (linear: starting at beginning of the list, binary: starting at the middle of the list). Each of these examples are activities young children are beginning to encounter in their everyday lives.

3.2. Fostering Computational Thinking with KIBO

KIBO is designed to promote a specific set of computational thinking skills. KIBO aims to foster seven “powerful ideas” of computational thinking described by Bers (2017). These ideas include: 1) algorithms, 2) modularity, 3) control structures, 4) representation, 5) hardware/software, 6) the design process, and 7) debugging. Table 1 below describes these concepts and how children explore them with KIBO. In the following section we provide examples of curricular units that foster these computational thinking concepts in a hands-on and playful way.

Table 1. Computational Thinking Concepts Explored with the KIBO Robotics Kit

Concept	Examples
Algorithms	Children use KIBO to explore logical organization and sequencing using the tangible programming blocks
Modularity	Children learn how to break up a large job into smaller steps when programming KIBO to navigate mazes or complete challenges
Control Structures	Children explore the ways KIBO can make decisions based on conditions using Repeat Loops and Conditional Statement blocks
Representation	Children learn that the colors and symbols on the blocks represent different types of actions
Hardware & Software	Children learn that computing systems, like KIBO, need both hardware (robotic parts) and software (blocks) to operate
Design Process	Children move through an iterative process used to develop programs and tangible artifacts
Debugging	Children troubleshoot their code when KIBO does not behave as expected

4. KIBO CURRICULUM

While the act of coding often evokes a very serious image of someone quietly working through lines of code on a computer, KIBO offers a more playful approach that is aligned with the spirit of early childhood education. Play in early childhood is not just fun; research has shown that it enhances children’s capacity for cognitive flexibility and, ultimately, creativity (Russ, 2004; Singer & Singer, 2005).

The DevTech Research Group has developed over a dozen curriculum units that focus on playful learning with KIBO in order to teach the computational thinking skills listed in the previous section. These curricular units also focus on STEAM (Science, Technology, Engineering, Arts, and Mathematics) content integration. In this section, we provide three examples of STEAM curriculum designed for KIBO: *Dances from Around the World*, *Robotic Animals*, and *Patterns All Around*. These illustrate how

KIBO can be used to explore computational thinking while teaching other STEAM content such as dance, social studies, and math.

4.1. Dances from Around the World

The *Dances from Around the World* (DevTech, 2015) unit is designed to combine music, culture, dance, and language with programming and engineering content. The end project involves children programming their KIBOs to perform their favorite dance from anywhere in the world. It is completed over the course of approximately seven weeks. Each week, teachers introduce new robotics and programming concepts, from basic sequencing through conditional statements, to their students within the curriculum’s music and dance theme. For the final project, students work in pairs or small groups to design, build, and program a dance of their choosing. This involves not only robotics and programming knowledge, but also research into the music, history and cultural relevance of the dance, and facts about the country or culture in which the dance originated. The unit culminates in a dance recital for both the children and the robots to perform in together. Children engaged with open-ended free-play time to listen to their chosen music and come up with a dance on their own.

While this project engaged children with all of the seven powerful ideas of computational thinking described in Table 1, children had to devote particular focus on the idea of **sequencing** when choreographing and programming their robot dances. They had to carefully consider the timing of the music and any traditional dance steps that needed to be included (and if so, in what order). They needed to program their robot’s actions in a sequential order that matched the order of the dance they choreographed for them to perform. Most students also had to explore **control structures**, learning how to use KIBO’s Repeat Loop commands in order to ensure their robot dances repeated the appropriate number of times to match the music.

4.2. Robotic Animals

Integrating the natural sciences with robotics and engineering, in the *Robotic Animals* curriculum (DevTech, 2015), children explore animals and their natural habitats. After choosing an animal and researching its behavioral and physical characteristics, students create a robotic representation of that animal and its habitat for their final projects (See Figure 4).



Figure 4. The image (left) shows final project examples from the *Robotic Animals* curriculum.

When building and programming their robotic animals, children grappled with the concept of **hardware and software**. They learned that to create an effective robot that looks, moves, and reacts like a cat or wolf, they

needed to understand and use KIBO's hardware elements such as motors, sensors, and wheels as well as the right software, or program, to make the robot move the way the animal does. Children moved through an iterative **design process**, building their physical robot structure and made improvements to its sturdiness and aesthetic features. They also moved through an iterative process developing their programs.

4.3. Patterns All Around

The Patterns All Around unit (DevTech, 2015) focuses on an explicit exploration of math through KIBO. This unit integrates mathematics with fundamental engineering and programming concepts. Throughout the curriculum, students learn about different types of patterns using mathematics. They also explore other foundational math skills such as counting, shape recognition, and more. As a final project, students then have the opportunity to create a class "quilt" using large pieces of posterboard. By attaching a pen or crayon to KIBO, they were able to complete hands-on programming challenges where they were prompted to program KIBO to draw specific shapes or create different types of patterns on paper. This unit also offered many opportunities for free play and artistic exploration.

In this unit, students explored the computational concept of **modularity**, or breaking down a large task into a series of smaller steps. While programming complex patterns was often a daunting task for the kids, their teachers prompted them to focus on programming just one part at a time. After coming up with a series of short programs, children were able to put it all together and **debug**, or troubleshoot if it still did not look quite right.

5. RESEARCH WITH KIBO

5.1. Methods

During the research and development of KIBO, we have collected quantitative and qualitative data and published findings from $N=322$ children and $N=32$ early childhood teachers over the course of dozens of studies looking at what children have learned about robotics, engineering, sequencing, and more using KIBO (See Table 2 on the following page). Our research has been conducted across the United States, in Denmark, and as part of a large-scale study in Singapore (Sullivan & Bers, 2017). In order to measure children's mastery of computational concepts, the DevTech Research Group developed the "Solve-Its" assessment (Strawhacker, Sullivan, & Bers, 2013; Strawhacker & Bers, 2014). Solve-Its entail listening to different stories or songs being read or sang aloud by a researcher. After listening to the story or song, the Solve-Its prompt children to arrange paper blocks into a sequential program that matches what they heard (See Figure 6). Each task assesses a different computational concept such as control flow or sequencing.



Figure 6. The image (above) shows an example of a child completed Solve-It task assessing their knowledge of repeats.

Highlights from this work are summarized in the following section. For a full list of publications detailing our studies with KIBO and to find out about the materials we have developed including teacher surveys, interview protocols, observation protocols, behavioral checklists, and more please visit: <http://ase.tufts.edu/devtech/publications.html>

Table 2. Summary of Topics Researched with KIBO

	Sample	Study Instruments
Sequencing	$N=27$	Baron-Cohen et al. picture sequencing cards
Computational Thinking	$N=28$	Solve-Its Debugging Assessment
Robotics & Programming Knowledge	$N=60$	Solve-Its Robot Parts Task
Gender	$N=45$	Solve-Its Interviews
Coding in Preschool	$N=64$	Solve-Its Observations
Teachers	$N=32$	Teacher surveys Interviews
Positive Technological Behaviors (PTD)	$N=98$	PTD Checklists
Total	$N=354$	

5.2. What Do Children Learn?

Our research has shown that learning to program with tangible robotics kits allows young children to practice sequencing, logical reasoning, and problem solving skills, along with positive behaviors such as collaboration and communication (Kazakoff, Sullivan, & Bers, 2013; Bers, 2015; Sullivan & Bers, 2015). In addition, we have shown that children as young as 4 years old can master powerful ideas from computational thinking and early engineering (Bers, 2017).

In a study with children in pre-kindergarten through second grade ($N = 60$) using a prototype of the KIBO robotics kit, results showed that beginning in pre-kindergarten, children were already able to master basic robotics and programming skills (Sullivan & Bers, 2016). This same study also demonstrated that older children were able to master increasingly complex concepts using the same kit in the same amount of time (Sullivan & Bers, 2016). Based on these findings, DevTech's most recent

study with KIBO focused explicitly on the pre-kindergarten years and what these very young children are capable of building and creating (Elkin, Sullivan, & Bers, 2016). In this study, with 64 children from seven preschool classrooms, findings indicated that although KIBO was originally designed for ages 4 and up, children as young as age 3 could create syntactically correct programs for KIBO (Elkin, Sullivan, & Bers, 2016).

These findings demonstrated that KIBO embodies the “high ceiling/low floor” approach to technology design. This means it is easy to get started with KIBO (i.e. “low floor”), in this case, even for children as young as 3 years old. But there is also a “high ceiling” (i.e. a lot of complex possibilities) for what you can do with KIBO as you get older and gain more mastery for the concepts. Resnick, et al. (2005) also describes the idea of “wide walls” saying that “tools should support and suggest a wide range of explorations.” In order to address this, the DevTech Research Group has created over a dozen curriculum units, such as the three described in the previous section, that explore the ways that robotics can be integrated across a variety of domains. Our research has demonstrated that it is not just children who need support and materials: teachers do too. We have seen that early childhood teachers need training, support, and resources in order to feel confident and competent teaching robotics (Bers, Seddighin, & Sullivan, 2013). Therefore, we have now made training videos, curriculum units, and other resources freely available on the Early Childhood Robotics Network (www.tkroboticsnetwork.ning.com).

5.3. Computational Thinking

A big piece of our research on computational thinking has focused on the impact of robotics and computer programming on young children’s sequencing skills. Sequencing, a key aspect of computational thinking outlined by Bers (2017), is also an important pre-math and pre-literacy skill for early childhood found in both curricular frameworks and learning assessments (Kazakoff, Sullivan, & Bers, 2013).

Our research has demonstrated that beginning in pre-kindergarten, learning to program a robot significantly improves children’s ability to logically sequence picture stories (Kazakoff, Sullivan, & Bers, 2013). This suggests that the sequencing skills gained through programming can be translated to sequencing things beyond code, such as stories.

In a recent study by Pugnali, Sullivan, & Bers (under review) the authors have begun to explore the impact of user interface on children’s computational thinking skills, comparing the tangible KIBO programming language to a graphical tablet-based programming language. This study found that children in the tangible KIBO group scored significantly higher on two key aspects of computational thinking: sequencing and debugging. While further research is required, this may suggest that the tangible nature of KIBO’s block language may make it more accessible to young children than onscreen languages.

6. CONCLUSION

The KIBO kit is being used by a growing number of children, parents, teachers, schools, camps, museums, and after school programs all around the world. Since its launch in 2014, KIBO is now used in 48 states across the U.S. as well as 43 countries worldwide. Countries such as Singapore are now using KIBO on a widespread basis to address technological literacy in the early childhood years (Sullivan & Bers, 2017). The research summarized here demonstrates the power of a tool like KIBO to effectively teach computational thinking beginning as early as pre-school and kindergarten. It also highlights the many ways that robotics and computer programming can easily integrate into traditional early childhood domains such as math, science, and social studies. Moreover, the work done with KIBO over the past five years has shown the possibilities for teaching computational thinking without forgetting that young children are still young children. Learning to code should not come at the sacrifice of learning to play and socialize. The curriculum units developed for KIBO have demonstrated successful ways to teach coding while still engaging in physical movement, listening to music, dancing, and collaborating. All of these are key components of a well-rounded early childhood experience.

7. REFERENCES

- American Academy of Pediatrics (2016). Media and young minds. *Pediatrics*, 138(5).
- Bers, M.U. (2017). *Coding as a Playground: Programming and Computational Thinking in the Early Childhood Classroom*. Routledge press.
- Bers, M. (2008). *Blocks to Robots: Learning with Technology in the Early Childhood Classroom*. Teachers College Press, NY, NY
- Bers, M.U., Seddighin, S., & Sullivan, A. (2013). Ready for robotics: Bringing together the T and E of STEM in early childhood teacher education. *Journal of Technology and Teacher Education*, 21(3), 355-377.
- Brennan, K., & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.
- DevTech (2015). *Dances from Around the World Curriculum*. Retrieved from: tkroboticsnetwork.ning.com
- DevTech (2015). *Robotic Animals Curriculum*. Retrieved from: tkroboticsnetwork.ning.com
- DevTech (2015). *Patterns All Around Curriculum*. Retrieved from: tkroboticsnetwork.ning.com
- Elkin, M., Sullivan, A., & Bers, M.U. (2016). Programming with the KIBO Robotics Kit in Preschool Classrooms. *Computers in the Schools*, 33(3), 169-186
- Clements, D. H. (1999). Young children and technology. In G. D. Nelson (Ed.), *Dialogue on early childhood science, mathematics, and technology education*. Washington, DC: American Association for the Advancement of Science.

- Froebel, F. (1826). On the Education of Man (Die Menschenerziehung), Keilhau/Leipzig: Wienbrach.
- Horn, M. S., & Jacob, R. J. (2007). Tangible programming in the classroom with tern. In *CHI'07 extended abstracts on Human factors in computing systems* (pp. 1965-1970). ACM.
- Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia*, 21(4), 371-391.
- Kazakoff, E., Sullivan, A., & Bers, M.U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, 41(4), 245-255.
- Lee, K., Sullivan, A., Bers, M.U. (2013). Collaboration by design: Using robotics to foster social interaction in Kindergarten. *Computers in the Schools*, 30(3), 271-281.
- Manches, A., & Price, S. (2011). Designing learning representations around physical manipulation: hands and objects. In *Proceedings of the 10th International Conference on Interaction Design and Children* (pp. 81-89). ACM.
- McNerney, T. S. (2004). From turtles to tangible programming bricks: Explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5), 326-337.
- Montessori, M., & Gutek, G. L. (2004). *The Montessori method: The origins of an educational innovation: including an abridged and annotated edition of Maria Montessori's the Montessori method*. Lanham, MD: Rowman & Littlefield Publishers.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- Piaget, J. (1936). *Origins of intelligence in the child*. London: Routledge & Kegan Paul.
- Perlman, R. (1976). *Using computer technology to provide a creative learning environment for preschool children*.
- Pugnali, A., Sullivan, A., & Bers, M.U. (under review). The impact of user interface on young children's computational thinking. *Journal of Information Technology Education: Innovations in Practice*.
- Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., & Eisenberg, M. (2005). Design principles for tools to support creative thinking. *The School of Computer Science at Research Showcase at Carnegie Mellon University*.
- Russ, S.W. (2004). *Play in child development and psychotherapy*. Mahwah, NJ: Earlbaum.
- Singer, D.G. & Singer, J.L. (2005). *Imagination and play in the electronic age*. Cambridge, MA: Harvard University Press.
- Smith, A. C. (2007). Using magnets in physical blocks that behave as programming objects. In *Proceedings of the 1st international conference on Tangible and embedded interaction* (pp. 147-150). ACM.
- Sullivan, A., & Bers, M.U. (2015). Robotics in the early childhood classroom: Learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education*. Online First.
- Sullivan, A. & Bers, M.U. (2017). Dancing Robots: Integrating Art, Music, and Robotics in Singapore's Early Childhood Centers. *International Journal of Technology & Design Education*.
- Sullivan, A., Elkin, M., & Bers, M. U. (2015). KIBO Robot Demo: Engaging young children in programming and engineering. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)*. ACM, Boston, MA, USA.
- Suzuki, H., & Kato, H. (1995). Interaction-level support for collaborative learning: AlgoBlock—an open programming language. In *The first international conference on Computer support for collaborative learning* (pp. 349-355). L. Erlbaum Associates Inc..
- Svensson, A. K. (2000). Computers in school: Socially isolating or a tool to promote collaboration? *Journal of Educational Computing Research*, 22(4), 437-453.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-36
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881), 3717-3725.
- Yadav, A. (2011). Computational Thinking and 21st Century Problem Solving.

Programming with ScratchJr: a review of the first year of user analytics

Kaitlyn D. LEIDL^{1*}, Marina UMASCHI-BERS², Claudia MIHM³

The DevTech Research Group^{1 2 3}

Tufts University, United States^{1 2 3}

Kaitlyn.Leidl@tufts.edu, Marina.Bers@tufts.edu, Claudia.Mihm@tufts.edu

ABSTRACT

ScratchJr is a free programming application for young children ages 5-7, available for most tablet devices. This programming environment, developed by the DevTech Research Group at Tufts University, the Lifelong Kindergarten Group at MIT, and the Playful Invention Company, was launched in July, 2014. During the first year after the app's launch, no information was collected regarding usage other than informal communication with local educators and parents. Starting in January 2016, the ScratchJr team began to use the tool Google Analytics to gain a deeper insight into user behavior, and began to investigate the learning analytics data that could shed light on computational thinking in early childhood. This paper presents the first year of user data collection of ScratchJr.

KEYWORDS

Computational thinking, programming, early childhood, educational technology, analytics.

1. INTRODUCTION

ScratchJr is a free tablet app that provides an introductory programming environment for young children ages 5-7. It was developed as a collaboration between the DevTech Research Group at Tufts University, the MIT Lifelong Kindergarten Group, and the Playful Invention Company, with funding from the National Science Foundation (DRL-1118664). ScratchJr was first launched as a freely downloadable app on iPads in July, 2014, and has since been released for use on several other platforms including Android tablets, Amazon tablets, and Chromebooks. Used in classrooms and homes worldwide, ScratchJr enables children to create interactive stories and games by snapping together graphical programming blocks to make characters move, jump, dance, and sing. As shown in Figure 1, the ScratchJr interface allows children to use blocks that control motion, looks, sound, character communication, and more. Through these programming blocks, young children learn the basic concepts and powerful ideas of coding while creating personally meaningful projects (Bers, 2017). The programming app has been widely available for over two years, and in that time, educators, parents, and children around the world have used it to expand the range of creative programming projects and to connect coding to traditional school subjects such as science, mathematics, literacy, history, and more (Bers & Resnick, 2015).



Figure 1. ScratchJr programming app interface

In January, 2016, the ScratchJr team was able to integrate Google Analytics to examine how people are using the programming app. This tool has allowed the team to better understand when and where ScratchJr is being used, which programming blocks are most popular, how many projects are being created, and how long users spend during sessions with ScratchJr. Data collected for a year provides insights into early coding and computational thinking.

2. COMPUTATIONAL THINKING IN EARLY CHILDHOOD







ScratchJr was developed to encourage all young children to engage in computational thinking while coding. Within the open-ended programming environment, children learn the basic powerful ideas of computer science, such as algorithms, debugging, and modularity by snapping together programming blocks. While programming in ScratchJr, children think creatively, logically, and sequentially (Bers, 2008, 2012, 2017). Computational thinking has the potential to benefit all individuals as it involves understanding sequencing and order, as well as logical thinking. This type of thinking is involved in many everyday tasks, such as learning the steps to ride a bike, following a recipe, or editing and rewriting a research paper (Bers, 2017; Wing, 2006).

When computational thinking is supported at a young age by teaching children about coding, it has the potential to supplement and solidify many other social and behavioral skills, which will be valuable to society whether or not the child becomes an engineer or a computer scientist in the future (Wing, 2006). Therefore, we designed ScratchJr to be a developmentally appropriate programming language to engage children in computational thinking, and to provide a space for them to encounter powerful ideas from computer science (Bers, 2017).

3. ScratchJr PROGRAMMING APP

ScratchJr can be described as a technological “playground” for young children (Bers, 2012). They are encouraged to learn by experimenting, to try out new programming blocks, to express themselves creatively and artistically, to tell stories, and to collaborate with peers while having fun. When the ScratchJr app is opened, users are prompted to create a new project, open an existing project, or explore various learning resources (Bers & Resnick, 2015). Once a user is on the project screen, there is no one right way to begin coding with the available programming blocks. Users have the opportunity to explore the block categories and interface features by testing them out and “tinkering” with different options (Flannery et al., 2013).

Users can start by dragging programming blocks into the programming area, snapping them together using their puzzle piece-like features to create a program sequence

Category	View in ScratchJr	Function
Triggering Blocks		Start a program
Motion Blocks		Move characters
Looks Blocks		Change characters' appearance
Sound Blocks		Play or record sound
Control Blocks		Control parts of a program
End Blocks		End a program

(see Figure 1). There are six categories of programming blocks: Triggering Blocks, Motion Blocks, Looks Blocks, Sound Blocks, Control Blocks, and End Blocks (see Figure 2) (ScratchJr, 2017).

Figure 2. ScratchJr programming blocks

Users can add different characters and backgrounds to their project, or create their own using the Paint Editor Tool. This feature was intended to enhance the personalization of projects, as children can edit existing characters and backgrounds, or completely create their own from their imagination (Strawhacker, Lee, Caine, & Bers, 2015). When characters are added, users are free to explore different block options, and to create programs for their characters by snapping the blocks together in the programming area. Users can create code with just motion blocks, or move on to more complex concepts such as making their characters communicate with each other via message blocks. Users can also create interactions between characters using unique triggering blocks like “Start on Bump,” where one character will not start their program unless another character physically bumps into them. This open-ended, “low floor and high ceiling” programming environment design makes ScratchJr approachable for young children and novice programmers alike, as it is easy to start programming by trying out different features, yet there is still room to grow in program complexity (Flannery et al., 2013).

4. METHODS

4.1. Google Analytics Tool

To better understand how and where children and adults use ScratchJr, and, how often they program with it, the ScratchJr team uses Google Analytics. Google Analytics is a free tool developed by Google Inc. in 2005 that gives small or medium-sized companies or teams insights on users' behaviors to understand areas for improvement (Google Inc., 2016; Luo, Rocco, & Schaad, 2015). The program acquires information about how ScratchJr is being used by installing a "cookie" on devices that download ScratchJr from the respective app store. Cookies are small bits of information that are stored on devices, without personally identifiable information (Clark, Nicholas, & Jamali, 2014; Google Inc., 2016).

As noted by other researchers using the Google Analytics tool to gain insight on users' behavior, "Google Analytics...makes it easy to identify patterns and trends in user behavior by combining specific dimensions and metrics to be investigated and plotting the results in its pre-formatted or customized reports," (Luo et al., 2015, p. 265).

There are four main categories within Google Analytics that the ScratchJr team uses to investigate user activity:

1. **Real-Time:** Displays user activity as it happens in real-time on the ScratchJr app. Allows the team to monitor the number of people using ScratchJr at a given time, their geographic locations, which pages they are on within the app, and which app version they are using.
2. **Audience:** Provides information about how many individuals use ScratchJr, how many sessions have occurred, the average time a user spends in ScratchJr, which devices have downloaded ScratchJr, which languages these devices are set to, and where in the world ScratchJr is used.
3. **Acquisition:** Gives insight into how many new users begin programming with ScratchJr.
4. **Behavior:** Includes information about which screens are used most often, which programming blocks and characters are used in ScratchJr and how often, and screen-flow within the app.

Google Analytics organizes the data received from unique devices' cookies and IP addresses into data that can be visualized in line graphs, bar graphs, pie charts, flow charts, and map overlays (see samples of data visualization in Figure 3). This practice of data visualization allows quantitative figures about ScratchJr users to be better understood by the team.

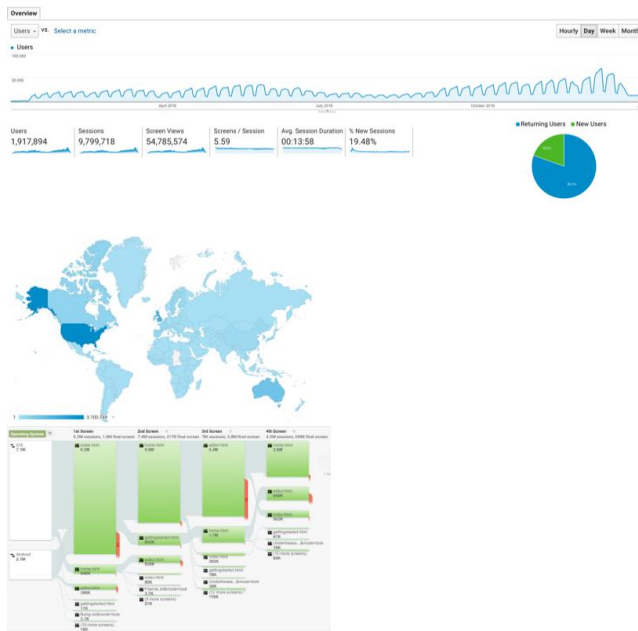


Figure 3. Google Analytics visualization data for ScratchJr

4.2. Using Analytics in Education

Data analytics tools can be used for a myriad of reasons. Large companies and small businesses alike often turn to data collection tools to redefine marketing strategies, increase revenue, and utilize user behavior patterns to improve overall user experience (Luo et al., 2015; Martin et al., 2015).

However, more recently in research, examples of studies around analytics data have emerged in the realm of education. In this context, the practice is known as learning analytics, and focuses on learners, the process of learning over time, and the context in which learning takes place (Baker & Inventado, 2014; Berland, Martin, Benton, Petrick Smith, & Davis, 2013; Luo et al., 2015).

As more data about learners becomes available due to the increased amount of and access to online courses, educational traffic on the web, and educational software and technology, more opportunities to expand educational research have subsequently emerged (Greller & Drachsler, 2012). Being able to transform the abstract progression of learning into tangible numbers and visual data gives researchers insight into learning patterns that could have major implications on the way educators teach core subjects in schools (Greller & Drachsler, 2012).

There have been several recent studies that use learning analytics to track how individuals learn how to program (Baker & Inventado, 2014; Berland et al., 2013; Blikstein et al., 2014). A common method of gathering data is taking screenshots of students' code generation over a period of time (Berland et al., 2013; Blikstein et al., 2014). Researchers can use computer algorithms to categorize these screenshots in terms of programming development by asking questions such as: how did the code change in complexity, length, and content over time? How did these changes impact the effectiveness of the programs overall? Did the later programs indicate growth in programming

knowledge? (Berland et al., 2013; Blikstein et al., 2014). By using computer programs to quantify learning curves among students while they learn programming languages, researchers are uncovering learning patterns that could have major implications on how we teach computer science in educational institutions (Blikstein et al., 2014).

Using data analytics in ScratchJr, we have gained insight into how the number of users, sessions, and locations has evolved over the course of one year. In this paper, we report these results.

4.3. ScratchJr in Google Analytics

Since January, 2016, the ScratchJr team has utilized the Google Analytics program to gain a better understanding of how ScratchJr is used across the globe. Although tools like Google Analytics are often used by businesses to track revenue and improve marketing strategy (Google Analytics Solutions, 2017; Luo et al., 2015), in the case of ScratchJr, our focus is on user behavior, location, patterns in new user acquisition, and the app features themselves. To protect the privacy of our young users, we do not collect personally identifying information, such as unique project content. Therefore, using Google Analytics alone, we cannot track the progression of project content and programming behaviors of individual users over time. Instead, we focused on data points presented and defined in Table 1:

Table 1. Data points and definitions (Google Inc., 2016)

Name of Data Point	Definition of Data Point
Session	The period time a user is actively engaged with the website, app, etc.
Users	Users that have had at least one session within the selected date range. Includes both new and returning users.
Returning Users	A user with existing Google Analytics cookies from a previous visit.
New Users	The number of first-time users during the selected date range. A new user is one who did not have Google Analytics cookies when they first opened the app. If a user deletes their cookies and re-opens the app, they will be counted as a new user.
Average Session Duration	The average length of a session.
Events	The categories that were assigned to triggered events.
Language	The language settings in the users' browsers. Analytics uses ISO codes.
Location	The location from which the session originated.
Real-Time	Data updates continuously and each pageview is reported seconds after it occurs. Shows the number of people on the app right now, their geographic locations, etc.

Through Google Analytics, the team collects data on where users click or tap within the app, which parts of the app users use, and geographic location of where the app is being used based on device IP addresses and network location. The “click data” helps the team determine ways to improve both the app interface and available learning and teaching resources. The geographic location data helps to understand where ScratchJr is and is not being used. ScratchJr does not share any specific user information it collects with Google, and Google does not collect any personally identifying information about users.

Since January, 2016, the ScratchJr team has been gaining insight into how users, both adults and children alike, use the app. Google Analytics allows teams to see reports from any date range, and view data in terms of hours, days, weeks, and months. This allows the ScratchJr team to refine data regarding time in meaningful ways. For example, it is possible to determine which month, week, day, or hour is the most popular time to use ScratchJr, in terms of both how many users are active at those times, and how many sessions occur in those times. This has been especially useful to gauge the impact of computer science and programming education events that occur around the world in which ScratchJr is present. In observing the hourly and weekly data patterns of when ScratchJr is used, we can infer if the majority of children are programming with ScratchJr in classrooms with educators or in their homes with family.

Throughout 2016, the ScratchJr team discovered several notable patterns in behavior of ScratchJr users, and has subsequently begun to make steps towards improving the app and its resources.

5. FINDINGS

Overall, the average amount of sessions and number of users increased as the year progressed, yet other data points such as average session duration, percentage of new users per week, and users per week remained consistent. These are all telling data points regarding user loyalty to the programming app. The analytics highlights from 2016 are described in the following subsections.

5.1. Users & Sessions

There were nearly 2 million total ScratchJr users in 2016. There were more than 104,000 average active users per week, and nearly 27,000 average users on Thursdays alone, the most popular day to use ScratchJr in 2016. Only slightly more sessions occurred on Thursdays in 2016 than on Fridays (see Figure 4).

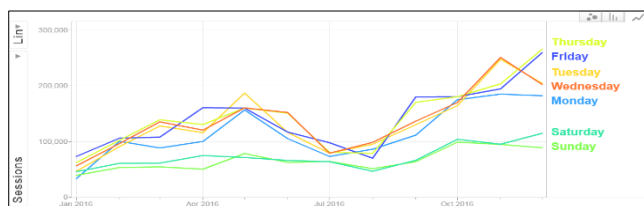


Figure 4. The most sessions occurred on Thursdays

The number of sessions on each of those days of the week came to over 1.7 million. The time of day that saw the

most sessions in ScratchJr was 9:00 AM EST (7.83% of the total sessions occurred during this hour). Spikes and patterns in weekly and hourly users are shown in the graphs in Figures 5 and 6. Consistently, 20% of users each week were new to ScratchJr, and 80% were returning users.



Figure 5. Google Analytics ScratchJr users daily view: peaks tend to be Thursdays and Fridays, low points tend to be Saturdays and Sundays.



Figure 6. Google Analytics ScratchJr users hourly view: peaks tend to be at 9:00 AM EST and 2:00 PM EST; low points tend to be at 11:00 PM EST and 12:00 AM EST.

There was an average of nearly 37,000 new users to ScratchJr each week in 2016. The week that recorded the most new users was December 4-10, 2016, with nearly 97,000 (see spike on right side of graph in Figure 7). This week was “Computer Science Education Week” in the United States, in which government officials encouraged engagement in programming in classrooms, and websites like Code.org provided numerous resources for learning how to code, including ScratchJr lesson plans (Code.org, 2016; Computer Science Education Week, 2016; The White House, 2016). Furthermore, the DevTech Research Group at Tufts University created ScratchJr videos teaching pillars of computational thinking, or “powerful ideas” (Bers, 2017; Papert, 1980), which were viewed hundreds of times, indicating a definite presence of the programming app in the United States throughout the week (DevTech Research Group, 2016).



Figure 7. New Users per week in 2016; spike on right side of graph indicates Computer Science Education week in the U.S., which brought many new users to ScratchJr.

In 2016, there were nearly 9.8 million recorded sessions in ScratchJr. The average session duration was 13 minutes and 58 seconds. Users averaged viewing 5.6 screens per session. The most common flow of screens for both iOS and Android operating systems began with the Index screen that appears when users first open the app, followed by the Home Lobby screen, then the Editor to create programs, followed by the Home Lobby screen again, and then the Editor again. A smaller percentage of users went from the Index screen to the “Getting Started” screen to learn how to use ScratchJr.

5.2. Programming Projects Content

The year 2016 saw over 7.5 million projects created in ScratchJr. Furthermore, there were over 9 million existing projects edited, showing that users tend to go back into projects to work on them. There were 254,000 ScratchJr projects shared via either email or Apple AirDrop in 2016.

In 2016, there were nearly 148 million ScratchJr programming blocks added by users to the programming area in the app. The ten most popular programming blocks added were the Forward block (25 million added), Start on Green Flag, Move Up, Move Back, Say (a speech bubble block that allows characters to converse), Record Block (allows users to record their own sounds and add them into their program), Move Down, Shrink, Turn Right, and Grow. The least popular blocks were Reset Size, Send Message, Start on Message, Start on Bump, and Stop (Figure 8).

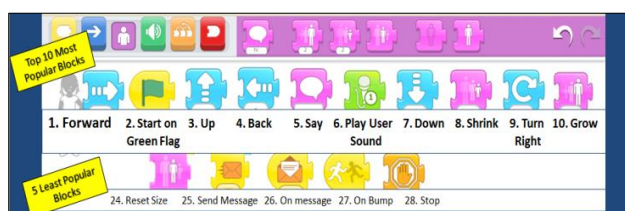


Figure 8. Most and least popular ScratchJr blocks in 2016

The most popular characters used by children in 2016 were those self-created or edited by the children in the Paint Editor, the Child, the Teen, Tac, and the Dragon (Figure 9). Users entered the Paint Editor to customize their characters and backgrounds over 23 million times.



Figure 9. Most popular characters in ScratchJr in 2016

The ScratchJr app includes eight sample projects to provide examples of programs users can make. These sample projects were viewed 1.6 million times in 2016.

5.3. Location & Language

In 2016, ScratchJr was used in all 50 states in the United States of America, and in all but five countries worldwide. The top 10 countries using ScratchJr based on the number of sessions recorded are displayed in Table 2.

The top language codes on devices using ScratchJr include English-US, English-Great Britain, English, English-Australia, Spanish-Spain, English-Canada, Swedish-Sweden, French-France, Korean-Korea, and Finnish-Finland.

Country	% of Total Sessions
United States	31.65%
United Kingdom	17.35%
Australia	10.32%
Canada	4.33%
Sweden	3.30%
Spain	3.16%
Finland	2.52%
France	2.28%
South Korea	2.24%
China	2.10%

Table 2. Top nations using ScratchJr

6. CONCLUSION

In using Google Analytics, the ScratchJr team is able to understand user behavior in a quantitative way. The team has been able to better comprehend the global reach of ScratchJr, using location and language statistics to determine the best methods for localization of ScratchJr. In learning that ScratchJr was used in 191 of 196 registered countries worldwide in 2016, the importance of and demand for computer science education across the globe became clear.

Furthermore, the tremendous growth in numbers during Computer Science Education Week in December, 2016 is an indication that ScratchJr was a popular vessel for learning about computer science and programming when classrooms reserved the time to teach the topics. This gives the team reason to continue making resources available for educators and parents, particularly during national and global initiatives to promote computer science.

Data that the ScratchJr team has gathered about when ScratchJr is used also gives a unique insight into how to support users. Thursdays and Fridays were the two most popular days for ScratchJr in 2016, and the most popular time of day was around 9:00 AM EST. This suggests teachers are using ScratchJr on a weekly basis towards the end of the week and in the mornings. The ScratchJr team could use this information to promote educational resources, tips, and ideas for ScratchJr at these times.

Although the ScratchJr team does not collect individual projects and thus cannot currently see the learning progression of users programming in ScratchJr, there are many insights we can still gain by having visual and numerical data. Based on the data we collected in 2016, it is clear that educators, parents, and children are finding ways to learn programming, and ScratchJr has the potential to be one of the leading platforms young children use to engage in computational thinking.

7. FUTURE WORK

Moving forward, the ScratchJr team will continue to use Google Analytics to better understand user behavior. The team will use the data gathered to optimize localization efforts, and provide resources based on project content trends. Using data regarding the most popular days and times of day ScratchJr is used, the team will use social media outlets to support educators who may be teaching with the programming app at those times, and continue to build a ScratchJr community for users to share their ideas and experiences. Furthermore, the team will develop surveys to gather data that is not currently collected to be able to start inferring learning trajectories.

8. REFERENCES

- Baker, R. S., & Inventado, P. S. (2014). Educational data mining and learning analytics. In *Learning analytics* (pp. 61-75). Springer New York.
- Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, 22(4), 564-599.
- Bers, M. (2008). *Blocks to Robots: Learning with Technology in the Early Childhood Classroom*. Teachers College Press, NY, NY.
- Bers, M.U. (2012). *Designing digital experiences for positive youth development: From playpen to playground*. Cary, NC: Oxford.
- Bers, M.U. (2017). *Coding as a Playground: Programming and Computational Thinking in the Early Childhood Classroom*. Routledge Press.
- Bers, M.U. & Resnick, M. (2015). *The Official ScratchJr Book*. San Francisco, CA: No Starch Press.
- Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4), 561-599.
- Clark, D. J., Nicholas, D., & Jamali, H. R. (2014). Evaluating information seeking and use in the changing virtual world: the emerging role of Google Analytics. *Learned Publishing*, 27(3), 185-194.
- Code.org (2016). *Hour of Code*. Retrieved from <https://code.org/learn>.
- Computer Science Education Week (2016). *Computer Science Education Week*. Retrieved from <https://csedweek.org/>.
- DevTech Research Group (2016). *DevTech Celebrated Computer Science Education Week!* Retrieved from <http://ase.tufts.edu/DevTech/CSEdWeek2016.html>.
- Flannery, L.P., Kazakoff, E.R., Bonta, P., Silverman, B., Bers, M.U., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 1-10. DOI=10.1145/2485760.2485785
- Google Analytics Solutions (2017). *Success Stories*. Retrieved from https://www.google.com/analytics/success-stories/?product=analytics#?modal_active=none.
- Google, Inc. (2016). Google Analytics Solutions. Retrieved from: <https://www.google.com/analytics/>.
- Greller, W., & Drachsler, H. (2012). Translating Learning into Numbers: A Generic Framework for Learning Analytics. *Educational Technology & Society*, 15 (3), 42-57.
- Luo, H., Rocco, S., & Schaad, C. (2015, October). Using Google Analytics to Understand Online Learning: A Case Study of a Graduate-Level Online Course. In *2015 International Conference of Educational Innovation through Technology (EITT)* (pp. 264-268). IEEE.
- Martin, T., Petrick Smith, C., Forsgren, N., Aghababayan, A., Janisiewicz, P., & Baker, S. (2015). Learning fractions by splitting: Using learning analytics to illuminate the development of mathematical understanding. *Journal of the Learning Sciences*, 24(4), 593-637.
- Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas*. New York, Basic Books.
- ScratchJr (2017). *Learn Page*. Retrieved from <http://scratchjr.org/>.
- Strawhacker, A., Lee, M., Caine, C., & Bers, M.U. (2015). ScratchJr Demo: A coding language for Kindergarten. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)*. ACM, Boston, MA, USA.
- The White House (2016 December 5). FACT SHEET: A Year of Action Supporting Computer Science for All. *The White House Office of the Press Secretary*. Retrieved from <https://obamawhitehouse.archives.gov/the-press-office/2016/12/05/fact-sheet-year-action-supporting-computer-science-all>.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-36.

Technology Strategy Mapping in My First Skool Childcare Centres, Singapore

Ai-ling THIAN, Belinda CHNG, Meei-yen LONG

My First Skool, NTUC First Campus, Singapore

thianailing@myfirstskool.com, belinda.csm@myfirstskool.com; longmeeiyen@myfirstskool.com

ABSTRACT

This paper shares the background, vision, the eclectic approach and sharing of the technology-supported initiatives introduced for young children in the network of 129 My First Skool (MFS) childcare centres in Singapore. Focus is given to the communication of a widespread initiative to bring multiple technologies to young children in a large network of schools and systemic provisions for teacher equipping to use these technologies.

KEYWORDS

early childhood, technology strategy, professional development, affordances, play-based learning

1. INTRODUCTION

This paper shares the overarching eclectic approaches for the adoption of technology for teaching and learning with an aim to introduce a sustainable and scalable technology-infused teaching practice to the network of 126 My First Skool early-childhood centres. Our centres provide childcare service to children from birth to age 6, and our centre size can range from about 100 to 500 children. Since 2015, My First Skool centres had intentionally embarked on the journey to leverage technology for teaching and learning through the adoption of tech-enabled toys and focus on professional development. Through these efforts, MFS strives to lay a firm foundation for centres to integrate technology into the curriculum.

2. BACKGROUND

The pervasiveness of mobile devices such as smartphones, laptops and tablets has made “screens” a ubiquitous part of our lives. Guidelines published in 1999 by the American Academy of Pediatrics (AAP) and adopted by Media Development Authority of Singapore to limit “screen time” to two hours a day for children over 2 years old are evolving. AAP recognised that “screen time” is becoming simply “time” now. The “screen time” rhetoric that used to accompany the television is no longer relevant. Key messages surfaced from AAP’s conference in 2015 evolved around parents’ awareness of the negative effects of passive “screen time” and the strategies to balance the use of “screens” so as to reap the benefits of interactive media.

Against the backdrop where 98% of Singapore household with school-going children have Internet access (IDA, 2014) and that Singapore is ranked highest globally for

smartphone penetration (Deloitte, 2015), young children’s exposure to interactive media cannot be ignored.

A study by National University of Singapore called Project iBaby (NUS, 2014) found that nine out of 10 children in the 18 to 24 months age group are exposed to screen devices. Another study published by the Asian Parent (Samsung, 2014) showed that across South East Asia, Singapore has the highest number of children age 3 to 8 years old using a parent-owned device.

These studies pointed out an inevitable need for MFS to move our conversation from an “if” we are going to introduce technologies and interactive media that are developmentally appropriate for young children into our classrooms to “how” and “why” we are going to do it in the most responsible and impactful way.

In an increasingly technology-rich environment, young children are progressively exposed to various technology devices for communication, leisure and learning. While there is growing concern from educators and parents about excessive screen time and the lack of kinesthetic and social interactions, there is no denial that these tech-tools provide much enriched information and experiences to the children. Hence, there is a need to skillfully harness technology to provide more positive learning.

3. VISION

The vision for the technology strategy mapping in MFS centres is to empower educators with the skills and knowledge to leverage on the affordances of technology to create “a joyful and inspiring early learning experience for all, which fulfils the promise of each child”.

4. APPROACHES

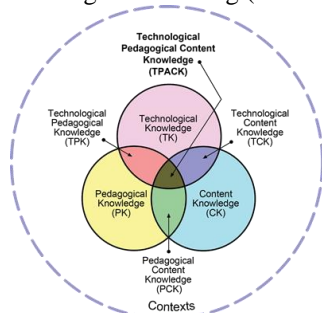
The introduction of technology into teaching and learning is never about adding gadgets or tools into the classroom. Educators are required to make informed choices about the use of technology for young learners through the lens of known child development theories and developmentally appropriate practice. Making informed choices about the use of technology and interactive media requires knowledge, experience, and active exploration.

The following approaches are aimed to provide educators the opportunities to acquire the necessary skills and knowledge to review, assess, design, refine and reflect how technologies could support, facilitate, or even play a key role in day-to-day activities/lessons. The exposure to different uses of technology and interactive media for teaching and learning should build educators’ confidence

and ease to infuse them with an intentionality to grow and develop our young children.

Approach 1: Integrating technology for curriculum and pedagogy looks at the professional development of teachers which enable them to be confident in using technology and gain the skills and knowledge to apply the appropriate practices for teaching and learning.

Technological, Pedagogical and Content Knowledge (TPACK) is a framework developed by Dr. Koehler and Dr. Mishra from the College of Education at Michigan State University to understand and describe the kinds of knowledge needed by an educator for effective pedagogical practice in a technology enhanced learning environment (Koehler, 2011)(Refer to the figure below). We leverage in this framework to understand the possible effects and practical applications of use of TPACK construct on teachers' levels of technology integration in learning and teaching (Koehler & Mishra, 2009)



Used widely by educators around the world including the Singapore Ministry of Education (e.g. <http://www.nie.edu.sg/project/oer-12-10-khl>), TPACK will guide our training and workshops for educators to make creative links between what is being learned (content), how it is taught (pedagogy), and the appropriate tools (technology).

Approach 2: Research, Innovate and Scale - seeks to nurture a culture of innovation and reflective practice across centres. To do this, it is important for our educators to work with external partners (such as government agencies, universities, etc.) and participate in experimentation and innovation efforts. This approach will allow educators to engage in professional discourse, learn, explore, reflect, prototype tools/devices/approaches that could deepen practice and improve their craft. The initiatives introduced to educators should provide opportunities for our educators participate in innovative projects within Singapore and beyond.

Approach 3: Connected Learning Ecosystem - looks to build educators' confidence in using technology within their learning environment to guide, facilitate or engage young learners. A connected technological infrastructure will provide flexibility and agility for policies on quality teaching and learning with tools and devices to be implemented quickly and cost effectively. The initiatives introduced could shape the learning ecosystem and make technology an integral part of the physical environment.

MFS Technology-Led Initiatives

Initiative 1:

The PlayMaker Programme in MFS - partnership with Infocomm & Media Development Authority (IMDA) of Singapore

The PlayMaker Programme is the piloting of the use of tech-enabled toys for 5 to 6 years old to elicit creativity, problem-solving skills and innovation mindset - typical 21st century competencies we want to teach our young. Guided by "*tinkering*" through play-based learning and the larger concept of "learning by doing", IMDA piloted the PlayMaker Programme with all the 5 anchor operators in 2016.

As one of the 5 anchor operators, 33 MFS centres were selected to participate in the PlayMaker Programme over 2 phases between Jan – Dec 2016. Funded by IMDA, each centre was supported with 4 sets of developmentally appropriate tech-enabled toys worth \$6,000.

Workshops were conducted to train educators how to use these toys and to learn about the educational design behind these toys. Sample lesson plans were shared to scaffold and guide the learning activities at these workshops. To ease the introduction of these tech-enabled toys into classrooms, IMDA also provided the on-site consultancy to the participating centres.

As participants of this programme, MFS centres were required to develop lesson plans for these toys as part of an inquiry project for our children. Educators would also be invited to share their lesson plans and experience at the PlayMaker Symposium after 6 months.

To support our educators and enable the culture of sharing, a Professional Network Learning Community is formed within MFS. The PlayMaker PNLC aimed to provide peer reviews and support across the 33 centres in the planning, designing and development of lesson plans and activities. A Lead Team consisting of 2 Principals and 2 Senior Teachers is formed to guide, monitor and co-ordinate the learning and sharing across participating centres.

Through the PlayMaker Pilot programme, young children in MFS were provided with the opportunity to learn technology through tactile and kinesthetic educational experiences through the introduction of this suite of child-friendly, technology-enabled toys. Through touch and feel, and learning how to play and how to use it, children can build up their creative confidence while being familiarise with technology from a young age.

Two Learning & Sharing Festivals were also organised in 2016 to offer a platform for all 33 centres to share their PlayMaker Pilot progress and learning.



Figure I - the MFS Learning & Sharing Festivals

Initiative 2:

The PlayMaker Research Programme - partnership with Dr. Marina Bers, Tufts University sponsored by IMDA (from Jan to June 2016)

Guided by the same principles as the PlayMaker Pilot Programme, the Research Programme with Prof Marina Bers studied the impact of introducing developmentally-appropriate robotics in early childhood in the Singapore context, aimed at drawing out the pedagogical practices and benefits, as well as the support and structures needed to bring about the desired child development and learning outcomes.

One of our MFS centres, Westgate Centre (WGC) was selected as one of the 5 centres working directly with Prof. Marina Bers – creator of Kibo, for a 9-weeks research study on the impact of tech-enabled toy in a pre-school setting. WGC received both training for Kibo and 10 sets of the Kibo tech-enabled toy.

As participants of this research programme, Westgate centre worked closely with the research consultants, adapted and delivered a set of lesson plans designed with the intentionality to teach sequencing to children aged 5 and 6. Sequencing is a fundamental component of computational thinking and sequencing skills are predictors of academic success in math and literacy. Our teachers conducted weekly 1-hr lessons with Kibo over a course of 9 weeks and submitted their weekly reflection logs and engagement checklists to Prof. Marina Bers.

WGC was exposed to innovative pedagogical approaches from Prof. Marina Bers and participated frequently in professional discourse with educators at both the national and international levels.

IMDA has commissioned a research with Eliot-Pearson Department of Child Study and Human Development, Dev Tech Research Group at Tufts University, on “*Dancing robots: integrating art, music, and robotics in Singapore’s early childhood centers*” and Prof Marina Bers had also

highlighted the PlayMaker Programme at the White House Symposium for Early STEM in 2016.

Initiative 3:

The Apple Lighthouse Project - partnership with Apple Singapore

The Lighthouse approach aimed to introduce the use of user-friendly, intuitive Apple technologies in centres that would complement classroom activities and enhance engagement with young learners. The Lighthouse approach aimed at (i) encouraging and building teachers’ experience to use Apple technology for different activities and; (ii) equipping centres with a seamless environment to practice and use technology efficiently and effectively. These classroom experiences and practices were shared across participating centres in a Professional Learning Network Community (PNLC) to encourage and inspire the educators.

The critical success factors for this project is to get teachers (i) comfortable with the technologies that were introduced and; (ii) to be creative and leverage on the affordances of these tools they have within their classrooms to deliver impactful activities for their children.

A total of 8 MFS centres were identified to champion this effort and participate in this project with Apple Singapore. Educators attended training workshops conducted by Apple trainers, learnt about design principles for technology infused lessons, features and tools available on Apple devices that supports learning for young children or classroom aides that facilitate learning.

Consultancy provided by the Apple trainers were further customised for each centre, tailoring to individual centre’s training needs. The trainer also supported the development and implementation of technology-infused lessons plans. An Apple Lighthouse PNLC was formed to allow discussion, sharing and refinement of lesson plans and activities developed by other centres.

As participants of the Apple Lighthouse project, 8 of these centres championed the introduction of intuitive and user-friendly Apple solutions for young learners. They were all guided to explore the use of Apple technology as (i) a replacement for some teaching tools; (ii) an amplification to improve the efficiency and productivity; and (iii) a transformation of teaching and learning that were previously inconceivable.

Focusing on centre’s own niche areas (such as Green Ecosystem, Literacy, Bilingualism, etc.), these 8 centres developed technology-infused lessons plans and documented their experience/reflections as case studies which were shared with other Principals and at other platforms (e.g. MFS’ Learning & Sharing Festivals, educational conferences)

Initiative 4:

Proof of Concept (PoC) – Pepper the Humanoid Robot

In 2016, MFS Jurong Point Centre was one of the two PoC centres in Singapore to pilot the use of robots to teach kindergarten students social skills. Between March to

October in 2016, our children and teachers at Jurong Point centre had a new family member – Pepper the Robot - Pepper worked alongside the teachers and children in class, encouraging interactions and creativity during lessons.

Pepper is a human-like or humanoid robot that can read emotions and learn from human interactions, helping it respond naturally to people. The robot has been used worldwide, from fronting retail stores to helping out in food joints. Its interactive nature helped increase kids' participation in class, especially among less confident children. Children had been observed to be curious and less intimidated around the robot, initiating interactions with them.

Pepper's first class in MFS-Jurong Point taught our children about emotions through the story-telling of the story of the tortoise and the hare which was used to help students to relate to different forms of emotion.

Teachers have found it easier to engage their kids in classes, as Pepper responds to voice, touch and sight. Through this PoC, our teachers had been working with researchers and developers from the Nanyang Technological University's Robotics Research Centre and the robots' parent company, SoftBank Japan, to develop lesson plans. A total of some 8 lesson plans and ensuing lessons were developed and conducted over this PoC period.

For this PoC, the TEPI (Toy Effects on Play Instrument) and the RICA (Robot in Classroom Assessment) instruments had been developed by the Nanyang Technology University (Robotics Research Centre) and used to assess the impact of toys on critical thinking and problem solving, imagination and creative thinking and sociability and independence.

5. CONCLUSION

As we believe that there is no one size fits all where technology is concerned, in MFS, the approach to undertaking technology to support children's learning and teachers' development is an eclectic one and is one that goes beyond individual programmes and projects. To best support our network of 129 centres (to date) with their technology effort, we are committed to providing all children with access to quality ICT infrastructure, learning resources, and ensuring that our principals and teachers

are well-developed, connected and supported by the larger fraternity.

We believe that technology is a tool and not an end itself. It is the Principals and the teachers who can make the difference to our children's learning with the wise selection and planning of lessons supported by technology. Intentional effort is given to ensure that technology is a '*means*' and leveraged upon only if it creates education and that our children will not notice that they are learning through a co-design process where they are empowered while keeping their best interests as key consideration.

6. REFERENCES

- Deloitte. (2015). CIO Website. Retrieved from CIO Website: <http://www.cio-asia.com/tech/mobile-and-wireless/singapore-ranks-highest-smartphone-penetration-in-the-world-deloitte-survey/>
- IDA. (2014). IDA website. Retrieved from <https://www.ida.gov.sg/~media/Files/Infocomm%20Landscape/Facts%20and%20Figures/SurveyReport/2014/2014%20HH%20public%20report%20final.pdf>
- Karuppiyah, D. (2013). Straits Time. Retrieved from <http://www.straitstimes.com/singapore/kids-using-gadgets-at-earlier-age-being-exposed-to-risks-study>
- Koehler, M. &. (2011). Teaching Teachers of Future. Retrieved from <http://www.ttf.edu.au/what-is-tpack/what-is-tpack.html>
- NUS. (2014). Retrieved from ECDA: https://www.ecda.gov.sg/growatbeanstalk/Documents/ECDA%20ECC%202014%20Slides/ConcurrentSession_C/C4%20Is%20Our%20Generation%20of%200Babies%20Turning%20Into%20i-Babies.pdf
- Samsung. (2014). theasianparent. Retrieved from <https://s3-ap-southeast-1.amazonaws.com/tap-sg-media/theAsianparent+Insights+Device+Usage+A+Southeast+Asia+Study+November+2014.pdf>
- The TPACK Framework, <http://tpack.org/>
- Koehler, M. J., & Mishra, P. (2009). What is technological pedagogical content knowledge? Contemporary Issues in Technology and Teacher Education, 9(1), 60-70

Computational Thinking Development in Higher Education

Integrating Computational Thinking into Discrete Mathematics

Kwong-cheong WONG

The Hong Kong Polytechnic University
Hong Kong Community College
wongkwongcheong@gmail.com

ABSTRACT

This paper argues that the various problems caused by the traditional mathematical approach to teaching discrete mathematics to computing students can be alleviated by way of integrating computational thinking into discrete mathematics. The paper proposes a combination of three ideas to facilitate such integration: (a) aiming at understanding the notion of computation, (b) emphasizing both abstraction and automation, and (c) incorporating a functional programming language. The paper exemplifies a plausible approach to developing computational thinking in higher education, namely, through integrating it with an existing subject.

KEYWORDS

Computational thinking, discrete mathematics, computation, abstraction and automation, functional programming

1. INTRODUCTION

Discrete mathematics is “the study of mathematical structures that are ‘discrete’ in contrast with ‘continuous’ ones” (Ouvrier-Bufferet, 2014, p. 181). Here *discrete* means finite or countably infinite in cardinality. Thus discrete mathematics is the branch of mathematics in which we deal with questions involving finite or countably infinite sets, such as integers and fractions (Biggs, 2002). Content-wise, discrete mathematics comprises a diverse range of topics, including logic, proofs, sets, functions, relations, Boolean algebra, combinatorial circuits, recurrence relations and generating functions, combinatorics, discrete probability, coding theory, graph theory, trees and networks, algebraic structures, number theory, algorithms and complexity, finite automata and languages, and cryptography (see the tables of contents of the popular textbooks on the subject, e.g., Chartrand & Zhang, 2011; Epp, 2011; Johnsonbaugh, 2014; Rosen, 2013). Discrete mathematics plays a fundamentally important role in the computing curriculum because it is a foundation for many computing subjects such as data structures, formal software engineering, database systems, compiler design, operating systems, artificial intelligence, theory of computation, computer security, just to name a few. Despite its importance, discrete mathematics is generally regarded as a difficult subject both to teach and to learn, for a number of reasons. First, it contains many disparate topics as listed above, without a unifying theme (except for *being discrete*). Second, these topics are highly theoretical and abstract in nature, full of symbols and definitions. Third, the subject is usually taught in the first year or at the beginning of the second year of the

computing curriculum, before which students usually have studied only a few computing subjects. Fourth, the subject is usually characterized as a mathematical subject without providing a sufficient number of applications to show its connections with, and its relevance to, computing. Finally, the traditional approach to teaching this subject is a mathematical one, using pen and paper and following the sequence of definitions, theorems, proofs and examples (e.g., Chartrand & Zhang, 2011; Epp, 2011; Johnsonbaugh, 2014; Rosen, 2013; see also Jaume & Laurent, 2014). As a consequence, computing students usually cannot see the point of learning this difficult subject and hence very easily lose interest in studying it, even though they are constantly being reminded of the importance of this subject to the computing subjects they are going to study in the curriculum.

Computational thinking has attracted a lot of attention worldwide in recent years since the publication of Jeannette M. Wing’s (2006) highly influential paper in the *Communications of the ACM*, in which she argues that the way computer scientists think about the world is useful in other contexts. Wing writes:

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

(Wing, 2006, p. 33)

Despite its popularity, there is yet no consensus on the definition of computational thinking (e.g., Selby & Woollard, 2014, Tedre & Denning, 2016). In 2010, Wing offers a refined definition of computational thinking:

Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.

(Wing, 2010)

Later, Alfred V. Aho (2012) gives a similar definition of computational thinking:

We consider computational thinking to be the thought processes involved in formulating

problems so their solutions can be represented as computational steps and algorithms.

(Aho, 2012, p.

832)

Due to its exceptional clarity (Tedre & Denning, 2016), we adopt, in this paper, Aho's definition of computational thinking, and argue that most, if not all, of the aforementioned problems caused by the traditional mathematical approach to teaching discrete mathematics to computing students can be alleviated by adopting instead a computational approach in which discrete mathematics is integrated with computational thinking. In the following, we propose a combination of three ideas to facilitate such integration: (a) aiming at understanding the notion of computation (Section 2), (b) emphasizing both abstraction and automation (Section 3), and (c) incorporating a functional programming language (Section 4).

2. AIMING AT UNDERSTANDING THE NOTION OF COMPUTATION

Discrete mathematics has been lacking a unifying theme in organizing its contents, as evidenced by the bewildering array of topics in those popular textbooks (e.g., Chartrand & Zhang, 2011; Epp, 2011; Johnsonbaugh, 2014; Rosen, 2013). As pointed out by Alfred V. Aho (2012, p. 834), "mathematical abstractions called models of computation are at the heart of computation and computational thinking." This entails that to understand computational thinking, we need to understand *the notion of computation*, and to understand the notion of computation, we need to understand *models of computation*. Consequently, this points to one way of integrating discrete mathematics with computational thinking, namely, to set "understanding the notion of computation" as one of the course's objectives and to teach (the rudiments of) models of computation like finite automata and Turing machines. This course objective can help serve as a unifying theme to organize the course's contents in the following way: since learning models of computation (finite automata and Turing machines) presupposes knowledge of logic and graph theory (see, e.g., Kinber & Smith, 2001), we need to teach the latter two (together with other prerequisite topics) first. The following (see Table 1) is a model syllabus for a one-semester discrete mathematics course with 13 lectures designed for beginning year-2 college students who have taken only two computing subjects: Applied Computing, and Introduction to Computer Programming. Note that this course is not intended to be a full-fledged course on the theory of computation, which is usually a senior undergraduate and postgraduate course based on advanced textbooks (e.g., Arora & Barak, 2009; Sipser, 2013); rather, it is intended to be a genuinely introductory course in discrete mathematics and is aimed at, towards the end of the course (lectures 12 and 13), understanding the notion of computation (and its limits) – this is arguably the most fundamentally important concept in computing (see, e.g., Appel, 2014; Bernhardt, 2016). The last chapter (Chapter 13 Modeling Computation) of

Rosen (2013) contains suitable material for teaching this part of the course. So do the last chapters of Jenkyns & Stephenson (2013) and of Chakraborty & Sarkar (2011), both of which are on finite automata and Turing machines. In addition to these textbook chapters, *JFLAP* (Java Formal Languages and Automata Package; see, e.g., Jarvis & Lucas, 2008; Rodger, 2006) and *Visual Turing* (a Turing machine simulator at <http://visual-turing.software.informer.com/2.0/>) are two free visualization software that can help render this part of the course more accessible and fun. In fact, there is evidence that shows that models of computation can be successfully taught even to high school students (Isayama et al., 2016).

Table 1. A one-semester discrete mathematics syllabus aimed at understanding the notion of computation.

<i>I</i>	<i>Logic and Mathematical Proof</i>
1.	Propositional logic
2.	Predicate logic
3.	Mathematical proof
<i>II</i>	<i>Set Theory and Boolean Algebra</i>
4.	Sets
5.	Relations
6.	Functions
7.	Boolean algebra
<i>III</i>	<i>Combinatorics and Graph Theory</i>
8.	Counting
9.	Graphs (1)
10.	Graphs (2)
11.	Trees
<i>IV</i>	<i>Models of Computation</i>
12.	Finite automata
13.	Turing machines

3. EMPHASIZING ABSTRACTION AND AUTOMATION

Given that the mathematical background of our computing students is usually rather weak and their interests usually lie in computing and not in abstract mathematics, we believe that, instead of the traditional mathematical approach, a computational approach, in which computational thinking is integrated, should be adopted to teach discrete mathematics to our computing students. As pointed out by Wing (2008), the two essences of computational thinking are *abstraction* (i.e., model building) and *automation* (i.e., algorithms and their implementation on the computer) – in the words of Wing (2008), "the essence of computational thinking is abstraction" (p. 3717) and "computing is the automation of our abstractions (p. 3718). Consequently, this points to yet another way of integrating discrete mathematics with

computational thinking, namely, to put a good emphasis throughout the course on these two important concepts, abstraction and automation, while teaching and learning each and every topic of the course. The following (Table 2) are some examples of computational projects that are designed for students to do. Each project emphasizes both abstraction and automation in that students have to build a model (representation, symbolization) first and then devise an algorithm and finally implement it on the computer. This computational approach can be seen as a kind of the general teaching method called *contextualization* coined by Guzdial (2016, p.18). The benefit of this method is that “[i]f the learner perceives the relevance of the course context, the course is more concrete and less abstract. There is increased motivation to succeed. That motivation increases success rates.” Guzdial (2016, p. 64)

Table 2. Examples of computational projects emphasizing both abstraction and automation.

Logic	Write a program to accept a propositional logic formula and print out its truth table; write a program to test whether two given propositional logic formulas are logically equivalent.
Sets	Write a program to accept two sets and output their union, intersection, difference, and Cartesian product; write a program to test whether two given sets are the same or one set is a subset of the other.
Relations	Write a program to accept a relation and test whether it is reflective, symmetric, and transitive and hence determine whether it is an equivalence relation; write a program to accept a relation and output its reflective closure, symmetric closure and transitive closure.
Boolean Algebra	Let a given string of eight 0s and 1s be interpreted as the rightmost column of a truth table with Boolean variables x , y and z . Write a program to accept such a string and output the corresponding Boolean expression in minterms.

Graphs	Write a program to implement Dijkstra’s shortest path algorithm.
Trees	Write a program to implement Prim’s algorithm and Kruskal’s algorithm for finding minimum spanning trees.
Automata Theory	Construct a non-deterministic pushdown automaton which recognizes the language $\{a^n b^n \mid n \geq 1\}$ for the JFLAP platform.
Turing Machines	Design a single-tape Turing machine, and then a 2-tape Turing machine, that accepts the language $\{a^n b^n \mid n \geq 1\}$ for the JFLAP platform; implement a universal Turing machine for the JFLAP platform (see Jarvis & Lucas (2008) for a solution).

4. INCORPORATING A FUNCTIONAL PROGRAMMING LANGUAGE

Since the early days of computing, there have been advocates for using programming languages to teach mathematics in general (e.g., Papert, 1980; Harel & Papert, 1990; Feurzeig et al., 2011; Schanzer et al., 2015). In recent years, there have been advocates for using programming languages to teach discrete mathematics in particular (e.g., da Rosa, 2002; VanDrunen, 2011; Ureel & Wallace, 2016). Their rationale, specifically for the latter, is that “problem solving through the medium of the machine is the essence of computer science” and “a programming approach to discrete mathematics affords active learning.” (Ureel & Wallace, 2016, p.1) Although in principle to learn computational thinking does not require any actual programming (Curzon & McOwan, 2017), in teaching and learning discrete mathematics actual programming can help make the abstract contents more concrete because the students can construct the very objects they are learning (Cf. *Constructionism*, see, e.g., Papert (1980) and Harel & Papert (1991)) and thereby rendering the subject more congenial and accessible to our computing students. Consequently, this points to yet another way of integrating discrete mathematics with computational thinking, namely, to incorporate into the subject a programming language. Further, we argue that functional programming languages (e.g., Haskell, ML, O’Caml) are particularly well-suited for this purpose, for the following reasons:

- Functional programming is a method of program construction that emphasizes [mathematical] functions and their applications rather than commands and their execution.

- Functional programming uses simple *mathematical* notation that allows problems to be described clearly and concisely.
- Functional programming has a simple *mathematical* basis that supports equational reasoning about the properties of programs.

Bird (2015, p. 1) (emphasis added)

Indeed, there is an added bonus for incorporating a functional programming language: students can thereby learn to program in one more programming paradigm – the functional programming paradigm (where a computer program is a collection of functions), besides the imperative programming paradigm (where a computer program is a series of commands) and the object-oriented programming paradigm (where a computer program is a collection of interacting objects).

Due to their strong similarity to mathematical language, functional programming languages are a very suitable medium to teach mathematics and are *not* difficult to learn. In fact, there is evidence that shows that the functional programming language Haskell can be successfully taught to even high school students (Algre & Moreno, 2015). To illustrate the succinctness and declarative-ness of code written in functional programming languages, the following are two Haskell programs, one for insertion sort and the other for Prim's algorithm for finding minimum spanning trees; for details, see Hutton (2016, pp. 62-63) and Rabhi & Lapalme (1999, p. 149) respectively.

```
insert x [] = [x]
insert x (y : ys) | x <= y = x : y : ys
                  | otherwise = y : insert x ys
isort [] = []
isort (x : xs) = insert x (isort xs)
```

Figure 1: Insertion sort (Hutton, 2016, pp. 62-63).

```
prim g = prim' [n] ns []
  where (n : ns) = nodes g
        es = edgesU g
        prim' t [] mst = mst
        prim' t r mst
          = let e@(c, u', v') = minimum[(c, u, v) | (u, v,
```

```
c) <- es, elem u t, elem v
r]
in prim' (v' : t) (delete v' r) (e : mst)
```

Figure 2: Prim's algorithm for finding minimum spanning trees (Rabhi & Lapalme, 1999, p. 149).

For more on using functional programming languages to teach discrete mathematics, see O'Donnell et al. (2007), Doets (2012), and vanDrunen (2013).

5. CONCLUSION AND FUTURE WORK

For various reasons, discrete mathematics is a difficult subject for most computing students. We have argued that the problems caused by the traditional mathematical approach to teaching discrete mathematics to computing students can be alleviated by integrating computational thinking into discrete mathematics. Concomitantly, we proposed a combination of three ideas to facilitate such integration, namely, aiming at understanding the notion of computation, emphasizing both abstraction and automation, and incorporating a functional programming language into the subject. We thereby exemplified that integrating an existing subject with computational thinking is a plausible approach to developing computational thinking in higher education (Czerkawski & Lyman III, 2015). We expect that this integration approach can help students learn both the subject and computational thinking better. In our future work, we will implement this proposal and see how it is received.

6. REFERENCES

- Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, 55(7), 832-835.
- Alegre, F., & Moreno, J. (2015). Haskell in middle and high school mathematics. Submission to *TFPIE*.
- Appel, A. W. (2014). *Alan Turing's systems of logic: The Princeton thesis*. Princeton University Press.
- Arora, S. & Barak, B. (2009). *Computational complexity: A modern approach*. Cambridge; New York: Cambridge University Press.
- Bernhardt, C. (2016). *Turing's vision: The birth of computer science*. The MIT Press.
- Biggs, N. L. (2002). *Discrete mathematics* (2nd Edition). Oxford University Press.
- Bird, R. (2015). *Thinking functionally with Haskell*. Cambridge, United Kingdom: Cambridge University Press.
- Chakraborty, S. & Sarkar, B. (2011). *Discrete mathematics*. New Delhi: Oxford University Press.
- Chartrand, G., & Zhang, P. (2011). *Discrete mathematics*. Long Grove, Ill.: Waveland Press, Inc..
- Curzon, P., & McOwan, P. W. (2017). *The power of computational thinking: Games, magic and puzzles to*

- help you become a computational thinker. World Scientific Europe Ltd.
- Czerkawski, B. C., & Lyman III, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends*, 59(2), 57-65.
- da Rosa, S. (2002). The role of discrete mathematics and programming in education. In *Proceedings of the Workshop on Functional and Declarative Programming in Education*.
- Doets, H. C. (2012). *The Haskell road to logic, maths and programming*. Texts in Computing, 4.
- Epp, S. (2011). *Discrete mathematics with applications* (4th edition). [Pacific Grove, Calif.]: Brooks/Cole/Cengage Learning.
- Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, 19(5), 487-501.
- Guzdial, M. (2016). *Learner-centered design of computing education: research on computing for everyone*. San Rafael, California: Morgan & Claypool Publishers.
- Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive learning environments*, 1(1), 1-32.
- Harel, I. E., & Papert, S. E. (1991). *Constructionism*. Ablex Publishing.
- Hutton, G. (2016). *Programming in Haskell* (2nd Edition). New York: Cambridge University Press.
- Isayama, D., Ishiyama, M., Relator, R., & Yamazaki, K. (2016). Computer Science Education for Primary and Lower Secondary School Students: Teaching the Concept of Automata. *ACM Transactions on Computing Education (TOCE)*, 17(1), 2.
- Jarvis, J., & Lucas, J. M. (2008). Understanding the Universal Turing Machine: An implementation in JFLAP. *Journal of Computing Sciences in Colleges*, 23(5), 180-188.
- Jaume, M., & Laurent, T. (2014). Teaching Formal Methods and Discrete Mathematics. arXiv preprint arXiv:1404.6604.
- Jenkyns, T., & Stephenson, B. (2013). *Fundamentals of discrete math for computer Science: A problem-solving primer*. Springer.
- Johnsonbaugh, R. (2014). *Discrete mathematics* (7th edition). Harlow: Pearson.
- Kinber, E., & Smith, C. (2001). *Theory of computing: A gentle introduction*. Upper Saddle River, N.J.: Prentice Hall.
- O'Donnell, J., Hall, C., & Page, R. (2007). *Discrete mathematics using a computer*. Springer Science & Business Media.
- Ouvrier-Buffet, C. (2014). Discrete mathematics teaching and learning. In *Encyclopedia of mathematics education* (pp. 181-186). Springer Netherlands.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc..
- Rabhi, F., Lapalme, G. (1999). *Algorithms: A functional programming approach* (2nd ed.) Harlow; Reading, Mass.: Addison-Wesley.
- Rodger, S. (2006). Learning automata and formal languages interactively with JFLAP. *ACM SIGCSE Bulletin*, 38(3), 360-360.
- Rosen, K. (2013). *Discrete mathematics and its applications*. Mc-Graw Hill.
- Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015). Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical symposium on computer science education* (pp. 616-621). ACM.
- Selby, C., & Woollard, J. (2014). Refining an understanding of computational thinking. Author's original, 1-23.
- Sipser, M. (2013). *Introduction to the theory of computation*. Boston, Mass.: Cengage Learning.
- Tedre, M., & Denning, P. J. (2016, November). The long quest for computational thinking. In *Proceedings of the 16th Koli Calling Conference on Computing Education Research* (pp. 24-27).
- Ureel, L. C., & Wallace, C. (2016). Discrete mathematics for computing students: A programming oriented approach with Alloy. In *Frontiers in Education Conference (FIE)*, 2016 IEEE (pp. 1-5). IEEE.
- VanDrunen, T. (2011). The case for teaching functional programming in discrete math. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 81-86). ACM.
- VanDrunen, T. (2013). *Discrete mathematics and functional programming*. Franklin, Beedle & Associates.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*. Vol. 49, No. 3 (March).
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881), 3717-3725.
- Wing, J. M. (2010). Computational thinking--What and why. *The Link*.

Computational Thinking and Non-formal Learning

Computational Thinking Affordances in Video Games

Sue-inn CH'NG*, Yunli LEE, Wai-chong CHIA, Lee-seng YEONG

Sunway University

(sueinn, yunli, waichong, leeseng)@sunway.edu.my

ABSTRACT

The task of learning programming is a complex process that requires students to simultaneously master the syntax and programming tool while applying problem-solving skills to the given situation. Failure to do so have led to students dropping out of computer science programs and students being disenchanted with programming to the point that these graduates are reluctant to practice in the field. To counter this issue, problem-solving training is sometimes introduced before programming to introduce them to primary concepts of program design and programming without the complexity of syntax and tools as a hindrance. However, problem-solving skills is not something that can be developed over a short period of time. For some, it takes time, practice and effort in which semester long courses do not permit. Previous studies have shown that games can be used as educational tools in the classroom. However, video games are frequently overlooked as an educational tool in favor of serious games. In this paper, we analyze the game play of selected game titles to determine if existing video games contain activities that can be associated to each of the five core skills that characterize computational thinking within the Computer Science discipline.

KEYWORDS

Computational thinking, video games, games and learning

1. INTRODUCTION

Computer science educators have explored the use of games to teach students programming concepts as a solution to cultivate interest in programming among youths. Games have been introduced into such classes as game design assignments where students design and create games to demonstrate the application of learnt technical concepts (Basawapatna, Koh, & Repenning, 2010; Leutenegger & Edgington, 2007; Monroy-Hernández & Resnick, 2008) or as an interactive learning platform where students learn technical concepts through game-play (Kazimoglu, Kiernan, Bacon, & MacKinnon, 2012; Liu, Cheng, & Huang, 2011; Muratet, Torguet, Jessel, & Viallet, 2009). Game design assignments are noted to be more effective in capturing students' interest compared to implementing mock enterprise software since games are something that most students can relate to (Becker, 2001). However, such assignments can also be challenging for students especially for those without considerable knowledge in computer graphics and background in playing/designing games (Sung et al., 2011). Although visual programming tools (such as Scratch (MIT, 2016), Alice3d (Cooper, Dann, & Pausch, 2000), AgentSheets (Basawapatna et al., 2010)) can be used in place of full-blown Integrated Development

Environment (IDEs) to minimize the learning curve for such tasks, sufficient support still needs to be put in place to provide students the necessary knowledge to go about game design otherwise students might end up being intimidated by the feat.

Alternatively, specially developed programming environments (Chaffin, Doran, Hicks, & Barnes, 2009; Kazimoglu et al., 2012; Liu et al., 2011; Muratet et al., 2009) have been created to encourage students to learn through gaming. Unlike traditional video games, these serious games incorporates technical concepts into the game play in the form of coding (to complete tasks/missions) (Chaffin et al., 2009) (Liu et al., 2011; Muratet et al., 2009) or manipulation of the graphical interface as done by (Kazimoglu et al., 2012) for the purpose of learning instead of leisure. The use of serious games in classrooms do indicate an improvement in student engagement and motivation towards the content in most of these game-based learning implementations. However, the creation of these games requires time, skills and careful coordination between the game developer and course facilitator to ensure that the curriculum is integrated into the game and deployed in the classroom in a cohesive manner. Since skills such as computational thinking takes time to develop and requires practice, students with increased frequency of gameplay should exhibit better levels of computational thinking and programming skills compared to those who rarely play the game. But, most of these research do not study the adoption rate of these games as leisure activities at the end of the course or the effects of extended usage of the proposed serious games on students' problem-solving and programming skills over time.

On the other hand, recent studies shows that playing video games has cognitive, emotional and social benefits (Granic, Lobel, & Engels, 2014). Adachi et al (Adachi & Willoughby, 2013) work showed that the more adolescents reported playing strategic video games, the more improvements were evident in self-reported problem-solving skills recorded the following year. The same positive predictive association was not recorded for fast-paced game genres such as racing and fighting games. Ventura et al (Ventura, Shute, & Zhao, 2013) hypothesized that there might be a relationship between video game usage and persistence in the face of failure. To investigate this, they used anagram riddle task to measure the level of persistence among video-game players and found that frequent game players are more likely to spend longer times on unsolved problems compared to infrequent video game players. Video games have also been reported to bridge generation gaps in (Osmanovic & Pecchioni, 2016) and to promote prosocial behavior in an

international study among school children (Gentile et al., 2009).

Although video games are not originally designed to be an education medium, they may possess many good learning principles and educational affordances (Frazer, Argles, & Wills, 2008) that is usually underutilized in education. The question is: Can video games be used as an informal aid to foster computational thinking skills in its players? As a preliminary study, we evaluate the educational affordances displayed in selected video games titles. The games are then evaluated based on the core five skills (Kazimoglu et al., 2012) that characterize Computational Thinking (CT) within the Computer Science discipline to identify aspects of gameplay within the game that affords the cultivation of CT skills.

The paper is structured as follows: Section 2 presents an introduction to Computational Thinking. Section 3 provides readers with a brief background of video game research and classification. The details of the survey conducted and how the video game titles are selected to be evaluated during the study is presented in Section 4 while the significance of our findings are discussed in Section 5. The paper is concluded in Section 6.

2. COMPUTATIONAL THINKING

Computational Thinking (CT) is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Wing, 2008). Since computing heavily affects everyone lives, Wing (Wing, 2008) envision that CT will be a fundamental skill in the 21st century that have the same importance as numeracy and literacy. Since the introduction of the term computational thinking, there have been many definitions in literature defining the skills and activities that encompass CT for different disciplines. Barr and Stephenson (Barr & Stephenson, 2011) provided examples on how the nine core² CT concepts and capabilities may be embedded in different discipline activities. Lee et al. (Lee et al., 2011) examined CT in practical youth programs and identified the terms “abstraction”, automation” and “analysis” to describe how young people use CT to solve novel problems.

In the field of computer science, the recent work by (Kazimoglu et al., 2012) defined five core skills that characterizes CT within the computing discipline as problem-solving, building algorithms, simulation, debugging and socializing. Table 1 associates the generic activities within video games to each CT skill described in (Kazimoglu et al., 2012) to show how video games in general can support the cultivation of CT skills in its players.

Table 1. Game activities associated with each core CT skill

Core CTS	Game Activities
Problem solving	Identifying purpose of game (can be main goal or mini-tasks) to complete level or whole game.
Building algorithms	Formulate steps to achieve goal or complete mini-tasks encountered during the game. Select appropriate algorithmic technique to execute chosen approach.
Simulation	Manoeuvre game controls to move game character(s) to execute steps and analyse if actions brings player closer to achieving the goal or completing the game/level.
Debugging	Modify existing plan to improve performance (gain more points, increase survivability, reduce time taken)
Socializing	Discuss game plan with other players or analyse game play of other players

3. VIDEO GAMES

Video games are interactive games played using a computing device for the purpose of leisure. Once restricted to only the desktop computer, video games are now played on a wide range of mobile devices and special game players. Video games can be classified in a number of different ways ranging from the device that it is executed on, to the gameplay and interactivity of the game. Due to the vast array of dimensions on which video games can vary and lack of concretely defined identification criteria that can be used by all parties; it is difficult to create a comprehensive taxonomy of games that is used and accepted by all (Clarke, Lee, & Clark, 2015). Thus, game titles are normally provided along with the game genre to illustrate games that are categorize under that particular genre. For example, the work by (Granic et al., 2014) provided an overview of game genres sorted along the dimensions of complexity and social interaction and also uses game titles as stand-in language for different types of gameplay encountered. In the work by (Frazer et al., 2008), only four game genres are explored in their research and example of game titles for each genre is provided to guide the study done. For this paper, the game titles selected for analysis is based on an online survey conducted to study the gaming habits of undergraduate students when they were young.

According to (Gee, 2005), video games that encourage players to stop, thoroughly explore different possibilities and consider new strategies and goals before moving on, rather than simply progressing towards their goals as fast as possible can promote problem solving skills in players.

² Definition proposed by the American Computer Science Teacher Association (CSTA) and International Society for Technology in Education (ISTE) for use in K-12 education.

Table 2 Definition of different game genres and example video games for each genre

Game Genre	Definition	Example Video Games
Action	Emphasizes physical challenges such as hand-eye coordination and reaction time.	Fruit Ninja, Pinball, O2Jam
Adventure	Player assumes the role of protagonist in an interactive story driven by exploration and puzzle solving.	Grand Theft Auto, Bully, Assassin's Creed
Fighting	Player controls an on-screen character and uses this character to engage his opponent in close combat.	Street Fighter, Naruto
Platformer	Player controls a character to jump between suspended platforms, over obstacles, or both to advance the game	Super Mario, Mushroom Men
Racing	Players competes in a race using a vehicle.	3D Mario Kart, Need for Speed, Test Drive
Shooter	Player controls the character from a first-person or third-person view to shoot opponents to proceed through missions without the player character dying	Call of duty, Counterstrike, Halo
Simulation	Game designed to closely simulate the aspects of the real-world inside the game such as farming, managing sports team or merely living through the virtual characters' lives.	The Sims, Fifa, NBA, Harvest Moon
Strategy	Games that require the player to strategize or formulate a plan in order to win.	Warcraft, Civilization, Red Alert, Dota

These game features are mostly present in game genres such as RPG, strategy, simulation and adventure games

(Adachi & Willoughby, 2013) categorizes these games as strategic games while games such as fighting, action and racing games that have little downtime between battles/races are categorized as fast-paced games. In his study, he found that fast-paced games do not promote problem-solving skills in the long run because it provides little to no opportunity to gather information and strategize before a battle or a race. Thus, the game genres are then further sorted into two categories – strategic game (strategy, platform, shooter, simulation, adventure/RPG) and fast-paced to determine if strategic game titles contain more affordances that promotes CT skills compared to fast-paced game titles.

4. METHODOLOGY

Data was collected from 655 students (509 students from the school of business and 146 students from the school of computing) first year students undertaking an “Introduction to Computers” course from Sunway University in Malaysia during the January – March 2016 semester. The average age of the students is 20 years old. In the survey, the students were asked to self-report their gaming habits (now and when they were young) through multiple choice questions (starting age and frequency of play) and open-ended questions (name of favorite video game). Based on the premise that gaming is a memorable experience during the students’ childhood or adolescence, students who truly played games and for those who have spent a sizeable amount of their time doing this would at the very least remember the name of the game that they have played and/or be able to describe the game play of that particular game. The age at which the students start playing games is used as a reference point to check the validity of the responses. For example, if the respondents claim that they started playing Candy Crush at an age of less than 6 years old, this response would be deemed invalid because Candy Crush was only released in the year 2012. Responses that were incomplete or those who gave non-existent/invalid games for either instances were ignored in the study.

The game titles were then categorized into game genres based on the genres provided by gaming website IGN (IGN Entertainment Inc., 2016) and Gamespot (Gamespot, 2016). Table 2 shows the different types of genres (Rollings & Adams, 2003) considered and example video games provided by students that are classified under each genre. The most frequently occurring game title for each genre were then selected and the game activities within each game were matched to each core CT skills presented in Table 1. To extract the game activities within each game, the games were re-played by the authors using online game emulators (UtopiaWeb, 2009).

5. RESULTS & DISCUSSION

Figure 1 shows that the percentage of players who play each game genre now (current) and then (youth). The most frequently played game genre reported by students when they were young was adventure type (32.2%) games. This is followed by platform-based games and strategy type game. It is noted that the type of game genre played changes as the students age. There were more students opting to play Strategy games now compared to platform-based and Adventure/RPG type of games when they were young. The authors in (Sung et al., 2011) built their serious game based on strategy type of gameplay since their data also supports the same observation – majority of players reportedly played strategy game compared to other game genres. However, this trend might be only true because strategy games are popular now; as is the case with platform games; once popular in the early 90’s (Boutros, 2006). Hence, explaining the high percentage of respondents who played platform games when they were young.

It was observed that, regardless of the game genre, all the games have goals/missions and a reward mechanism that entices players to continue playing. Players would then try to find ways to maximize these rewards while minimizing damage on their game characters during game play. This feature in all games requires players to determine the problem that they are currently encountering and to devise new solutions based on whatever information that they

have at hand. These are the exact features observed in (Adachi & Willoughby, 2013) that promotes problem-solving skills.

The type of information to be collected for each games are different but players would need to analyze this information to determine the next best course of action to get them a step towards the main goal. Information in strategic games can come in the form of combination of in-game items, character abilities and environment. For example, in the strategy game of Defense of the Ancient (DOTA), players would have to come up with strategies to conquer the opponents' "throne" while defending their own. During the battle to conquer/defend, they would have to buy weapons and items to ensure that their game character, termed heroes in the game, is able to defend or defeat opponent characters. In the event that they encounter enemy heroes, players would have to analyze their own and the opponent hero's statistics (item, level), environment and position of their own team mates during the game to determine whether to proceed and engage the enemy or to retreat. On the other hand, in the football management game of FIFA 2002, players are required to select their players and determine the team's formation for each football game. During each game, players are given control of each of the football players action to score/defend football goals. They would then have to observe opponents' character actions to determine whether to defend or attack (score a goal).

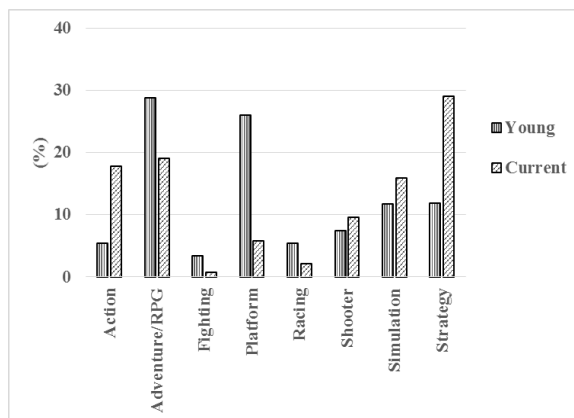


Figure 1. Game genre played by participants now (current) and then (young)

Table 3 decomposes and associate the game activities of the most popular game titles, provided by the students, from each genre with the five core CT skill category. Although fast-paced games focuses on reflex during the game play, it was also observed during our analysis that it also contains opportunities that allows players to collect information on the effects of their actions. With this information, players can reflect and modify their actions for future games to increase their rewards, refer to "Debugging" row in Table 3.

6. CONCLUSIONS

This paper discusses the possibility that video games contain educational affordances that promote CT skills. We analyzed the game play of eight popular game titles

played by students during their youth. Our analyses show that the video games regardless of game genre contains activities that support the cultivation of each CT skill category. Since video games are equally enjoyed by both male and female students during their formative years, this translates to a wealth of gaming experience that can be tapped by educators in the classroom. This can be used by instructors to make the task of learning CT skills less intimidating. Seen in this light, the challenge now is perhaps not to develop serious games to instill CT skills but to get students to apply the skills gained from one domain (video games) to other domains (programming).

We acknowledge that the game titles selected for this study only covers an extremely small subset of game titles that is available in the market and that the analysis done is based on the authors' experiences in playing the games. More concrete evidences on the relationship between video games and computational thinking skills can be collected in the future by observing the actual game play of student players for each of the game.

7. REFERENCES

- dachi, P. J. C., & Willoughby, T. (2013). More than just fun and games: The longitudinal relationships between strategic video games, self-reported problem solving skills, and academic grades. *Journal of Youth and Adolescence*, 42(7), 1041–1052.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *Acm Inroads*, 2(1), 48–54.
- Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 224–228). ACM.
- Becker, K. (2001). Teaching with games: the minesweeper and asteroids experience. *Journal of Computing Sciences in Colleges*, 17(2), 23–33.
- Boutros, D. (2006). A detailed cross-examination of yesterday and today's best-selling platform games. *Gamasutra [Online]*.
- Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental evaluation of teaching recursion in a video game. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games* (pp. 79–86). ACM.
- Clarke, R. I., Lee, J. H., & Clark, N. (2015). Why Video Game Genres Fail A Classificatory Analysis. *Games and Culture*, 1555412015591900.
- Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D

- tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges* (Vol. 15, pp. 107–116). Consortium for Computing Sciences in Colleges.
- Frazer, A., Argles, D., & Wills, G. (2008). The same, but different: The educational affordances of different gaming genres. In *Advanced Learning Technologies, 2008. ICAIT'08. Eighth IEEE International Conference on* (pp. 891–893). IEEE.
- Gamespot. (2016). Gamespot - Video Games Reviews & News. Retrieved from <http://www.gamespot.com/>
- Gee, J. P. (2005). Good video games and good learning. In *Phi Kappa Phi Forum* (Vol. 85, p. 33). THE HONOR SOCIETY OF PHI KAPPA PHI.
- Gentile, D. A., Anderson, C. A., Yukawa, S., Ihori, N., Saleem, M., Ming, L. K., ... Bushman, B. J. (2009). The effects of prosocial video games on prosocial behaviors: International evidence from correlational, longitudinal, and experimental studies. *Personality and Social Psychology Bulletin*.
- Granic, I., Lobel, A., & Engels, R. C. M. E. (2014). The benefits of playing video games. *American Psychologist*, 69(1), 66.
- IGN Entertainment Inc. (2016). IGN South East Asia. Retrieved from <http://ap.ign.com/>
- Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, 9, 522–531.
- 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.
- Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. In *ACM SIGCSE Bulletin* (Vol. 39, pp. 115–118). ACM.
- Liu, C.-C., Cheng, Y.-B., & Huang, C.-W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education*, 57(3), 1907–1918.
- MIT. (2016). Scratch - Imagine, Program, Share. Retrieved from <https://scratch.mit.edu/>
- Monroy-Hernández, A., & Resnick, M. (2008). FEATURE empowering kids to create and share programmable media. *Interactions*, 15(2), 50–53.
- Muratet, M., Torguet, P., Jessel, J.-P., & Viallet, F. (2009). Towards a serious game to help students learn computer programming. *International Journal of Computer Games Technology*, 2009, 3.
- Osmanovic, S., & Pecchioni, L. (2016). Beyond Entertainment Motivations and Outcomes of Video Game Playing by Older Adults and Their Younger Family Members. *Games and Culture*, 11(1-2), 130–149.
- Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on game design*. New Riders.
- Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. W., Goldstein, D. S., & Nordlinger, J. (2011). Game-themed programming assignment modules: a pathway for gradual integration of gaming context into existing introductory programming courses. *IEEE Transactions on Education*, 54(3), 416–427.
- UtopiaWeb. (2009). My Abandonware. Retrieved April 10, 2017, from <http://www.myabandonware.com/>
- Ventura, M., Shute, V., & Zhao, W. (2013). The relationship between video game use and a performance-based measure of persistence. *Computers & Education*, 60(1), 52–58.
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717–3725.

Table 3. Example game activities associated to each Computational Thinking skill (CTS) category for each game genre.

CTS Category	Game Activity							
	Strategic Games				Fast-paced Games			
	Strategy (DOTA)	Simulation (Fifa 2002)	Shooter (Counter Strike)	Adventure (Grand Theft Auto)	Platform (Super Mario Bros)	Fighting (Street Fighter)	Racing (Need for speed)	Action (02Jam)
Problem solving	Destroy opponent's throne.	Manage soccer team to win each game to ultimately win World Cup.	Defeat terrorist. Mission to rescue/escort/protect depends on type of map.	Become the biggest criminal (reaching target points) by performing missions for local crime syndicate.	Progress through levels by defeating enemies, collecting items and solving puzzles.	Defeat opponent.	Control vehicle to win race.	Collect experience points by controlling player's character.
Building algorithms	Farm gold by neutral creeping, laning or killing opponent heroes. Buy items to increase survivability and/or attack power. Player determines hero build.	Choose team. For chosen team, player needs to select team formation and approach to defeat computer opponent.	Choose team - terrorist/counter-terrorist. Buy items to defeat opponent based on character, battle background (map) and mission.	Choose criminal missions, gather equipments to assist in mission and avoid being caught by police while executing mission.	Determine how to complete level, defeat different types of enemies and what items can do.	Identify combos for selected game character. Determine combo that can deal most damage to opponent.	Choose racing map. Choose car type - manual or automatic Choose car model.	Play music by pressing corresponding button. Mostly hand-eye coordination. Can decrease/increase speed of music to change level of difficulty.
Simulation	Perform chosen tactic.	Play match using chosen team formation and players.	Play game using chosen armament.	Play chosen mission.	Play levels with maneuvers to evade enemy attacks and gather needed items to advance.	Perform combo.	Determine how to control car over race terrain.	Press corresponding keys with accordance to music.
Debugging	If hero dies too often or cannot defeat opponent hero, <u>modify tactic</u> to improve deficiency.	If opponent team wins, <u>modify team formation</u> and approach to improve defence or offence.	<u>Modify item build</u> to increase survivability or damage to opponent.	<u>Modify strategy</u> to execute mission if player keeps getting caught by police or lose their equipment while doing mission.	If keep dying by same type of enemy at same spot, <u>change approach to solve</u> problem.	If current combo does not work, <u>modify combo to increase damage</u> .	If lose car control, <u>modify car movements the next round</u> .	Practice to increase accuracy of pressing buttons in accordance to music.
Socializing	Get item tips from team members or watch other players play similar heroes.	NA	Killed players get to watch current game to view how players use chosen armaments to defeat opponent.	NA	Watch other players play the same level.	Discuss with other players combo for game character.	Multiplayer option that allows players to play with other players but difficult to share strategy with other players.	Compete with other players.

You Can Code – An innovative approach to transform the workforce in the textile and apparel industry

Bessie CHONG,

Director, Group Training and Talent Management
Esquel Group, Hong Kong
Chongbe@esquel.com

ABSTRACT

Textile and apparel industry has long been stereotyped as “traditional” and “old-fashioned”. As a non-traditional company in a traditional industry, Esquel encourages employees to innovate and to challenge the status quo. “You can code” campaign was initiated in 2015 to engage and propel staff at all levels towards its vision of “Making a Difference”. The campaign aims to drive a sustained cultural transformation to turn the less technically minded employees into confident users of technology with computational thinking (CT) ability, through developing a mobile apps. Many useful mobile apps have been developed and some have been commercially adopted. The campaign helps the Company to nurture a culture of innovation, problem-solving and collaboration.

KEYWORDS

Coding, Computational Thinking, App Inventor, Mobile Technology, Innovation.

1. INTRODUCTION

Esquel believes that coding will soon become a basic job skill for everyone. It is vital for the employees to have some understanding of programming regardless of what professional they are in. Failing that, the career mobility of an individual may be hindered, and so as organizational growth. This is especially important for the manufacturing sector, as Industry 4.0 is fast approaching. The new age employees need to equip with a new set of skills.

Founded in 1978, Esquel started as a shirt maker and have over the years developed the capacity to weave innovative technologies into its people-centric culture. With key production bases established in strategic locations in China, Malaysia, Vietnam, Mauritius and Sri Lanka, and a network of branches in the US, Europe and Asia, it exports over 120 million shirts per year and offer total shirt solutions to global apparel and textile markets, from concept to rack.

Esquel employs 57,000 diversified workforce united under the corporate 5E culture – Ethics, Environment, Exploration, Excellence and Education, and the motto “Fun People Serving Happy Customers”. It operates with an aspiration of “making a difference” by weaving positive impact to the employees, societies and environment.

1.1. Our Business Challenges and Opportunities

Esquel is in the industry of textile and apparel manufacturing, where all players face structural challenges, including rising labor and material costs, reduced profit margin and shortage of skilled labors. On the other hand, the rise of fast fashion further disrupts the industry by demanding quicker production cycles, more rapid prototyping and smaller order sizes. Therefore, the traditional manufacturing model with long lead time and mass production will no longer survive.

The textile and apparel manufacturing industry employs over 75 million people worldwide (Stotz & Kane, 2015) with an aggregate export amount over US\$744 billion in 2015 (World Trade Organization, 2015). The industry is still versatile, and has huge potential. The question is - how do manufacturers stay competitive and enable sustainable growth amidst the changing environment? Many players in this industry believe that growth must be tied with the overuse of labor and that competition must be based on low wages. Therefore, it is typical for those players to migrate the manufacturing base to chase for cheap labor. But Esquel decided to stay in locations where it has good operating conditions and to build local talent pool. Esquel strives to improve labor productivity to offset rising wages. It would rather improve the efficiency of the people and pay them well by integrating them into the technology, not by replacing them by using the technology.

The coming fourth industrial revolution, known as Industry 4.0, will provide Esquel with an opportunity to sustain its leadership position in apparel operations. The advent of the fourth industrial revolution is associated with the development of global industrial networks, to which all production processes of a wide variety of enterprises will be connected. As a result, computer interaction environment is developed around the modern human (Yastreba, 2015). That means employees would work with cyber-physical systems in a smart factory environment. They will make use of the mobile technology and data to enhance real-time communication, improve alignment, and make timely decisions. Eventually, it will increase productivity and efficiency.

1.2. People Challenges

Nowadays, working environments are changing at unprecedented speed. The rise of robotics and artificial intelligence calls for new skills and competencies. In the workforce of Esquel, only 3.4% have any technical qualifications, only 12% have a diploma or above, and

38% were born before the personal computer became popular. There is a serious shortage of technically minded and savvy employees. So how can it turn those less technically minded employees into confident technology users? How can it empower people to come up continuous improvement ideas and solve their own work problems systematically and independently?

It needs to have a campaign to drive this transformation. It needs to inspire the employees to participate in the revolution. It will be a huge challenge as the target group is highly diversified, and spreads over 9 countries, 20 operations.

2. “YOU CAN CODE” CAMPAIGN

Coding is a skill that helps people learn how to think, systematically. By developing computational thinking, people can deconstruct problems step by step, and identify different recommendations. However, computational thinking may seem too abstract and coding may seem too technical. To engage everyone who does not have technical knowledge, a fun and practical approach is needed. The ‘App Inventor’ application developed by the Massachusetts Institute of Technology (MIT) was identified as the driver of this campaign.

The simple graphical interface of App Inventor allows an inexperienced user to create basic, fully functional mobile apps within an hour or less. It transforms the complex language of text-based coding into visual, drag-and-drop building blocks. It can change employees’ perception of technology through this campaign. It would develop their logical reasoning skills, programming capabilities, and more importantly, computational thinking ability. Computational thinking is a fundamental skill for everyone, and it is a way humans solve problems (Wing, 2006). It includes problem decomposition, algorithmic thinking, abstraction, and automation (Yadav et al., 2017). By equipping computational thinking ability, employees will ultimately become innovators, problem-solvers, collaborators as well as process owners. They equip themselves for life. Yadav et al. further stressed given the irreplaceable role of computing in the working life of today, the competence to solve problems in technology-rich environments is of paramount importance.

“Therefore, there is a need to pay attention to CT as part of the broader concept of digital literacy in vocational education and training, as otherwise adults with only professional qualification may not be well prepared for the working life in the twenty-first century”. (Yadav et al., 2017, p.1065).

To Esquel, no matter whether they are workers, staff, managers or executives, they need to have the attitude and ability to solve problems and realize ideas which will improve productivity.

2.1. Campaign Design and Implementation

This campaign is designed around how to change Attitudes, upgrade Skills and build Knowledge.

Table 1. ASK model.

To change	Attitude	<ul style="list-style-type: none"> by developing those ideas gradually from accepting, to understanding, to embracing, to exploring by showcasing employees that everyone can code
To upgrade	Skills	<ul style="list-style-type: none"> by conducting workshops, activities, events and competitions by encouraging employees to innovate or to solve specific problems using the technology
To build	Knowledge	<ul style="list-style-type: none"> by developing an independent learning mindest by creating a rich learning resource environment.

It is impractical if only IT colleagues are involved in providing some classroom training and expect that employees will change the attitude towards technology.

In order to engage all employees, the role of IT throughout the campaign is purposely downplayed. An “all-in” approach was adopted and the campaign was launched out in 5 development phases.

Table 2. Five development phases and achievements

Phase	What have been achieved
Pioneering	Workshops for senior management team and board members were conducted to collect their feedback and get their buy-in. About 90% of them attended the training. Some of them also became Esquel’s pioneers.
Modelling	Some General Managers and Directors were invited to be the trainers to conduct workshops for other colleagues, from workers to managers. Almost 300 employees were trained. They reinforced the notion ‘If I can code, you can code too.’
Changing	43 super-users were identified and trained to be the change agents or ambassadors. They joined a customized master trainer course. Then they delivered training at different operations.
Cultivating	The master trainers were sponsored to launch a series of workshops and fun days, for the staff and for their kids in order to cultivate the skills and mindset. The ambassadors set up information and promotional booths to educate the frontline operators. A total of almost 800 people were trained.
Realizing	The first “Esquel’s App Challenge” Competition was organized in order to encourage the applications of the new skills. Many interesting and practical Apps were developed.

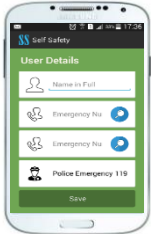

Through 28 workshops, over 2,430 training hours were provided for 1,200 participants, including 1,100 employees and 100 of their children from 10 different locations in first 10 months. The strategy of teaching the children and letting children teach their parents back also worked well. Overall, the impact was encouraging. A lot of positive feedbacks were received -



- “Something looks complicated but can be very user friendly for us in building an App. Useful and valuable of some information/tools can be shared from the company!” - By a Sales Manager in Hong Kong.
- “The introduction of the online programme 'App Inventor' is useful for non-professionals to build up our own app.” - By a Senior Sales Executive in Hong Kong
- “Easy to operate for dummies. All ordinary people can participate in creating app without the support of IT”. – by an Engineering Officer, in a Garment Factory

2.2. Value Created from the Campaign

The most important impact is the values created, including attitude change towards technology, employee engagement, employer branding, and process improvement. Many more app ideas from the employees were received. It shows after innovating once, employees are likely to innovate again. Now, non-IT employees can perform IT tasks, and even build prototypes by themselves. This in turn will allow IT professionals to focus on enterprise level app development.

Table 3. Some examples of the mobile apps developed in this campaign.

Applications	Functions
	<u>Safety app for women.</u> A Sri-Lanka colleague developed this app with a GPS-function for women leaving work at night.
	<u>Parking space and free bike locator app</u> This app helps colleagues find a parking lot or an available bicycle near operation sites.

	<u>Recruitment app</u> This app helps HR colleagues in processing some manual work of the recruitment, such as making test paper and personnel data entry.
	<u>New-born baby photo sharing app</u> This app allows a new parent to send the 1 st new born baby photo to their friends with the baby information and the logo of company's hospital.

Those applications help to save time and improve efficiency, while the broader benefits are incalculable. Department heads and IT team are reviewing many more bottom-up initiatives from employees. This campaign also shows Esquel's commitment to upgrading their workers to become knowledgeable. It reinforces Esquel's brand as a caring and non-traditional company.

Besides, in the process of developing their own mobile applications, the employees started to integrate computational thinking into their everyday work. They took attempt to analyze the problems by breaking them down and identifying the root cause, instead of jumping to quick fixes. The story of Yang Hua Mei illustrates how a basic coding training can bring an impact on a sewing worker.

2.3 Story of Yang Hua Mei - A Garment Factory Worker

‘You can code’ is just the beginning of a transformational journey. Esquel is introducing a new world to its employees who might never have had the chance to realize their own ideas otherwise. Hua Mei's story is one of many at Esquel that inspires many others to learn the technique of app coding.

Yang Hua Mei was a young woman from the southwest of China with an immense interest in fashion design and a desire to follow a career in apparel manufacturing. After graduating from high school in 2014, she joined Esquel as a sewing worker and brought many undeveloped fashion ideas that were waiting to be realized.

One day, Hua Mei found that the company was running a campaign. She believed that this campaign could teach her the computational skills necessary to turn her undeveloped fashion ideas to life; skills such as computational thinking, logical reasoning and simple programming. Without any hesitation, Hua Mei took the opportunity, like other 1000 employees. By the end of the campaign, Hua Mei and two other colleagues had built an app allowing users to mix-and-match their wardrobe.



Figure 1. Interface of the wardrobe application developed by Yang Huamei's team.

"Before joining 'You can code', I didn't even know what was meant by an 'app'! I have learned so much in the program, and now I appreciate the work of the technology gurus – however simple an app might seem, building one requires many steps and logical thinking!", said Hua Mei.

She also realized that the basic coding technique equips her with computational thinking ability, which in turn helps her become an independent thinker. As a sewing worker, from time to time, she faced problems in operating her sewing machine and managing the sewing quality. Before she learnt coding, whenever she came across problems, she simply asked the mechanic to fix it or change some machine parts by herself. She would not bother to understand the problems, identify the root causes, and think about how to prevent them in future. But now, she becomes proactive in learning technical skills and integrating the computational thinking ability to solve her everyday work problems. She also aspires to evolve from a sewing worker to a technician one day.

3. KEY SUCCESSFUL FACTORS OF THE CAMPAIGN

3.1 Align and strengthen Esquel's Vision

Esquel's Vision: Making a Difference

Esquel makes a difference by growing with its employees not by squeezing from them. 'You Can Code' campaign upskills the employees, opens their minds to technology and equips them for life. This aligns the company's vision of making a difference and receives huge support from different levels.

3.2 Use a creative way to inspire innovation, problem solving, and collaboration

Esquel is the first commercial entity to adopt App Inventor to train employees in computational thinking. Even though computational thinking is conceptual and hard to develop in a short period, the Company tries to change attitudes, upgrade skills and build knowledge through the development of mobile app.

3.3 Engage all levels and collaborate across teams

The campaign engaged all levels by enrolling board members and senior managers as pioneers. Some modelled the skills and trained their teams. Some members of their teams became ambassadors and trainers.

They promoted the notion of 'If I can code, you can code too', and they changed the perception that senior staff are conservative and less tech-savvy. They encouraged, trained and supported their peers at their sites.

A cross-functional organizing committee for the App Challenge contest was established so that their expertise can be leveraged. The committee members collaborated on the planning and execution of the contest.

An 'all-in' approach encourages everyone to engage in the campaign. Many employees, including board members and sewing workers, joined the fun and easy '1-hour coding' workshops. The kids of employees were also invited, who in turn, influenced and motivated their parents to learn coding. The campaign engaged primary students and PhDs, and people aged 6 to over 60. The 'all-in' approach has formed a community to ensure the long-term sustainable benefits.

3.4 Leverage the use of existing communication platforms to promote the campaign

HR colleagues and ambassadors from different operation sites used existing communication platforms to promote the campaign, its workshops and events. Having used these platforms during the campaign, those users are more likely to use those platforms to communicate and collaborate on matters which affect their business. Platforms included Yammer, WeChat, Intranet, and company's TV broadcasting, as well as traditional channels such as the notice boards and promotional booths at factories.

4. IMPACT FROM THE CAMPAIGN

What the campaign has done is only a small step. But it started the momentum. After the "You Can Code" campaign, it is found the rise of mobile app culture in Esquel. Employees are keen on thinking how to build some applications to improve some work-related or living-related issues. Recently, there is another interesting application, named Esquel Carpool, developed by the factory colleague using another software. The impact is enormous.

4.1 Need for Carpool

According to the data provided by Chinese Environmental Protection Bureau, 15 to 30% of the pollution comes from the emission from the cars (Chinese Environmental Protection Bureau, 2016). And with exponentially increasing number of cars, now traffic is seen everywhere in many cities in China.

Esquel's largest operation is located in Gaoming, Foshan. It has about 23,000 employees working in several factories where are spread over the city of Gaoming. That means about 40% of Esquel employees are working and living here. Employees have to travel from home to these working locations, in similar timing, and similar routines every day.

Many of them have to take company bus or city bus, and wait in a long line in sun, rain and wind. Commuting can

easily take up half an hour or even one full hour per trip. For those 2,000 employees with their own private cars, the situation is no better. Driving to work is not at all pleasant when they have to be stuck in traffic and fight for the meagre 200 parking spaces available around the factories. A lot of times, they have to park far away and then take a 10-minute walk to the office.

How can Esquel make a difference for the colleagues, for both group of people so that they can save time in waiting for bus, fighting for the traffic and looking for parking spaces? How can they save some gasoline bills meanwhile reduce their carbon footprint?

Can something be done to change their lifestyle and behaviour, reduce the environment impact, and inspire others to contribute in building a green city?

4.2 Birth of Esquel Carpool Application

An employee in the factory initiated an idea to develop an app to facilitate the carpool process in Esquel. This is how the Esquel Carpool App was born.

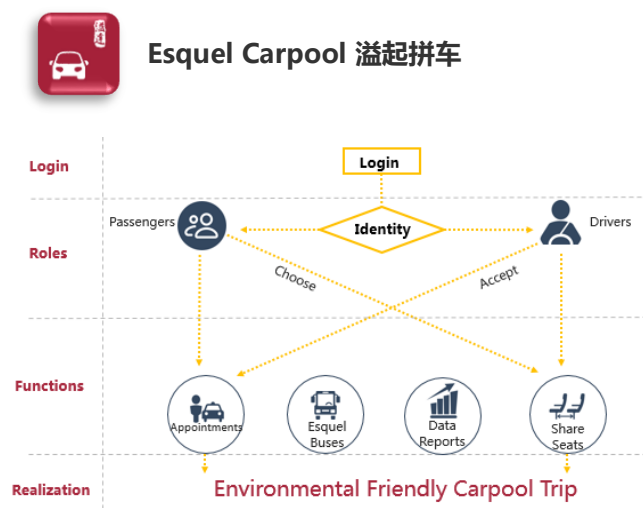


Figure 2. Design map of Esquel carpool application.

Employees can use their staff ID to login, and they can choose to be either passengers or drivers. Passengers can publish their needs (e.g. where and when they want to go), or select the remaining seats available from the drivers. Drivers can publish their free seats in the App to the passengers, or directly select the passengers. After that, they can form a group to communicate. This App can also share the real-time location of company bus, and show the data report of the carpool usage.

4.3 Benefits of Using Esquel Carpool

Within the first six months, this App already recorded 8070 carpools, with the saving of more than 6,000 litres of gasoline, saving 14,000 kilograms of carbon emission.

This app helps to realize the benefits of carpooling on saving the environment. More important, it provides a platform to make connection with colleagues from different department and foster the caring culture.

4.4 What's Next

The company committed to provide this software for free to any companies and organizations, and already put this software in the open source community GitHub. It hopes millions of the programmers in the world would help to improve this App.

5. CONCLUSION

This is just the beginning of Esquel using modern technology beyond work to make the lives better. This is a signal to the colleagues that they can come up with great initiatives to make Esquel a better work environment. It demonstrates how a simple app can address daily needs.

Different departments and non-IT colleagues have already built their own apps or collaborated with IT. Esquel committed to continuously advance the coding skills of its colleagues and build its organizational capability. The company plans to introduce more sophisticated programming training gradually. More super-users and change agents will be identified and invited to participate into this people transformation journey. They will be the voice, and the generation of leaders that make a difference.

6. REFERENCES

- Chinese Environmental Protection Bureau. (2016). 我国雾霾成因及对策. 紫光阁(3), pp. 82-83. Retrieved from cnki.net: [http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST2016&filename=ZIGU201603061&uid=WEEvREcwSIJHSlIdRa1FhdKJkcGkzRmlaVDg1WXA3WG9XMmJobkw3NEM5UT0=\\$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4ggI8Fm4gTkoUKaID8j8gFw!!&v=MjU1MTJySTIEWlISOGVYMUxleFITN0Ro](http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST2016&filename=ZIGU201603061&uid=WEEvREcwSIJHSlIdRa1FhdKJkcGkzRmlaVDg1WXA3WG9XMmJobkw3NEM5UT0=$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4ggI8Fm4gTkoUKaID8j8gFw!!&v=MjU1MTJySTIEWlISOGVYMUxleFITN0Ro)
- Stotz, L., & Kane, G. (2015). *Facts on The Global Garment Industry*. Retrieved February, 2015, from Clean Clothes Organization: <https://cleanclothes.org/resources/publications/factsheets/general-factsheet-garment-industry-february-2015.pdf>
- Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35.
- World Trade Organization (2015). Retrieved 2015, from World Trade Organization <http://stat.wto.org/StatisticalProgram/WSDBViewData.aspx?Language=E>
- Yadav, A. et al. (2017). *Computational Thinking as an Emerging Competence*. Retrieved January, 2017, from ResearchGate: <https://www.researchgate.net/publication/307942866>
- Yastrebe, N. (2015). The internet of things and the forth industrial revolution: the problem of humanitarian expertise. *Journal of Scientific Research and Development* 2(8), 24-28.

Computational Thinking and IoT

Off the Screen, and Into the World of Everyday Objects: Computational Thinking for Youth with the Internet of Things

Mike TISSENBAUM¹, Josh SHELDON¹, Evan PATTON¹, Arjun GUPTA¹, Elaine ZHANG¹, Divya GOPINATH¹

¹Massachusetts Institute of Technology

mtissen@mit.edu, jsheldon@mit.edu, ewpatton@mit.edu, argupta@mit.edu, elainez@mit.edu, divyagop@mit.edu

ABSTRACT

This paper discusses the opportunities presented by the growth of the Internet of Things (IoT) to provide youth opportunities to develop their computational thinking and digital empowerment. This paper argues that to support youth in developing these literacies, we need to develop platforms that reduce the barriers of entry while still allowing them to explore and develop their computational identities. To this end, this paper introduces an extension to App Inventor by MIT that enables students to quickly design, develop, and implement IoT applications. We outline one IoT activity for youth and future directions for both curricular and technical development.

KEYWORDS

Computational Thinking, Digital Empowerment, App Inventor, Internet of Things

1. INTRODUCTION

By 2018, it is expected that physical devices that make up the Internet of Things (IoT) will surpass mobile devices as the leading type of Internet-connected devices. IoT devices are projected to comprise nearly 16 billion of the expected 28 billion connected devices by 2021 (Ericsson, 2016). The explosive growth of this ubiquitous computing landscape, in which computers will seamlessly integrate into our everyday lives and objects (Weiser et al., 1999), will have profound effects on how people relate to the world around them and to each other. This is especially true for youth, who will know no other world. If we want these youths to be active creators and shapers of their digital futures, rather than simply passive consumers of it, there is a growing need to support them in developing the necessary computational literacies (Vee, 2013; Wilensky, Brady & Horn, 2014) for this rapidly growing IoT landscape. While applications such as Scratch and App Inventor have made traditional and mobile computing accessible to youth (Meerbaum-Salant et al., 2010; Wolber et al., 2011), currently, there are no similar platforms for supporting computational literacy development in the IoT space. In response, this paper outlines a research and technology agenda with a focus on two critical elements for supporting youth as they develop these computational literacies: 1) The need for low barrier-of-entry tools with which young learners can create, test, and refine IoT designs that connect with their everyday lives; and 2) A means for abstracting the many (often conflicting and confusing- Banfa, 2016) standards for developing IoT interventions.

2. COMPUTATIONAL LITERACIES, COMPUTATIONAL IDENTITIES, AND DIGITAL EMPOWERMENT

Since computational thinking (CT) entered the mainstream over a decade ago, there has been a growing recognition for the need for everyone, not just computer scientists, to develop computational thinking (Wing, 2006; Voogt, et al., 2015). CT's origins draw Seymour Papert's work on the Logo programming language, which focused mainly on procedural thinking and programming (1980). Since Papert's groundbreaking work, the idea of computational thinking has been broadened to encompass a broader range of computational concepts (Grover & Pea, 2013). While there is no single agreed upon definition of computational thinking, most definitions focus on the ability to recognize the role that computation plays in our world, and to formulate problems and solutions through computational means (Wing, 2006; Brennan & Resnick, 2012).

As we seek to create environments and tools with which students' can become computational thinkers, we argue that there are two especially important computational thinking aspects we must support. Based on Brennan and Resnick's computational thinking framework, we posit that two additional computational thinking perspectives (2012; Tissenbaum et al., 2017) impact young people's long-term success as computational thinkers. The first is computational identity (CI); their identities as people who can think computationally and as members of the computational community more broadly. The second is digital empowerment (DE); recognizing their personal ability to affect the world around them through computation (Tissenbaum et al., 2017). The latter is especially important, and builds on Papert's ideas of students as self-aware, empowered, intellectual agents who feel capable of making their own learning decisions, posing their own questions, and finding answers to those questions (Papert, 1972; 1987). Digital empowerment extends Papert's vision by instilling in children the knowledge that they can, through computation, effect real change in the world.

These perspectives, digital empowerment and computational identity are closely intertwined. As Friere (1993) argues, students need to understand their relation to the world (identity) in order to transform it (empowerment). Thus, in an increasingly digital world, we advocate that by developing critical computational identities and literacies, students will become empowered to create computational solutions to

challenges in their local communities, as posited by Lee and Soep (2017).

3. BLOCKS-BASED PROGRAMMING LANGUAGES FOR COMPUTATIONAL THINKING

Creating conditions in which young learners can develop digital empowerment is a challenging endeavor often complicated by the frequently complex tools required to create digital artifacts. The need to understand the sometimes arcane syntax and grammar of traditional programming languages has long been a barrier for engaging youth in computational practices (Maloney et al., 2004). In response, many educational researchers have developed blocks-based environments. In other words, these environments leverage a primitives-as-puzzle-pieces metaphor, in which users assemble functioning programs by snapping together "blocks" of code together (Weintrop & Willensky, 2015; Tissenbaum et al, 2017). The blocks provide visual cues that show users which pieces fit together and which do not. These systems also give visual and (often) auditory feedback when pieces may, or may not, connect together. The use of blocks-based programming languages can help scaffold novice programmers to more easily develop relatively complex programs and have been used to support young learners to develop games (Maloney et al., 2004), 3D animations (Dann, Cooper, Pausch, 2011), and computational models (Begel & Klopfer, 2007).

4. AIM: BRINGING COMPUTATIONAL THINKING AND DIGITAL EMPOWERMENT INTO EVERYDAY LIVES

This abundance of low-barrier approaches to developing computational artifacts has helped realize Papert's vision of intellectually empowering youth; however, the introduction of smartphones and truly mobile computing radically changed the role that computing plays in our everyday lives. Instead of having to go to the computer, for many of us, the computer now comes with us everywhere we go. This is a radical extension of Papert's vision of bringing every learner into the computer lab (Papert, 1993; Klopfer, 2008), and fundamentally changes how we think of computing and how we think computationally - taking computing off the computer and into the lived world. It also offers the promise of moving beyond youth who are intellectually empowered towards youth who are empowered to change the world. In response to this radical change in the relationship between learners, computation, and the "real world" and the need to provide low-barrier ways for learners to be truly empowered, new programming environments needed to be developed that harness this potential. App Inventor by MIT (AIM) is an example of a platform that responded to this need. AIM is a blocks-based programming that allows youth to develop fully-functional mobile applications for the Android operating system. Currently, AIM has over 6 million registered users (with over 300,000 unique monthly users) spread

across 195 countries, who have collectively worked on more than 20 million mobile app projects (<http://appinventor.mit.edu/explore/>). Given the breadth and scope of AIM users, AIM is in a unique position to have a direct impact on the computational thinking and digital empowerment of children all over the world.

5. EXTENDING COMPUTATIONAL THINKING AND DIGITAL EMPOWERMENT INTO EVERYDAY OBJECTS

Just as smartphones made computers truly personal, the explosive growth of the Internet of Things (IoT) is having a similar impact on how we relate to the world around us and the objects within it (Ashton, 2009). Every day, the world becomes more connected, with everyday objects containing sensors, actuators, displays and other input and output channels all woven together computationally and over the Internet (Weiser, Gold & Brown, 1999). The growing connectivity between everyday objects and the computational power of the Internet offers a potential for us to harness our creativity to extend the capabilities of our lived environments (Rogers, 2006). However, in order to realize this vision of our youth as active empowered creators of their digitally augmented lives, rather than passive consumers of it, we need to develop tools that allow them to design, build, and test IoT-based interventions. By creating integrated environments that allow programming of both mobile apps and IoT hardware with common metaphors, we extend digital empowerment to all aspects of the lives of young learners.

5.1. Subsections

While the promise of youth developing transformational interventions using IoT is exciting, the technical complexity required to actually develop these interventions is a clear barrier. In response, we have developed an extension to AIM that allows youth to create mobile applications that can send and receive data from Arduino, a popular and modular computing platform for IoT. AIT leverages the Bluetooth low energy (BLE) standard to enable AIM applications to communicate with a wide range of peripheral devices. To communicate with Arduino, for example, one would use the Generic Attribute (GATT) Profile Specification. Sensors and actuators attached to an Arduino device can then be exposed as services and characteristics using GATT and read/written by an AIT application. Normally, developing these kinds of communications protocols would be, for most young learners, prohibitively complex; however, with AIM we have created an abstraction layer that allows young learners to focus more on creating and implementing their designs.

5.2. Instantiating AIT: Building an Interface for Healthy Plants

In order to show how AIT can help young learners we developed an exemplar activity students can follow to build an application that connects directly with the

physical world. We wanted an application that provided young learners a lens into the potential of IoT; exposed them to the programming building blocks for basic IoT functionality; while also offering opportunities for them to extend and explore the design further

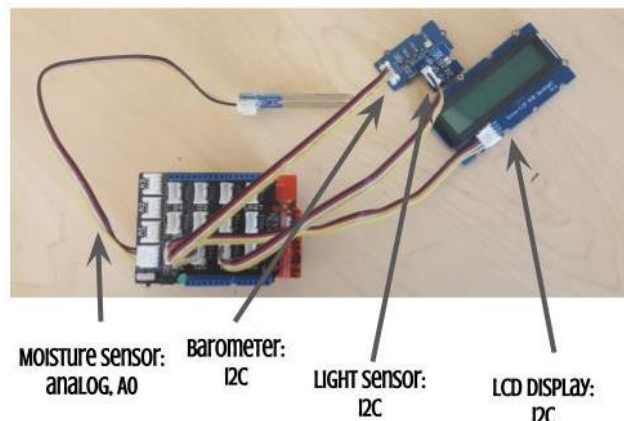


Figure 1. An Arduino with a Grove Kit, moisture sensor, barometer (temperature sensor), light sensor, and LCD screen

The activity allows youth to build a plant monitoring app and uses Arduino and the Grove Kit (<https://www.seeedstudio.com>), a popular extension kit for Arduino that lets users add various inputs (e.g., buttons, and touch, heat and light sensors) and outputs (LCD displays, buzzers, LEDs). For this activity, students use three Grove inputs - a temperature sensor, a moisture sensor, a light sensor - and an LCD display for output (Figure 1).

Once built, the app allows users to get notifications about the state of the plant (how much light the plant is getting, the temperature of the room, and its moisture levels - Figure 2) on their phone via Bluetooth. The application also allows for a "conversation" to take place between the child and their plant. For instance, when the child waters the plant, the "plant" (via the Arduino and moisture sensor) sends the student the message "Did you water me?" If the child replies "Yes" on their phone, it sends a message back to the plant (via the BLE on the Arduino) and the plant send a message back to the child saying "Thanks for watering me!" In this way, the child begins to understand the potential for receiving data about, and communicating with, the everyday objects in their lives.

As part of this activity, youth are encouraged to extend the application to try out new ways of connecting and communicating with the physical world. For instance, youth are prompted to think about how they might set up specified alerts to the phone based on the plant's condition (e.g., it needs water, or it is too hot in the room for the plant). By having kids expand and iterate on the initial version of the healthy plant app, we provide them with opportunities to develop their computational identities and recognize their own growing digital empowerment.

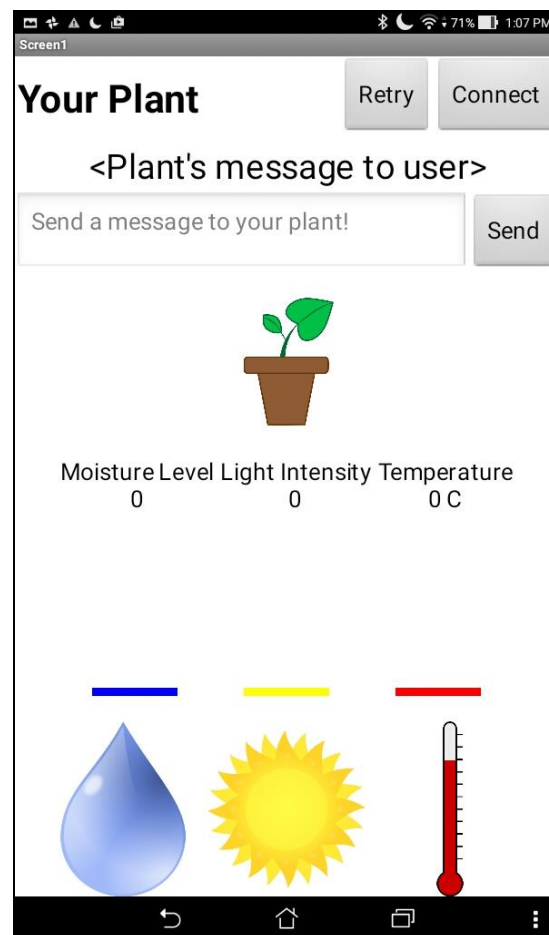


Figure 2. The AIM Healthy Plant mobile interface.

6. DISCUSSION AND FUTURE WORK

This paper outlines a theoretical imperative for moving computational thinking and digital empowerment beyond the computer lab and into the world. Equally important, this paper argues the need to extend this empowerment into the objects that occupy our daily lives. By providing low-barrier opportunities for young children to critically explore their potential for changing their relationship with the world around them, we open up new ways for youth to transcend Papert's vision of intellectually empowered agents (1987), towards agents empowered to change the world. While the work is still preliminary, we see great potential for platforms such as AIT realize this vision.

Moving forward, we plan to create a series of structured inquiry activities that take advantage of the expanded capabilities of AIM to allow young learners to explore the world around them through computational means. For example, the next stage in the Healthy Plants activity, will connect it with a middle school curriculum on ecology. Student groups will individually develop approaches for monitoring and evaluating the health of their plants. Using a knowledge community approach (Brown & Campione, 1996; Scardamalia & Bereiter, 1994), students will work together to share technological approaches and scientific findings. In this way, we can

connect computational thinking and broader science practices.

We are also working to extend the capabilities of AIT. The next instantiation of AIT will connect to a virtual machine (VM) targeting a range of IoT hardware, including but not limited to the different flavors of Arduino, Raspberry Pi, and BBC micro:bit. This will leverage ongoing work to provide an abstraction over many of the core BLE concepts so users of AIT can focus on building and creating without the high barrier to entry required to understand and use Bluetooth or other wireless technologies. AIT will also provide a blocks editor for programming IoT devices running the VM platform, which can either be programmed directly from the browser or via any application built with AIT. This type of abstraction is similar to how AIM hides the complexity of portability between different Android implementation and hardware. We expect that this abstraction applied to IoT will further empower youth to build novel solutions to community problems and see themselves as digitally empowered citizens.

7. REFERENCES

- Ashton, K. (2009). That 'internet of things' thing. *RFiD Journal*, 22(7), 97-114.
- Banfa, A. (July, 2016). IoT Standardization and Implementation Challenges. *IEEE.org Newsletter*. Retrieved from <http://iot.ieee.org/newsletter/july-2016/iot-standardization-and-implementation-challenges.html>
- Begel, A. and Klopfer, E. 2007. Starlogo TNG: An introduction to game development. *Journal of E-Learning*.
- Dann, W. P., Cooper, S., & Pausch, R. (2011). *Learning to Program with Alice (w/CD ROM)*. Prentice Hall Press.
- Ericsson, A. B. (2015). Ericsson mobility report: On the pulse of the Networked Society. Ericsson, Sweden, Tech. Rep. EAB-14, 61078.
- Freire, P. (1993). *Pedagogy of the Oppressed*. 1970. New York: Continuum, 125.
- Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2), 199-237.
- Klopfer, E. (2008). *Augmented learning: Research and design of mobile educational games*. MIT press.
- Lee, C. H., & Soep, E. (2016). None But Ourselves Can Free Our Minds: Critical Computational Literacy as a Pedagogy of Resistance. *Equity & Excellence in Education*, 49(4), 480-492.
- Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004, January). Scratch: a sneak preview [education]. In *Creating, connecting and collaborating through computing, 2004. Proceedings. Second International Conference on* (pp. 104-109). IEEE.
- Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M.M. 2010. Learning computer science concepts with scratch. *Proc. of the 6th Annual ICER Conference* (2010), 69-76.
- Papert, S. (1972). Teaching children thinking*. *Programmed Learning and Educational Technology*, 9(5), 245-255.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
- Papert, S. (1987). A critique of technocentrism in thinking about the school of the future. Retrieved February 12, 2017, from <http://www.papert.org/articles/ACritiqueofTechnocentrism.html>
- Papert, S. (1993). The children's machine. *TECHNOLOGY REVIEW-MANCHESTER NH-*, 96, 28-28.
- Rogers, Y. (2006, September). Moving on from weiser's vision of calm computing: Engaging ubicomp experiences. In *International conference on Ubiquitous computing* (pp. 404-421). Springer Berlin Heidelberg.
- Tissenbaum, M., Sheldon, J., Seop, L., Lee, C., (2017, April). Critical Computational Empowerment: Engaging Youth as Shapers of the Digital Future. *IEEE Global Engineering Education Conference*, Athens, Greece.
- Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, 1(2), 42-64.
- Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715-728.
- Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question. *Proceedings of the 14th International Conference on Interaction Design and Children - IDC '15*.
- Weiser, M., Gold, R., & Brown, J. S. (1999). The origins of ubiquitous computing research at PARC in the late 1980s. *IBM systems journal*, 38(4), 693-696.
- Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM*, 57(8), 24-28.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor – Create Your Own Android Apps*. Sebastopol, CA: O'Reilly.

Computational Thinking and Inclusive society

Developing interest to share and craft based on the Technology Acceptance Model

Chien-sing LEE^{1, 2}, Samuel Hong-shan LOW¹

¹Department of Computing and Information Systems,
Sunway University, Malaysia.

²Faculty of Creative Industries, Universiti Tunku Abdul Rahman, Malaysia.
chiensingl@sunway.edu.my, samuelowbb@hotmail.com

ABSTRACT

The Malaysian Ministry of Education aims to increase interest in learning Science, Technology, Engineering and Mathematics, through Science2Action. Among these initiatives in Science2Action, is the use of Art(s). By combining the Internet, technology and crafts, e-crafting is formed. This e-crafting project aims to increase awareness about what interests the audience through sharing of and development of craft, hopefully towards possibilities of ideation and mixing crafts, extending from the original craft such as origami. Designed based on the Technology Acceptance Model, findings are positive.

Keywords: e-crafting, audience interests, share, STEAM, Technology Acceptance Model

1. INTRODUCTION

Commonly known as an art trade or occupation that requires a special set of manual skills or an ability majoring in handiwork, crafting is an art in the making or doing. E-Crafting is making waves across the world. It provides more fun, convenience and can improve digital lifestyle. To craft, one needs to first ideate.

Ideation is key to Wing's (2006) computational thinking. Two capstone projects were undertaken under Sunway University's internal grant, to explore how images and augmented reality (first project) and craft (second project) can increase interest in Science, Technology, Engineering and Mathematics (STEM), improve ideation and improve digital lifestyles. These projects are exploratory. The first project was reported in Wong and Lee (2016). This paper reports on the second project, i.e., e-crafting. It continues from the work by Lee and Wong (2015) on developing social innovations among youth via design thinking and is inspired by Penn University's e-crafting.

1.1 Objectives

There are many types of crafts and e-crafts. Hence, the main objective is to increase awareness, interest and appreciation in crafts by enabling people to craft more by making it a fun art. Fun art is by enabling playing and editing around the craft, adding one's own thoughts into it.

Second, is by encouraging users to share interesting crafts with people around the globe. An added incentive is that currently, there is no platform without a fee. Platforms such as in Table 1 require membership and a

certain fee (Table 1). For people in today's era, things that come free are always the best and there is no harm trying as they will not lose out. Hopefully one day they will make their own crafts and we hope to produce successful young entrepreneurs for the future.

2. RELATED WORK

There are different types of e-crafting around the globe today. Examples of e-crafting can be a photoshop tool, self-made flying aero plane, and a useless box. Some other examples are in Table 1 and e-crafting's website.

Table 1. e-Crafting websites which require fees.

Platform A-Z list	Payment model	Price range	Offline?	Notes
Craft Daily https://www.craftdaily.com Main topics: Beading, Crochet, DIY, Jewelry Making, Knitting, Mixed Media, Quilting, Scrapbooking and Paper Crafts, Sewing, Spinning, Weaving	Subscription	\$19.99 monthly or \$199.99 annual	Streaming only	Focus on crafts, subscription to Interweave video library. Run by Thought Industries who also run craftonlineuniversity.com Free? No free option or trial option.
Craftsy http://www.craftsy.com/ Main topics: Sewing & Quilting, Cake & Cooking, Yarn & Fiber Arts, Art & Photo, Home & Garden, More (jewellery and paper)	Pay per course	£9.99 – 35.50	Offline with the app	Focus on crafts, some entrepreneurial topics. Discounts and frequent incentives entice purchases. Free? Selected Free course give an idea of how the platform works.

3. METHODOLOGY

To make it possible for people to share their work, an online platform is needed. There are two parts to this capstone project. One is a Facebook website and the other an online portal.

Target age group are 18-29 years old as they bring in new innovations and ideas. Most of them are youths in the Boys Brigade in Selangor, Malaysia. Craft is one of the skills learnt in Boys Brigade, similar to the Boy Scouts, Girl Guides.

Systems design and development follows the Software Development Lifecycle. Design and assessment are based on the Technology Acceptance Model (Davis, 1989) as presented in Figure 1.

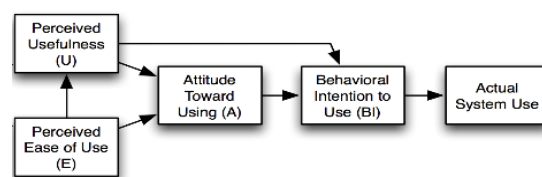


Figure 1. Technology Acceptance Model

Based on the Technology Acceptance Model (TAM) above, a questionnaire is given to users to find out whether the platform is a good idea, has its usefulness and ease of use. The initial survey consists of four students. At the end of the prototyping, another survey is carried out, on 10 students.

The system requirement specifications for this project are:

1. Database to store user account information.
 - 1.1. Log in, create account (integrated to Facebook)
2. A website platform for people to:
 - 2.1. Post and share their crafts
 - 2.2. View others crafts and also able to give opinions
 - 2.3. Crafts can be enhanced by others

For this current platform, users who upload crafts to the Facebook e-Crafting page will have their uploads at the website as well. At the moment, the data integration is done manually. In the future, it will be automated. An example of uploads to the Facebook site is in Figure 2.

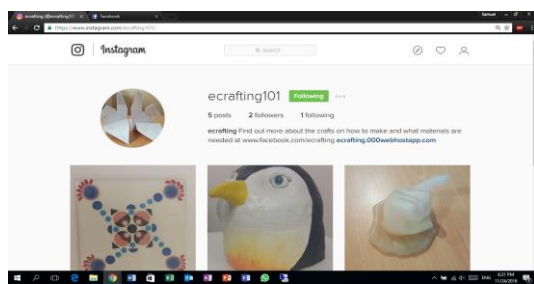


Figure 2. An Instagram page consisting of all the pictures of the art and crafts uploaded into the Facebook page.

4. FINDINGS

Based on the final survey, all ten students think it is a good idea to share crafts among users. Seven say that sharing is caring while three say that they have gained new knowledge and interest. All ten of them also think that technology and craft can go well together and that this website encourages them to share their crafts.

Nine of them said they learnt something useful from this website and only one did not learn anything useful. This may be due to different personal interests/preferences.

Next question, does this platform increase interest towards craft? Eight of the users said yes and two said no. This result also can be due to personal interests/preferences.

Most of the users feel pride, happiness, amazement and even creative when their craft is displayed and appreciated by people around them. This feeling makes them feel appreciated, making them share more of their ideas and crafts.

Based on the technology acceptance model, ease of use and intention to use have been considered. Eight of the users said it is easy to use and two said it is okay to use

and not hard or easy. As for the intention to use the platform again, all of them said they would use it again.

Facebook analytics for the week of Nov18 to Nov24 (the last week of testing) indicates reached 69 users, 41 page views, 282 post engagements and a total of 9 views for the videos (Figure 3).

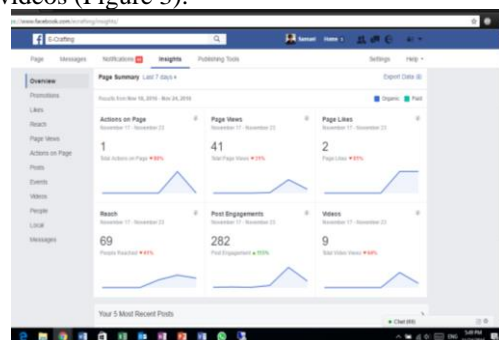


Figure 3. Facebook's analysis for the week of Nov18 to Nov24

5. CONCLUSION

This study shows that designing based on the Technology Acceptance Model can reap fruitful benefits, even to promote crafts and e-crafting. Possible extended users are seniors and their caregivers/ families whereby the website and portal can become a resource/community-sharing center. Adaptations to diverse users can be carried out through assessment of the resource's difficulty level and the contextual dialogues that can be generated from the respective resource.

ACKNOWLEDGEMENT

The authors would like to thank the reviewers for their kind and constructive comments/suggestions. This project is funded by Sunway University's internal grant (Feb-Dec 2016). The first author thanks Universiti Tunku Abdul Rahman where she first saw wonders of different types of crafts while she was a Faculty there, Assoc. Prof. Dr. K. Daniel Wong for collaborations/prior work on design thinking and Science-based research which led to this work, Dr. Juan Carlos Aguilera for prior discussions towards a broader system integrating the internal grant project(s) with his evolutionary algorithms to be funded under another grant with Dr. K. Daniel Wong.

REFERENCES

- Brown, T. & Wyatt, J. (2007). Design Thinking for Social Innovation. In *Stanford Social Innovation Review*, 31-35.
- Davis, F. D. (1989). Perceived usefulness, perceived ease of use and user acceptance of information technology, *MIS Quarterly*, 13(3): 319-340.
- E-crafting.com, 2016. [Online]. Available: <http://www.e-crafting.com/>. [Accessed: 21- Apr-2016].
- E-crafting, 2016 [Online]. Available: <http://www.e-crafting.org>.

Lee, C. S. & Wong, K. D. (2015). Developing a disposition for social innovations: An affective-socio-cognitive co-design model. *International Conference on Cognition and Exploratory Learning in the Digital Age*, 180-185.

Wing, J. (2006). Computational thinking.

Communications of the ACM. 49(3), 33-35.

Wong, C. K. & Lee, C. S. (2016). A better understanding of how gamification can help improve digital lifestyles. *International Conference on Virtual Systems and Multimedia*, 1-8.

General Submission to Computational Thinking Education

Complementary Tools for Computational Thinking Assessment

Marcos ROMÁN-GONZÁLEZ^{1*}, Jesús MORENO-LEÓN², Gregorio ROBLES³

¹ Universidad Nacional de Educación a Distancia, Spain

² Programamos.es & Universidad Rey Juan Carlos, Spain

³ Universidad Rey Juan Carlos, Spain

*mroman@edu.uned.es, jesus.moreno@programamos.es, grex@gsyc.urjc.es

ABSTRACT

Computational thinking (CT) is emerging as a key set of problem-solving skills that must be developed by the new generations of digital learners. However, there is still a lack of consensus on a formal CT definition, on how CT should be integrated in educational settings, and specially on how CT can be properly assessed. The latter is an extremely relevant and urgent topic because without reliable and valid assessment tools, CT might lose its potential of making its way into educational curricula. In response, this paper is aimed at presenting the convergent validity of one of the major recent attempts to assess CT from a *summative-aptitudinal* perspective: the Computational Thinking Test (CTt). The convergent validity of the CTt is studied in middle school Spanish samples with respect to other two CT assessment tools, which are coming from different perspectives: the Bebras Tasks, built from a *skill-transfer* approach; and Dr. Scratch, an automated tool designed from a *formative-iterative* approach. Our results show statistically significant, positive and moderately intense, correlations between the CTt and a selected set of Bebras Tasks ($r=0.52$); and between the CTt and Dr. Scratch (predictive value $r=0.44$; concurrent value $r=0.53$). These results support the statement that CTt is *partially convergent* with Bebras Tasks and with Dr. Scratch. Finally, we discuss if these three tools are complementary and may be combined in middle school.

KEYWORDS

Computational thinking assessment, Computational Thinking Test, Dr. Scratch, Bebras Tasks, middle school.

1. INTRODUCTION

Computational thinking (CT) is considered in many countries as a key set of problem-solving skills that must be acquired and developed by today's generation of learners (Bocconi et al., 2016). However, there is still a lack of consensus on a formal CT definition (Kalelioglu, Gülbahar, & Kukul, 2016), on how CT should be integrated in educational settings (Lye & Koh, 2014), and especially on how CT can be properly assessed (Grover, 2015; Grover & Pea, 2013). Regarding the latter, even though computing is being included into K-12 schools all around the world, the issue of assessing student's CT remains a thorny one (Grover, Cooper, & Pea, 2014). Hence, CT assessment is an extremely relevant and urgent topic to address, because "without attention to assessment, CT can have little hope of making its way successfully into any K-12 curriculum", and consequently "measures that

would enable educators to assess what the child has learned need to be validated" (Grover & Pea, 2013, p. 41).

Moreover, from a psychometric approach, CT is still a poorly defined psychological construct as its nomological network has not been completely established; that is, the correlations between CT and other psychological constructs have not been completely reported by the scientific community yet (Román-González, Pérez-González, & Jiménez-Fernández, 2016). Furthermore, there is still a large gap of tests relating to CT that have undergone a comprehensive psychometric validation process (Mühling, Ruf, & Hubwieser, 2015). As Buffum et al. (2015) say: "developing (standardized) assessments of student learning is an urgent area of need for the relatively young computer science education community" (Buffum et al., 2015, p. 622)

In order to shed some light on this issue, one of the major attempts to develop a solid psychometric tool for CT assessment is the Computational Thinking Test (CTt) (Román-González, 2015). This is a multiple-choice test that has demonstrated to be valid and reliable ($\alpha=0.80$; $r_{xx}=0.70$) in middle school subjects, and which has contributed to the nomological network of CT in regard to other cognitive (Román-González, Pérez-González, & Jiménez-Fernández, 2016) and non-cognitive (Román-González, Pérez-González, Moreno-León, & Robles, 2016) key psychological constructs. Continuing this research line, now we investigate the convergent validity of the CTt, that is, the correlations between this test and other tools aimed at assessing CT. Thus, our general research question is:

RQ_(general): What is the convergent validity of the CTt?

1.1. Computational thinking assessment tools

Focusing on K-12 education, especially in middle school and without being exhaustive, we find several CT assessment tools developed from different perspectives:

CT Summative tools. We can differentiate between: a) Aptitudinal tests such as the aforementioned *Computational Thinking Test* (which is further described in 2.1.), the *Test for Measuring Basic Programming Abilities* (Mühling et al., 2015), or the *Commutative Assessment Test* (Weintrop & Wilensky, 2015). And b) Content-knowledge assessment tools such as the summative tools of Meerbaum-Salant et al. (2013) in the Scratch context, or those used for measuring the students' understanding of computational concepts after introducing a new computing curriculum (e.g., in Israel, Zur-Bargury, Pârv, & Lanzberg, 2013).

CT Formative-iterative tools. They provide feedback, usually in an automatic way, for learners to improve their CT skills. These tools are specifically designed for a particular programming environment. Thus, we find *Dr. Scratch* (Moreno-León & Robles, 2015) or *Ninja Code Village* (Ota, Morimoto, & Kato, 2016) for Scratch; the ongoing work of Grover et al. (2016) for Blockly; or the *Computational Thinking Patterns CTP-Graph* (Koh, Basawapatna, Bennett, & Repenning, 2010) for AgentSheets.

CT Skill-Transfer tools. They are aimed at assessing the students' transfer of their CT skills to different types of problems: for example, the *Bebras Tasks* (Dagiene & Futschek, 2008) are focused on measuring transfer to 'real-life' problems; or the *CTP-Quiz* (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011), which evaluates the transfer of CT to the context of scientific simulations.

CT Perceptions-Attitudes scales, such as the *Computational Thinking Scales* (CTS) (Korkmaz, Çakir, & Özden, 2017), which uses five-point Likert scales and has been recently validated with Turkish students.

CT Vocabulary assessments. They are aimed at measuring elements and dimensions of CT verbally expressed by children (i.e., 'computational thinking language'; e.g., Grover, 2011).

Using only one type from the aforementioned assessment tools can lead to misunderstand the development of CT skills by students. In this sense, Brennan and Resnick (2012) have stated that looking at student-created programs alone could provide an inaccurate sense of students' computational competencies, and they underscore the need for multiple means of assessment. Therefore, as it has been pointed out by relevant researchers (Grover, 2015; Grover et al., 2014), in order to reach a total and comprehensive understanding of the CT of our students, different types of complementary assessments tools must be systematically combined (i.e., also called "systems of assessments"). Following this idea, our paper is specifically aimed at studying the convergent validity of the CTt with respect to other assessment tools, which are coming from different perspectives. Thus, our specific research questions are:

RQ_(specific-1): What is the convergent validity between CTt and Bebras Tasks? RQ_(specific-2): What is the convergent validity between CTt and Dr. Scratch?

Although the three instruments involved in our research are aimed at assessing the same construct (i.e., CT), as they approach the measurement from different perspectives, a *total convergence* ($r > 0.7$) is not expected among them, but a *partial* one ($0.4 < r < 0.7$) (Carlson & Herdman, 2012). Answering the aforementioned questions may contribute to develop a comprehensive "system of assessment" for CT in middle school settings.

2. BACKGROUND

2.1. Computational Thinking Test (CTt)

The Computational Thinking Test³ (CTt) is a multiple-choice instrument composed by 28 items, which are administered on-line (via non-mobile or mobile electronic devices) in a maximum time of 45 minutes. Each item of the CTt is presented either in a 'maze' or in a 'canvas' interface; and is designed according to the following three dimensions (Román-González, 2015; Román-González, Pérez-González, & Jiménez-Fernández, 2016):

- **Computational concept addressed:** each item addresses one or more of the following seven computational concepts, ordered in increasing difficulty: Basic directions and sequences; Loops-repeat times; Loops-repeat until; If-simple conditional; If/else-complex conditional; While conditional; Simple functions. These 'computational concepts' are progressively nested along the test, and are aligned with the CSTA Computer Science Standards for the 7th and 8th grade (Seehorn et al., 2011).
- **Style of answers:** in each item, responses are presented in any of these two styles: 'visual arrows' or 'visual blocks'.
- **Required task:** depending on which cognitive task is required for solving the item: 'sequencing' \approx stating in an orderly manner a set of commands, 'completion' of an incomplete set of commands, or 'debugging' an incorrect set of commands.

We show an example of a CTt item translated into English in Figure 1, with its specifications detailed below.

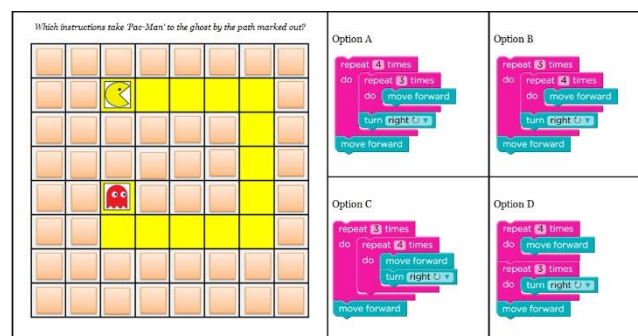


Figure 1. CTt, item n° 8 ('maze'): loops-repeat times (nested); visual blocks; sequencing.

2.2. Bebras Tasks

The Bebras Tasks are a set of activities designed within the context of the *Bebras International Contest*⁴, a competition born in Lithuania in 2003 which aims to promote the interest and excellence of primary and secondary students around the world in the field of Computer Science from a CT perspective (Dagiene & Futschek, 2008; Dagiene & Stupuriene, 2015). Each year, the contest launches a set of Bebras Tasks, whose overall approach is the resolution of 'real-life' and significant

¹ Sample copy available at: <https://goo.gl/GqD6Wt>.

² <http://www.bebras.org/>

problems, through the transfer and projection of the students' CT. These Bebras Tasks are independent from any particular software or hardware, and can be administered to individuals without any prior programming experience. For all these features, the Bebras Tasks have been pointed out to more than likely be an embryo for a future PISA (Programme for International Student Assessment) test in the field of Computer Science (Hubwieser & Mühling, 2014). As an example, one of the Bebras Tasks used in our research is shown in Figure 2.

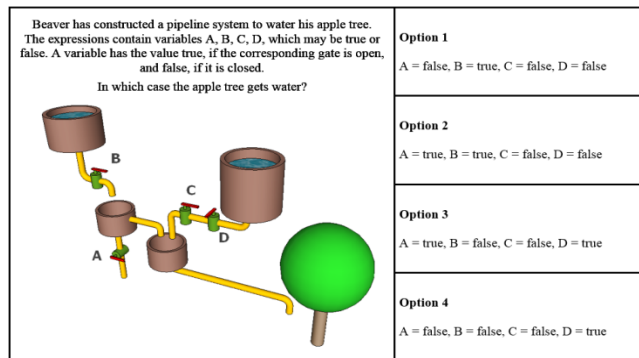


Figure 2. Example of a Bebras Task ('Water Supply').

2.3. Dr. Scratch

Dr. Scratch⁵ (Moreno-León & Robles, 2015) is a free and open source web application designed to analyze, in an automated way, projects programmed with Scratch. In addition, the tool provides feedback that middle school students can use to improve their programming and CT skills (Moreno-León, Robles, & Román-González, 2015). Therefore, Dr. Scratch is an automated tool for the formative assessment of Scratch projects.

As summarized in Table 1, the CT score that Dr. Scratch assigns to a project is based on the level of development of seven dimensions of the CT competence. These dimensions are statically evaluated by inspecting the source code of the analyzed project and given a punctuation from 0 to 3, resulting in a total evaluation ('mastery score') that ranges from 0 to 21 when all seven dimensions are aggregated.

Figure 3, which shows the source code of a Scratch project, can be used to illustrate the assessment of the tool. Dr. Scratch would assign 8 points of 'mastery score' to this project: 2 points for logical thinking, since it includes an 'if-else' statement; 2 points for user interactivity, as players interact with the sprite by using the mouse; 2 points for data representation, because the project makes use of a variable; 1 point for abstraction and problem decomposition, since there are two scripts in the project; and 1 point for flow control, because the programs are formed by a sequence of instructions with no loops. Parallelism and synchronization dimensions would be measured with 0 points.

Table 1. Dr. Scratch's score assignment.

CT dimension	Competence Level		
	Basic	Medium	Proficient

	(1 point)	(2 points)	(3 points)
Abstraction and problem decomposition	More than one script	Use of custom blocks	Use of 'clones' (instances of sprites)
Logical thinking	If	If else	Logic operations
Synchronization	Wait	Message broadcast, stop all, stop program	Wait until, when backdrop changes, broadcast and wait
Parallelism	Two scripts on green flag	Two scripts on key pressed or sprite clicked	Two scripts on receive message, video/audio input, backdrop change
Flow control	Sequence of blocks	Repeat, forever	Repeat until
User interactivity	Green flag	Keyboard, mouse, ask and wait	Webcam, input sound
Data representation	Modifiers of object properties	Variables	Lists

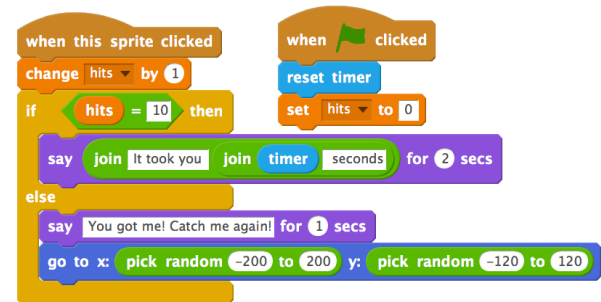


Figure 3. Source code of 'Catch me if you can 2'.

Available at <https://scratch.mit.edu/projects/142454426/>

Dr. Scratch is currently under validation process, although its convergent validity with respect to other traditional metrics of software complexity has been already reported (Moreno-León, Robles, & Román-González, 2016).

3. METHODOLOGY AND RESULTS

The convergent validity of the CTt with respect to Bebras Tasks and Dr. Scratch was investigated through two different correlational studies, with two independent samples.

3.1. First study: CTt * Bebras Tasks

Within the context of a broader pre-post evaluation of

Code.org courses, the CTt and a selection of three Bebras Tasks were concurrently administered to a sample of $n=179$ Spanish middle school students (Table 2). This occurred only in pre-test condition, i.e., students without prior formal experience in programming and before starting with Code.org.

Table 2. Sample of the first study

	7 th Grade	8 th Grade	Total
Boys	88	15	103
Girls	60	16	76

³ <http://drscratch.org/>

Total	148	31	179
-------	-----	----	-----

The three Bebras Tasks⁶ were selected attending to the following criteria: the activities were aimed to students in the range of 11-14 y/o, and focused in different aspects of CT. In Table 3, the correlations between the CTt score (which ranges from 0 to 28), the score in each of the Bebras Tasks (0 to 1), and the overall Bebras score for all of them (0 to 3) are shown. As the normality of the variables is not assured [$p\text{-value}_{(ZK-S)} > 0.05$], non-parametric correlations are calculated (Spearman's r).

Table 3. Correlations CTt * Bebras Tasks ($n=179$)

	Task #1: 'Water Supply'	Task #2: 'Fast Laundry'	Task #3: 'Abacus'	Whole Set of Tasks
CTt	.419**	.042	.490**	.519**

** $p\text{-value}_{(r)} < 0.01$

As it can be seen, the CTt has a positive, moderate, and statistically significant correlation ($r=0.52$) with the whole set of Bebras Tasks (Figure 4); and with Tasks #1 ('Water Supply', related to logic-binary structures) and #3 ('Abacus', related to abstraction, decomposition and algorithmic thinking). No correlation is found between the CTt and Task #2 ('Fast Laundry', related to parallelism), which is consistent with the fact that CTt does not involve parallelism.

3.2. Second study: CTt * Dr. Scratch

The context of this study is an 8-weeks coding course in the Scratch platform, following the *Creative Computing* (Brennan, Balch, & Chung, 2014) curriculum and involving three Spanish middle schools, with a total sample of $n=71$ students from the 8th Grade (33 boys and 38 girls).

Before starting with the course, the CTt was administered to the students in pre-test conditions (i.e., students without prior formal experience in programming). After the coding course, students took a post-test with the CTt and teachers selected the most advanced project of each student, which was analyzed with Dr. Scratch. These three measures offered us the possibility to analyze the convergent validity of the CTt and Dr. Scratch in predictive terms ($CTt_{\text{pre-test}} * Dr. Scratch$) and in concurrent terms ($CTt_{\text{post-test}} * Dr. Scratch$). As the normality of the variables is not assured either [$p\text{-value}_{(ZK-S)} > 0.05$], non-parametric correlations (Spearman's r) are calculated again (Table 4).

Table 4. Correlations CTt * Dr. Scratch ($n=71$)

	CTt Pre-test	CTt Post-test
Dr. Scratch ('mastery score')	.444**	.526**

** $p\text{-value}_{(r)} < 0.01$

As it can be seen, the CTt has a positive, moderate, and statistically significant correlation with Dr. Scratch, both in predictive ($r=0.44$) and concurrent terms ($r=0.53$, see Figure 5). As expected, the concurrent value is slightly higher because no time is intermediating among the tools.

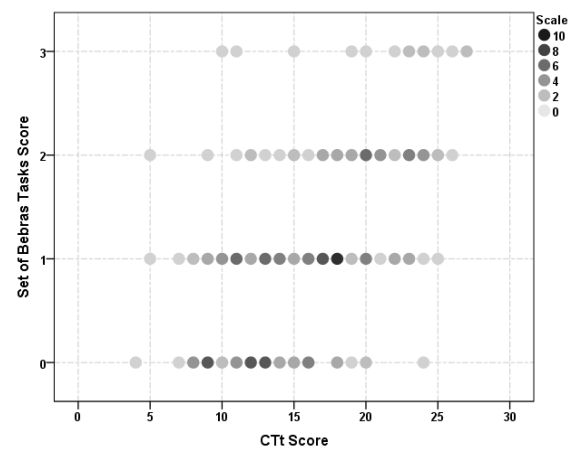


Figure 4. Scatterplot CTt * Set of Bebras Tasks.

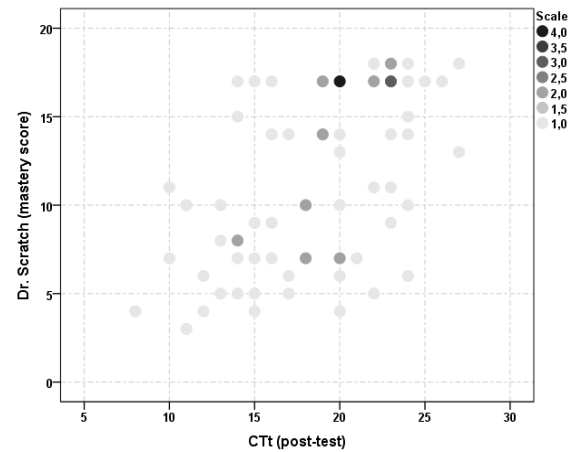


Figure 5. Scatterplot $CTt_{\text{post-test}} * Dr. Scratch$.

4. DISCUSSION AND CONCLUSIONS

Returning to our specific research questions, we have found that the CTt is *partially convergent* with the Bebras Tasks and with Dr. Scratch ($0.4 < r < 0.7$). As we expected, the convergence is not *total* ($r > 0.7$) because, although the three tools are assessing the same psychological construct (i.e., CT), they do it from different perspectives: *summative-aptitudinal* (CTt), *skill-transfer* (Bebras Tasks), and *formative-iterative* (Dr. Scratch). On the one hand, these empirical findings imply that none of these tools should be used instead of any of the others, as the different scores are only moderately correlated (i.e., a measure from one of the tools cannot substitute completely the others); otherwise, the three tools might be combined in middle school contexts. On the other hand, from a theoretical point of view, the three tools seem to be complementary, as the weaknesses of the ones are the strengths of the others.

The CTt has some strengths such as: it can be collectively administered in pure pre-test conditions, so it can be used in massive screenings and early detection of students with high abilities (or special needs) for programming tasks; and it can be utilized for collecting quantitative data in pre-

⁴ The Bebras Tasks used in our research, and their specifications, can be reviewed with more detail in: <https://goo.gl/FXxgCz>.

post evaluations of the efficacy of curricula aimed at fostering CT. However, it also has some obvious weakness: it provides a static and decontextualized assessment, and it is strongly focused on computational ‘concepts’ (Brennan & Resnick, 2012), ignoring ‘practices’ and ‘perspectives’.

As a counterbalance of the previous weakness, the Bebras Tasks provides a naturalistic and significant assessment, which is contextualized in ‘real-life’ problems that can be used not only for measuring but also for teaching and learning CT. However, the psychometric properties of these tasks are still far of being demonstrated, and some of them are at risk of being too tangential to the core of CT.

Finally, Dr. Scratch complements the CTt as the former includes ‘computational practices’ (Brennan & Resnick, 2012) that the others do not, such as iterating, testing, remixing or modularizing. However, Dr. Scratch lacks the possibility of being used in pure pre-test conditions, as it is applied to Scratch projects after the student has learnt at least some coding for a certain time.

All of the above leads us to affirm the complementarity of the CTt, Bebras Tasks and Dr. Scratch in middle school settings; and the possibility to build up a “system of assessments” (Grover, 2015; Grover et al., 2014) with all of them. Furthermore, we find evidence to consider an analogous progression between the Bloom’s (revised) taxonomy of cognitive processes (Krathwohl, 2002), and the three assessment tools considered along this paper (Figure 6).

5. LIMITATIONS AND FURTHER RESEARCH

Regarding the convergent validity of the CTt, another correlation value might have been found with Bebras Tasks if the researchers had selected a different set of them; also, another correlation value might have been found with Dr. Scratch if the teachers had selected a different set of projects. Further research should lead us to explore the convergent validity of the CTt with other assessment tools. For example, we are currently designing an investigation to study the convergence between the CTt and the Computational Thinking Scales (CTS) (Korkmaz et al., 2017), and another one that will study the convergence between Dr. Scratch and Ninja Code Village (Ota et al., 2016). As a major result of these future series of studies, it will be possible to depict a map with the convergence values between the main CT assessment tools all around the world, which ultimately would take CT to be well and seriously considered as a psychological construct.

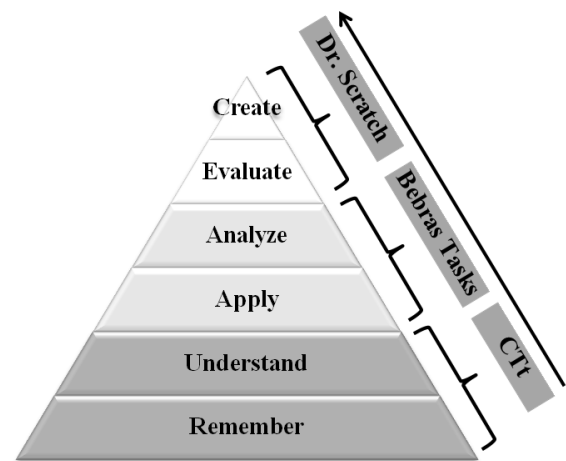


Figure 6. Bloom’s taxonomy and CT assessment tools.

6. REFERENCES

- Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 245–250).
- Bocconi, S., Chiocciariello, A., Dettori, G., Ferrari, A., Engelhardt, K., & others. (2016). *Developing Computational Thinking in Compulsory Education- Implications for policy and practice*.
- Brennan, K., Balch, C., & Chung, M. (2014). Creative computing. *Harvard Graduate School of Education*.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouv., Canada* (pp. 1–25).
- Buffum, P. S., Lobene, E. V, Frankosky, M. H., ... , & Lester, J. C. (2015). A practical guide to developing and validating computer science knowledge assessments with application to middle school. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 622–627).
- Carlson, K. D., & Herdman, A. O. (2012). Understanding the impact of convergent validity on research results. *Organizational Research Methods*, 15(1), 17–32.
- Dagiene, V., & Futschek, G. (2008). Bebras international contest on informatics and computer literacy: Criteria for good tasks. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 19–30).
- Dagiene, V., & Stupuriene, G. (2015). Informatics education based on solving attractive tasks through a contest. *KEYCIT 2014: Key Competencies in Informatics and ICT*, 7, 97.
- Grover, S. (2011). Robotics and engineering for middle and high school students to develop computational thinking. In *annual meeting of the American Educational Research Association, New Orleans, LA*.

- Grover, S. (2015). "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. In *Proceedings of the 2015 Annual Meeting of the American Educational Research Association* (pp. 15–20).
- Grover, S., Bienkowski, M., Niekrasz, J., & Hauswirth, M. (2016). Assessing Problem-Solving Process At Scale. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale* (pp. 245–248).
- Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57–62).
- Grover, S., & Pea, R. (2013). Computational Thinking in K-12 A Review of the State of the Field. *Educational Researcher*, 42(1), 38–43.
- Hubwieser, P., & Mühling, A. (2014). Playing PISA with bebras. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 128–129).
- Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A Framework for Computational Thinking Based on a Systematic Research Review. *Baltic Journal of Modern Computing*, 4(3), 583.
- Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. In *Visual Languages and Human-Centric Computing, 2010 IEEE Symposium on* (pp. 59–66).
- Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*.
- Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into Practice*, 41(4), 212–218.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61.
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, 23(3), 239–264.
- Moreno-León, J., & Robles, G. (2015). Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 132–133).
- Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 15(46).
- Moreno-León, J., Robles, G., & Román-González, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. In *Global Engineering Education Conference, 2016 IEEE* (pp. 1040–1045).
- Mühling, A., Ruf, A., & Hubwieser, P. (2015). Design and first results of a psychometric test for measuring basic programming abilities. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 2–10).
- Ota, G., Morimoto, Y., & Kato, H. (2016). Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on* (pp. 238–239).
- Román-González, M. (2015). Computational Thinking Test: Design Guidelines and Content Validation. In *Proceedings of the 7th Annual International Conference on Education and New Learning Technologies (EDULEARN 2015)* (pp. 2436–2444).
- Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*.
- Román-González, M., Pérez-González, J.-C., Moreno-León, J., & Robles, G. (2016). Does Computational Thinking Correlate with Personality?: The Non-cognitive Side of Computational Thinking. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 51–58).
- Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., ... Verno, A. (2011). CSTA K-12 Computer Science Standards: Revised 2011.
- Weintrop, D., & Wilensky, U. (2015). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *ICER* (Vol. 15, pp. 101–110).
- Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 267–272).

App Inventor VR Editor for Computational Thinking

Jane IM¹, Paul MEDLOCK-WALTON^{2*}, Mike TISSENBAUM^{2*}

¹ Korea University

² Massachusetts Institute of Technology

jane605@korea.ac.kr, paulmw@mit.edu, mtissen@mit.edu

ABSTRACT

This paper introduces the concept of a virtual reality (VR) programming environment that allows youth to both develop immersive VR experiences while enhancing computational thinking (CT). Specifically, we extended a blocks-based programming platform, MIT App Inventor, to allow youth to make VR Android apps (AI/VR). We compare AI/VR's support for CT to other existing VR editors using the CT concepts established by Brennan and Resnick (2012). Comparisons showed that AI/VR's support for all CT concepts and its ease of use for kids, makes it more preferable for teaching CT compared to other editors.

KEYWORDS

computational thinking, virtual reality, constructionism, immersive interface, MIT App Inventor

1. INTRODUCTION

In recent years, many educators have argued computational thinking (CT) (Wing, 2006) is an indispensable skill for everyone. In order to support widespread uptake of computational thinking, blocks-based approaches to programming have been developed, in which users program by snapping blocks of code. For example, Scratch allows students to build 2D multimedia (Brennan & Resnick, 2012). Alice helps students learn programming by building 3D media (Dann, Cooper, & Pausch, 2006).

Compared to Scratch, Alice provided a more immersive experience. Studies using Alice showed it is effective for learning programming, in part to its immersive nature (Sykes, 2007), indicating the potential for immersive experiences to enhance students' computational thinking. However, there has been limited research on how VR, an immersive environment, can support CT learning. Given the nascent field of VR, in order to understand the role of it in developing CT, there is a need to examine current VR editors. If these do not support the kinds of learning we wish to support, then it is critical to develop appropriate tools. This work was framed around two needs: 1) understand the state of current VR editors and examine their suitability for supporting developing computational thinking; and 2) develop a tool that supports the learning needed, if current platforms were found to be lacking.

Below, we examine current VR editors, discuss how they support CT and their suitability for young learners. We propose AI/VR that responds to their shortcomings.

2. BACKGROUND

2.1. Constructionism

Constructionism is the process of building understanding through the active use of tools to develop tangible artifacts (Kafai & Resnick, 2011). Building on constructionism, is the concept of "learning as designers", which has shown to increase higher-order thought process development and motivation (Cooper, Dann, & Pausch 2003; Fortus, Dershimer, Krajcik, Marx, & Mamlok-Naaman, 2004). Especially, programming interactive media has been shown to support CT (Brennan & Resnick, 2012). The development of interactive and immersive media, with platforms such as Alice, also embody constructionist characteristics, as they allow learners to design interactive media freely in the same context (Sykes, 2007).

2.2. Immersive Interface for Learning

Immersion is the subjective impression that one is participating in a comprehensive, realistic experience (Stanney, 2002; Lessiter, Freeman, Keogh, & Davidoff, 2001). Studies have shown that immersion in a digital environment can enhance education in at least three ways: allowing multiple perspectives, situating the learning, and transfer to other contexts (Dede, 2009). Below we describe two immersive learning environments.

2.3. Alice

Alice is a 3D graphics programming environment that allows users to create interactive 3D animations and learn programming in an object-oriented approach (Dann et al., 2006). Research on use of Alice to teach an entry level undergraduate computer science course showed that posttest performance among students who used Alice was significantly higher than comparison groups. Qualitative results showed that students using Alice enjoyed the process and spent more time engaged in the course (Sykes, 2007). There were diverse reasons for this engagement, including

the active graphical interface. While Alice is not completely immersive, it has a higher degree of immersion compared to text-based languages. The results suggest the potential for enhancing students' computational thinking skills within a more immersive environment.

2.4. Immersive Interface for Learning

Virtual Environment Interactions (VEnvI) is a platform that uses a database of dance sequences, VR, and a drag-and-drop interface to teach programming concepts

(Parmar et al., 2016). Although this study is limited because students did little programming, and were introduced to programming concepts in sessions, results showed that students found the immersion of VEnvI desirable and became more positive towards computer science.

3. PREVIOUS EDITORS

3.1. Introduction of VR Editors

In order to understand how current platforms support CT, we examined nine VR code editors: 360°& VR Editor (“360°& VR Editor”, n.d), HoloBuilder (“HoloBuilder”, n.d.), Smart2VR (“Smart2VR”, n.d.), CoSpaces (“CoSpaces”, n.d.), Vizer (“Vizer”, n.d.), Unity (“VR Overview”, n.d.), Unreal Engine VR Editor (“Unreal Engine VR Editor”, n.d.), Arma 3’s virtual reality editor (Arma 3 has a VR editor for creating games) (Zemánek, 2014), and Simmetri (“Simmetri”, n.d.).

3.2. Categorization and Analysis

We categorized the editors using Brennan and Resnick’s computational concepts, which are sequences, loops, parallelism, events, conditionals, operators, and data (Brennan & Resnick, 2012). The editors are also categorized based on their affordances into three groups, which are photo/video focused editors, visual programming editors, and text based editors (Table 1).

3.2.1. Photo or video focused editors

Photo or video focused editors are ones that: 1) focus on making rich scenes using photos and videos; and 2) only support acquiring the computational thinking concept ‘events’. The focus on scene creation is the goal of these editors, which explains why they have limited utility for CT. Users can place events inside scenes using drag-and-

drop (e.g., adding a button that is clickable). However, such editors lack the means to use sequences, loops, parallelism, events, operators, and data. 360°& VR Editor, HoloBuilder, and Smart2VR fit in this category.

3.2.2. Visual programming editors

Visual programming editors support most, if not all, the CT concepts through visual programming. Vizer and CoSpaces fall in this category, with CoSpaces also supporting text based programming.

In Vizer, students can apply all computational thinking concepts through using a ‘patch’, which can be connected to other patches (Figure 1). Users can combine patches like state/structure patches to understand sequence. There are prebuilt patches for loops, conditionals, operators, variables, and data. Users can learn parallelism, for example, by using two mouse press patches. However, there are limitations including the limited animation patches, which can make animating objects difficult. Additionally, unlike MIT App Inventor where blocks run from top to bottom, the order of patches do not indicate sequence in Vizer, requiring users to link extra patches.

In CoSpaces, users can use blocks to apply all seven CT concepts. Users can connect blocks from top to bottom to understand sequence, and use loop blocks to understand loops. The *execute in parallel* and the *on activate of* blocks enable parallelism and events in users’ projects, respectively. There are prebuilt blocks for conditionals, operators, variables and data. However, CoSpaces lacks blocks for dynamically creating objects, and has limited types of events compared to Vizer and AI/VR. These can limit the range of computational practices (which focus on “how”, instead of “what” users learn) (Brennan & Resnick, 2012).

Table 1. Categorization of VR Editors with colored boxes representing an attribute(column) that an editor(row) has. Numbers 1,2,3,4,5,6,7 of CT concept each refer to sequences, loops, parallelism, events, conditionals, operators, and data.

Platform	Editor Type	CT Concepts							Does not require programming background	Intended audience
		1	2	3	4	5	6	7		
360°& VR Editor	Photo/Video									Novices
HoloBuilder										Novices, especially construction company
Smart2VR										Novices
Vizer	Visual									Novices
CoSpaces	Visual/Text based									Novices
Unity	Text based									Professional, Experienced gamers
Unreal Engine VR Editor										Professional, Experienced gamers
Arma 3’s editor										Professional, Experienced gamers
Simmetri										Artists
AI/VR	Visual									Novices

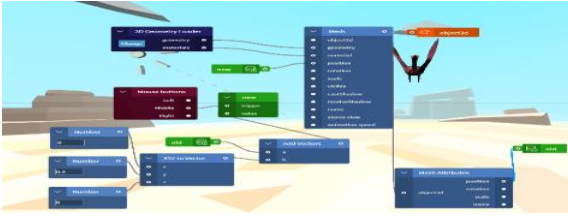


Figure 1. Usage of patches in Vizor.

3.3. Text-based editors

Text based editors are editors that require at least partial text based programming to exhibit the seven CT concepts. Unity, Unreal Engine VR Editor, Arma 3's virtual reality editor, and Simmetri fit in this category, with differing audiences (Table 1). Since these editors allow users to build complicated VR environments technically, they support all CT concepts, and users can employ complex CT with them. However, these text-based editors require a steep learning curve and are not suitable for beginners.

Building off of the various shortcomings of the tools described above, we identified a gap in the VR authoring landscape for a tool that allows novices to develop VR applications while developing CT concept understandings. Below we describe the tool and its use.

4. APP INVENTOR VR EDITOR

4.1. Blocks in AI/VR

There are four kinds of blocks in AI/VR: 1) **event**, 2) **method**, 3) **property setter and getter**, and 4) **object creation** blocks. **Event** includes *checkButton*, which checks if the user clicks the Cardboard button. **Method** blocks trigger interactions with objects and the player, including *moveUser*, which changes the location of the player. **Property setters and getters** change object's attributes, such as size. **Object creation** blocks, such as *createCube*, allow the user to dynamically add objects.

4.2. Sample AI/VR program

To demonstrate how the editor supports CT, we included a sample AI/VR program (Figure 2 & Figure 3). If the user gazes at one of the four cubes (that are made by shaking the phone), she will earn points (shown in a label). The cube also moves to a random position and changes color.

This example shows how all seven computational concepts are supported in AI/VR. First, users can understand sequences by checking that blocks are executed in order when the user gazes at a cube. The cube changing color, and the score increasing and updating shows sequentially (① of Figure 3). Loops are used to iterate over cubes in ②. Events are used through blocks like *checkGazeShort* (the block that returns 1 if the user gazed at the object - ③). For parallelism, two "if" blocks are used to check whether a cube was gazed at and the color and position of the cube are changed concurrently (④). For conditionals, the user can connect event blocks and attribute or animation related blocks with an "if" block (⑤). For operators, users can practice addition by adding 1 to the current score when a cube is gazed at (⑥)-

1) and checking that the increased score is updated in the scene (⑥-2). Lastly, users can understand data by keeping track of cubes using a list block (⑦).



Figure 2. Scene of demo

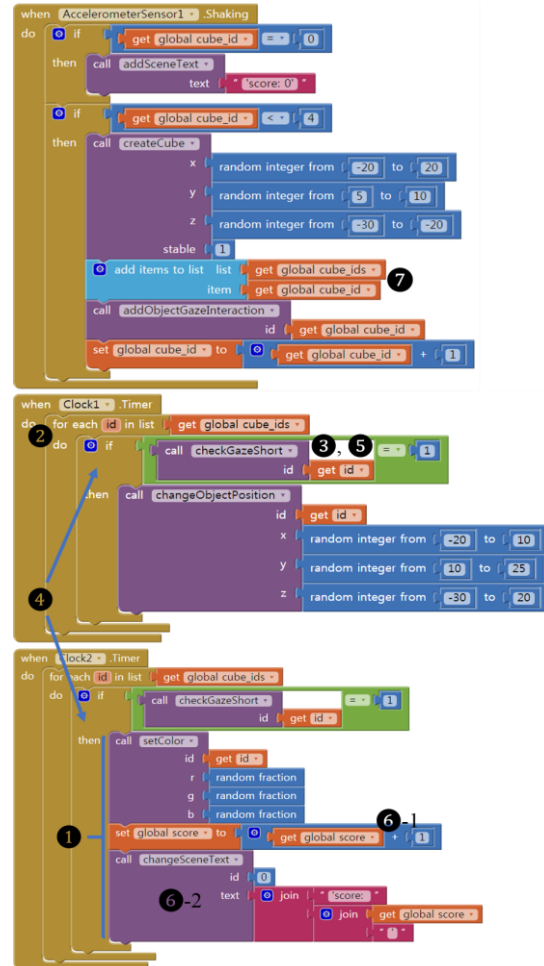


Figure 3. Code for a sample VR program in AI/VR.

4.3. Categorization and Analysis

AI/VR is in the category of visual programming editors and supports all CT concepts. AI/VR also targets ease of use by kids, employing the same drag-and-drop interface as the original MIT App Inventor (Wolber, Abelson, Spertus, & Looney, 2011). Considering these aspects, AI/VR balances usability for kids and features for supporting CT, overcoming the limitations of other editors. It also has animation blocks such as *moveObject*, overcoming the limitation of Vizor. Using the top down approach, sequences are also easier in AI/VR. Compared to CoSpaces, AI/VR supports creating new objects with blocks like *createCube*, and allows diverse triggering events with blocks like *checkButton*, which triggers an event when a user presses the Cardboard headset button.

However, AI/VR lacks a diversity of objects and media related blocks like video. In context of CT, this could be

a limitation because it could reduce the diversity of computational practices.

5. CONCLUSION

Considering the role constructionism plays in computational thinking and the possibility of immersive virtual reality to support this learning, we introduce AI/VR - a blocks-based tool to support kids to more easily create virtual reality apps. We have shown AI/VR's fit as a visual programming editor that supports all seven of Brennan and Resnik's CT concepts, while also being usable by young kids. Although AI/VR has limited diversity in objects and media related blocks, it overcomes the limitations of Vizor, such as animating objects and sequence, and those of CoSpaces such as the lack of blocks for dynamically creating objects and the limited types of triggers for events.

6. ACKNOWLEDGEMENT

We would like to thank Hal Abelson, Professor of EECS at MIT, whose insight was great help to this research.

7. REFERENCES

- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association (AERA 2012)*.
- Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. *Proceedings of the 34th SIGCSE technical symposium on Computer science education - SIGCSE '03*.
- CoSpaces. (n.d.). Retrieved from <https://cospaces.io/create.html>
- Dann, W., Cooper, S., & Pausch, R. (2006). *Learning to program with Alice*. Upper Saddle River, NJ: Pearson Prentice Hall.
- Dede, C. (2009). Immersive interfaces for engagement and learning. *Science*, 323(5910), 66-69.
- Fortus, D., Dershimer, R. C., Krajcik, J., Marx, R. W., & Mamlok-Naaman, R. (2004). Design-based science and student learning. *Journal of Research in Science Teaching*, 41(10), 1081-1110.
- HoloBuilder (n.d.). Retrieved from <http://landing.holobuilder.com/construction>
- Kafai, Y. B., & Resnick, M. (2011). *Constructionism in practice: Designing, thinking, and learning in a digital world*. New York, NY: Routledge.
- Lessiter, J., Freeman, J., Keogh, E., & Davidoff, J. (2001). A cross-media presence questionnaire: The ITC-Sense of Presence Inventory. *Presence: Teleoperators and Virtual Environments*, 10(3), 282-297.
- Parmar, D., Isaac, J., Babu, S. V., D'souza, N., Leonard, A. E., Jorg, S., . . . Daily, S. B. (2016). Programming moves: Design and evaluation of applying embodied interaction in virtual environments to enhance computational thinking in middle school students. *2016 IEEE Virtual Reality (VR)*, 131-140.
- Simmetri. (n.d.). Retrieved from <http://simmetri.com/>
- Smart2VR. (n.d.). Retrieved from <https://www.smart2vr.com/#how-it-works>
- Stanney, K. M. (2002). *Handbook of virtual environments: design, implementation, and applications*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Sykes, E. (2007). Determining the effectiveness of the 3D Alice programming environment at the computer science I level. *Journal of Educational Computing Research*, 36(2), 223-244.
- 360° & VR Editor. (n.d.). Retrieved from <http://demo.thinglink.com/vr-editor>
- Unreal Engine VR Editor. (n.d.). Retrieved from <https://docs.unrealengine.com/latest/INT/Engine/Editor/VR/>
- Vizor. (n.d.). Retrieved from <http://vizor.io/about>
- VR Overview. (n.d.). Retrieved from <https://unity3d.com/kr/learn/tutorials/topics/virtual-reality/vr-overview>
- Wang, T., Mei, W., Lin, S., Chiu, S., & Lin, J. M. (2009). Teaching programming concepts to high school students with Alice. *2009 39th IEEE Frontiers in Education Conference*, 1-6.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor – Create Your Own Android Apps*. Sebastopol, CA: O'Reilly.
- Zemánek, J. (2014, July 15). Arma3: Virtual Reality Custom Courses. Retrieved January 10, 2017, from https://community.bistudio.com/wiki/Arma3:_Virtual_Reality_Custom_Courses

Computational Thinking and Coding Initiatives in Singapore

Peter SEOW¹, Chee-kit LOOI¹, Bimlesh WADHWA², Longkai WU¹, Liu LIU¹

¹ National Institute of Education, Nanyang Technological University, Singapore

² National University of Singapore, Singapore

peter.seow@nie.edu.sg, cheekit.looi@nie.edu.sg, bimlesh@nus.edu.sg, longkai.wu@nie.edu.sg, liu.liu@nie.edu.sg

ABSTRACT

Many countries that recognize the importance of Computational Thinking skills are implementing curriculum changes to integrate the development of these skills and to introduce coding into formal school education. Singapore has introduced new programmes from Pre-school to Secondary children to develop Computational Thinking skills and introduce programming. A major change in the secondary school syllabus is the introduction of a new Computing subject taken at “O” levels. The new subject emphasizes on the development of Computational Thinking skills and coding in Python. Students are expected to apply technology for creating solutions to solve problems. In this paper, we describe the various initiatives in Singapore for Pre-school, Primary and Secondary schools. From initiatives in these three school going groups, we review Singapore’s approach to implementation of learning Computational Thinking. Unlike several countries that has decided to implement computing as compulsory education, Singapore has taken a route of creating interest amongst children in Computing in age-appropriate ways. Singapore’s pragmatic approach is characterized by opt-in by schools, nurturing students’ interest in computing, upskilling teachers in computing, and a multi-agency approach.

KEYWORDS

Computational Thinking, Computing, Programming, and Coding

7. INTRODUCTION

Since Wing’s (2006) argument on how computational concepts, methods and tools can develop thinking skills to transform how we work or solve problems, and with the emergence of computation-related fields such as Data Science and Artificial Intelligence in recent years, there has been great interest from academia, industry and government in Computational Thinking (CT) and coding. Sites such as code.org, which is sponsored by industry giants like Google, provide free resources on learning coding to anyone who is interested. National governments in addressing the manpower needs arising in the shift from a knowledge/information economy to an economy driven by computation, are introducing educational policies that would prepare its citizens to be future ready. Computer Science and computing education once were only available as courses at the University level. Wing (2017) reflecting 10 years after her publication on CT, never dreamt that Computer Science education would be taught in K-12 on a large scale. Today, it has become reality as governments or educational authorities, and schools are introducing Computer Science education in the different

levels of education. This paper describes Singapore’s effort in the introducing the CT and coding in the education from Pre-school to Secondary schools.

8. COMPUTING PROGRAMMES in K-10

In 2014, Singapore launched the Smart Nation Programme which is a nationwide effort to harness technology in the business, government and home sectors for improving urban living, building stronger communities, growing the economy and creating opportunities for all residents to address the everchanging global challenges (Smart Nation, 2017). One of the key enablers for the Smart Nation initiative is to develop computational capabilities. Programmes are implemented to introduce and develop CT skills and coding capabilities from pre-school children to adults. We survey the landscape of K-10 CT and coding related programmes in Singapore which are implemented by various government organizations. We present these programmes and have organised them according the groups: Pre-school, Primary and Secondary.

8.1. Pre-school

In Singapore, children aged from 3 to 6 years old attend pre-schools which are mostly privately run. The Infocomm Media Development Authority (IMDA) launched the Playmaker initiative with the aim of introducing Computational Thinking in the Kindergarten and Pre-schools in Singapore (IMDA, 2017). There are over 3000 pre-schools in Singapore and initial phase was to pilot the program in 160 pre-schools. IMDA’s approach to introducing CT was to identify toys that would engage young children in play while developing CT skills such as algorithmic thinking. IMDA would provide a set of the toys to pilot centres for use in the classroom by the teachers.

The toys that IMDA selected that would provide playful exploration of technology are: 1) Beebot; 2) Circuit Stickers; and 3) Kibo. The Beebot is a toy with simple programmable steps to control the movement. Children can program the toy to move in a path by logically sequencing the number of steps to move and direction. Playing Beebot can help young children to develop problem solving skills and logical thinking as they plan and program the movement of the toy. With the Kibo which was developed by researchers in Tuft University, children can create a sequence of instructions by arranging Kibo wooden blocks. The blocks can be scanned in the sequence with the instructions passed to the robot to execute the steps. Circuit sticker is a toolkit comprising of peel-and-stick electronic components such as LEDs and conductive copper tapes. With the toolkit, young children can create interactive art and craft projects embedded with

LED stickers and sensors that respond to the environment or external stimuli (See Figure 1). Children can be creative in hands-on activities while learning and applying basic electricity concepts.

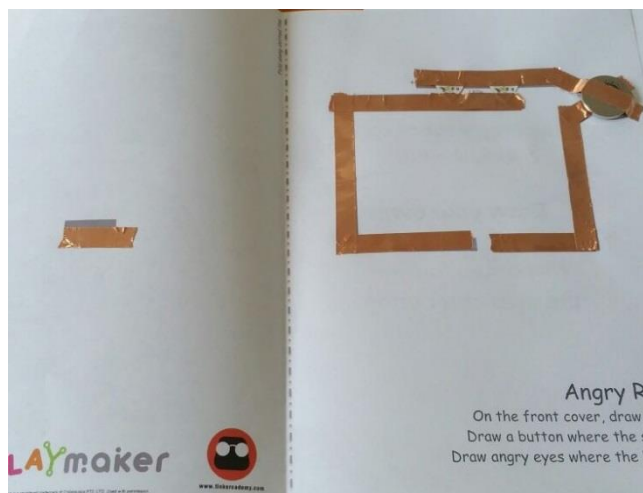


Figure 1. Circuit Stickers

Pre-school teachers in Singapore do not use much technology or handle technology in the classroom as the emphasis is more on literacy development and play. As a result, they may have apprehensions or concerns in using technology in their lessons. To address teachers' lack of experience and concerns, IMDA organised teacher seminars and workshops for teachers to experience the use of the Beebot, Kibo and Circuit Stickers. The hands-on sessions were facilitated by the instructors to introduce teachers to the tech toys and work on simple projects. The workshops are for the teachers to understand the potential learning opportunities by learning the technology for themselves. Hands-on sessions also help to alleviate any potential fear of handling technology as they experience the use of the technology with the support from instructors.

In preparing to pilot the Playmaker program and address the concerns of Pre-school teachers, IMDA worked with a local Polytechnic which offers pre-school training for teachers. At the Preschool Learning Academy, pre-school lecturers and trainers, with technologists worked to trial the use of the various tech toys in the pre-school classroom. Their learning experiences were shared with the teachers. Collaborating with the pre-school training academy provides implementers to understand how the tools can be used in the classroom and build capacity among the trainers to work with the teachers how these tools can be used to develop. The academy can provide on-going professional development to the current and new teachers.

8.2. Primary Schools

To expose and enthuse Primary school students in computational thinking, IMDA introduced its Code for Fun enrichment programme which was piloted in 2014. Since 2015, the programme has been implemented in 110 schools with about 34,000 students participating. The goals of the programme is to expose a large base of

students to CT concepts and coding, and build a generation of workforce equipped with basic coding and CT skills. To scale the enrichment programme, IMDA invited technology training partners to propose 10 hour programs that would include coding activities using visual-based programming language such as Scratch and combining it with a robotic kit such as the MoWay or microcontrollers such as the Arduino. The proposed programs should help students appreciate coding and develop CT skills such as solving problems and thinking logically. Schools that are interested in the Code for Fun programme can select from the list of vendors and apply for funding from IMDA to run the programme in the school. At present, IMDA fund 70% for each student with the rest funded by the school on the condition that a certain number of students will be attending the programme. Teachers are also required to attend a course conducted by the technology vendors on the programme. IMDA envisions the program to be taught by the teachers in the future. Currently, each 10-hour session is conducted by the technology trainers in the school lab. In each session, students are introduced computing concepts such as the use of variables and conditional students through the use of visual programming tools such as Scratch. Students also use the Robotic tools such as the Lego WeDo kits or MoWay robot based on the proposal by the different training partners. Schools can choose on the different tools offered by the various trainers based on their students' interest and budget.

The Code for Fun enrichment and Playmaker programme is part of the Code@SG movement initiated by the government to teach CT and coding to students from an early age. Driven by the IMDA, the initiative is important to build Singapore's national capability in a skilled workforce by creating interest in the Computational skills and promoting Infocomm as a career choice. A multi-pronged approach of working with different partners involves the development of enrichment programmes, school infocomm clubs and coding competitions.

8.3. Secondary Schools

In 2017, the Ministry of Education introduced a new Computing subject which will be offered to students as an "O" Level subject replacing the existing Computer Studies subject (MOE, 2017). Students taking the subject will be learning to code in Python which is taught only at "A" Level Computing. In the new syllabus design, students will develop CT and coding skills to create solutions with technology to solve problems. In the old Computer Studies syllabus, students were learning to be users of technology such as using software applications and understanding aspects of technology. This marks a distinct shift from a learning to be user of the technology to creator of solutions with technology.

The new Computing syllabus is built on the framework shown in Figure. 2: 1) Computer as a Science; 2) Computer as a Tool; and 3) Computer in Society.

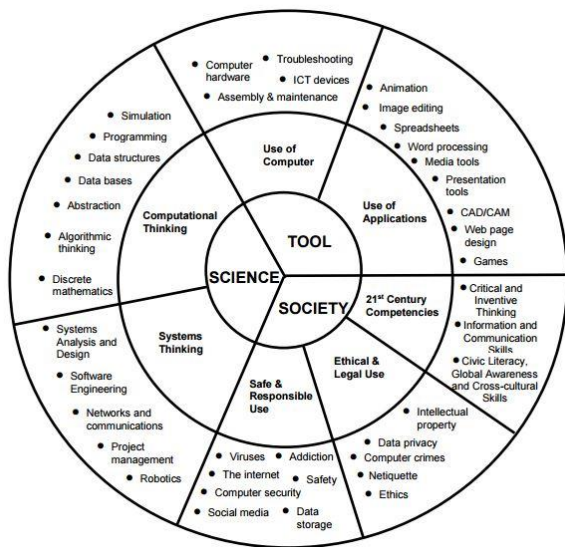


Figure 2. Computing Education Framework

The dimension of Computer as a Science comprises of the core components of Computational and Systems Thinking. Students will develop and apply CT skills such as abstraction and algorithmic thinking to solve problems and develop solutions through coding. Using both CT skills and systems thinking, students are required to work on a project of their own interest. This is to encourage students to take more ownership by identifying a problem that they are interested and developing ideas to solve the problem using programming tools. In the dimension of Computer as a Tool, students are exposed to the use of hardware, technology, and devices that are used in the everyday aspects of life at work and play. They learn about computer applications that are used for productivity, communications and creative tools for completing specific tasks such as video editing or creating websites. In Computer in Society, students learn about issues in using computers such as intellectual property, data privacy, internet security and the computer addiction. This dimension includes a component on 21st Century Competencies to prepare students to be Future-ready workers in the use of technology for self-directed learning, working in collaboration with others and fostering creativity.

A current challenge in implementing a Computing curriculum is equipping teachers to teach the subject as there are only few teachers who have Computing or Computer Science background. Teachers who are interested in teaching Computing and programming attend a year-long conversion course taught by Computer Science faculty from a University. The goal of the course is to prepare and equip teachers with the content and technical knowledge to teach computing. In addition to preparing teachers for the new Computing curriculum, Ministry of Education's Curriculum Planning and Development Division (CPDD) organised workshops for teachers to understand the aspects of the syllabus. Teachers are introduced to different pedagogies for teaching computing such as unplugged approaches and paired programming. In the workshop, teachers experienced the use of the tools for teaching such as the

Raspberry Pi. The workshop was a platform for teachers to raise their concerns about teaching the subject such as the project work for students.

9. Singapore's Approach in Computing

Singapore's approach is to provide opportunities for students to develop their interests in coding and computing skills through touchpoint activities at various ages as shown in Figure 3. Computing and CT skills are introduced to the children that are age-appropriate and engage them in learning. Children progressively develop interest and skills leading them to offer Computing as a subject in the "O" levels.

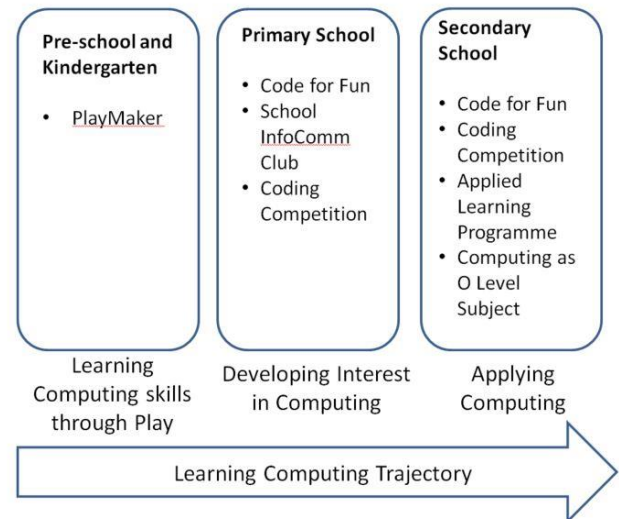


Figure 3. Learning Computing in Singapore

The following sections describe the characteristics of the approach.

9.1. Opt-in by Schools

Singapore uses an opt-in model recognizing the agency of each school in choosing programs to meet the needs of their students and readiness of the teachers. School-based programmes are planned by the school and teachers that would build students' interest and skills in identified areas like Computing. As teachers play in pivotal in the role in implementing the programmes, there must be buy-in from the teachers to see the importance of the programmes for the students. For the schools to opt-in to adopt computing, there must be teachers within the school to be ready to learn, experiment and implement

9.2. Nurturing Interest in Computing

Singapore's approach is to nurture interest at early age. Pre-school children are developing problem solving and logical thinking skills through play. Toys like the Beebot and Circuit Stickers are age-appropriate for the children to be engaged in play in their lessons while developing the computational thinking skills. In primary and secondary schools, students are introduced to visual programming tools like Scratch and tangible computing tools like the MoWay robots and Lego WeDo. Lessons are designed for children to have fun learning programming and

developing logical thinking skills. Leading to the O Levels, students can offer Computing as a subject based on their interest and choice. Starting from Pre-school, a pipeline is created for students to develop interest and computational thinking skills for them to choose Computing rather than making it compulsory learning.

9.3. Upskilling Teachers in Computing

To prepare teachers to develop logical thinking, algorithmic thinking, problem solving and coding skills in the lessons, professional development and support must be given. The professional development should be appropriate to the learning needs of the teachers to prepare them to teach their students. For the pre-school, a form of learning was for them to experience play with the toys and understand how their own children would learn from playing. Support from IMDA is given to the teachers to help them design and implement the lessons in the classrooms. In secondary school, Computing teachers undergo an intensive computing course equipping them with computer science concepts and coding skills for teaching students. Most of these teachers are non-Computer Science graduates but volunteered for the conversion course out of their own interest. The teachers have regular meet-ups to continue improving their knowledge in teaching computing.

9.4. Multiple-Agency Approach

The task of building CT and Computing skills takes the combined effort of multiple agencies to work together. These agencies include the government agencies like IMDA, Ministry of Education and the Ministry of Social and Family Development, Education centers like the Singapore Science Centre, Universities and educational providers. These agencies work together or singly to organize opportunities for children to learn computational thinking skills providing them with varied experiences. The agencies can pool resources such as funding and support for initiating, implementing and sustaining the programmes.

10. SUMMARY

Singapore has taken a pragmatic approach in the implementation of the learning and development of CT skills and coding. Taking such an approach provides

children with opportunities to generate interest in learning computing. Starting at an early age, children are exposed to developing CT skills through age-appropriate ways of playing. In primary school, children learn through fun and given opportunities to extend their interest in programming through clubs and coding competition. At the secondary school level, children can choose to pursue Computing as a subject. Schools can opt-in to offer programmes based on the students' needs, schools' niche programmes and readiness of the teachers to teach computing. Teachers who are keen can choose to extend their capacity to teach computing. Singapore as a Nation can harness various agencies to work together in providing a variety of learning experiences for children to be engaged in learning computing.

11. REFERENCES

- IMDA. (2017). *PlayMaker Changing the Game*. Retrieved Feb 13, 2017, from <https://www.imda.gov.sg/infocomm-and-media-news/buzz-central/2015/10/playmaker-changing-the-game>
- [27] MOE. (2017). O Level Computing Teaching and Learning Syllabus. Retrieved Feb 13, 2017, from <https://www.moe.gov.sg/docs/default-source/document/education/syllabuses/sciences/files/o-level-computing-teaching-and-learning-syllabus.pdf>
- [28] Smart Nation. (2017). *Why Smart Nation*. Retrieved Feb 13, 2017, from <https://www.smartnation.sg/about-smart-nation>
- Wing, J. M. (2006). *Computational Thinking*. Communications of the ACM, 49(3), 33-35
- Wing, J. M. (2017). Computational Thinking, 10 years. Retrieved Feb 13, 2017, from <https://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later>

Acknowledgements

The work reported in this paper is funded by EFRP grant OER 04/16 LCK.

Enabling Multi-User Computational Thinking with Collaborative Blocks

Programming in MIT App Inventor

Xinyue DENG¹, Evan W. PATTON^{1*}

¹ Massachusetts Institute of Technology, Cambridge, MA, USA
{dxy0420, ewpatton}@mit.edu

ABSTRACT

Collaboration becomes increasingly important in programming as projects become more complex. With traditional text-based programming languages, programmers typically use a source code management system to manage the code, merge code from multiple editors, and optionally lock files for conflict-free editing. There is a limited corpus of work around collaborative editing of code in visual programming languages such as block-based programming. We propose an extension to MIT App Inventor, a web-based visual platform for building Android applications with blocks, which will enable many programmers to collaborate in real-time on MIT App Inventor projects. We take the position that real-time collaboration within MIT App Inventor will encourage students in a group environment to interact with one another in ways that help them improve each other's understanding and practice of computational thinking practices that may not be achieved in the traditional one user-one project paradigm that is currently provided.

KEYWORDS

Real-time collaboration, App Inventor, visual programming, computational thinking

1. INTRODUCTION

Cloud-based collaborative technologies such as Google Docs have become a central part of how teams work together to collaborate in real time on all manner of content. While real-time collaboration for programming has been explored in research settings, a typical editing pattern in software development involves developers working separately and then merging their changes through a source code management system, such as Subversion or Git. These solutions work well for textual programming languages. However, little work has been done exploring real-time collaborative techniques for visual programming languages, including blocks-based languages including Scratch (Maloney, Resnick, Rusk, Silerman, & Eastmond, 2010) and MIT App Inventor (Wolber, Abelson, Spertus, & Looney, 2011). The remainder of this paper will focus on the challenges and possible benefits of real-time collaboration as they relate specifically to the MIT App Inventor software.

MIT App Inventor is a web-based platform for building mobile phone applications targeting Android. It provides two editors for building an application: a designer where users drag and drop *components*, such as buttons, to lay

out the user interface of an application, and the *blocks* editor where program logic is provided using a puzzle block-like language based on Google's Blockly. MIT App Inventor users require a Google account to identify themselves to the service and projects are tied to these accounts. While it is possible to perform group collaboration in MIT App Inventor given its current implementation, this is usually accomplished by student groups creating a shared Google account and trading off control over who is editing using the single account.

We propose a collaborative programming environment within the MIT App Inventor software that will enable multiple users to engage in computational thinking in a real-time collaborative manner. Section 2 describes the related work in computational thinking and collaborative programming. Section 3 illustrates our design and implementation of the collaborative environment. Section 4 presents a discussion that how this system can help users engage in computational thinking.

2. RELATED WORK

Brennan & Resnick (2012) gauge computational thinking with respect to three categories: computational concepts, computational practices, and computational perspectives. They defined "Connecting" as one of the computational perspectives, which involves programming with others and programming for others. By collaboration, programmers are able to accomplish more than what they could have on their own.

With text-based programming languages, programmers usually collaborate with a version control system, such as Git. Guzzi, Bacchelli, Riche, and Van Deursen(2015) presented an improved IDE with support of version control system to help programmers to resolve conflicts and detect problems introduced by others' code. Other than version control system, Goldman, Little, & Miller (2011) demonstrated a real-time collaborative web-based editor for the Java programming language.

Collaboration in blocks-based programming languages has typically been done via remixing, such as in the Scratch language (Maloney et al., 2010) and MIT App Inventor (Wolber et al., 2011). In remixing, a developer publishes an application publicly and others use it as a starting point for a new application. This remixing behavior makes iterate development between two developers more difficult because the project, rather than some subset, is the basis for remixing.

Greenberg & Gutwin (2016) highlight key challenges in enabling awareness in collaborative environments. We

leverage their findings by codifying awareness information via the locking mechanisms proposed in Section 3. These locking mechanisms allow users to direct awareness of their peers by synchronizing access to the environment on a per user basis. Gross (2013) provides a more in-depth review of awareness research.

3. DESIGN AND IMPLEMENTATION

Our collaboration system is mainly designed for group course projects of 2-4 students in middle school, high school, or college. The system will satisfy the following features:

1. Users are identified by their email address and share projects with others by email address. The user who creates the project can change others' access level of the project. The access level includes read, in which users can only view the project, and write, in which users can both view and edit the project.
2. Users can know who is currently working on the project, and the components or blocks that each individual is currently working on.
3. User can see others' changes simultaneously. There are several cases in MIT App Inventor:
 - a. When users work on different screens, their changes will not be shown until switching screens.
 - b. When users work on the same screen, and they work on the same editor, they can see the others' change immediately on the editor.
 - c. When two users work on the same screen, and one works on the designer editor, and the other works on the blocks editor, the one on the blocks editor can see new blocks when the one on the designer editor adds a new component. When the one on the designer editor removes a component, the other will see blocks related to that component disappear.

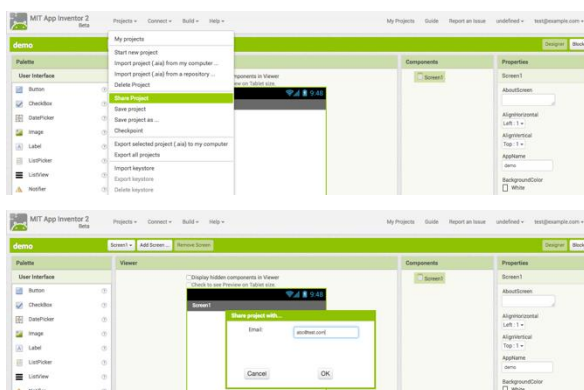


Figure 1. Share project by entering user's email address

3.1. User Interface Design

A user can share a project with others by providing their email address. Figure 1 shows the user interface of sharing a project. Once the project is shared successfully, the other user can find the project in her project explorer. Users can know who has opened the project by the colored square in the project title bar. When user hovers on the square, it will show the user's email address. The color of the square indicates the user's color. It is used to identify which part of the program a user is editing.

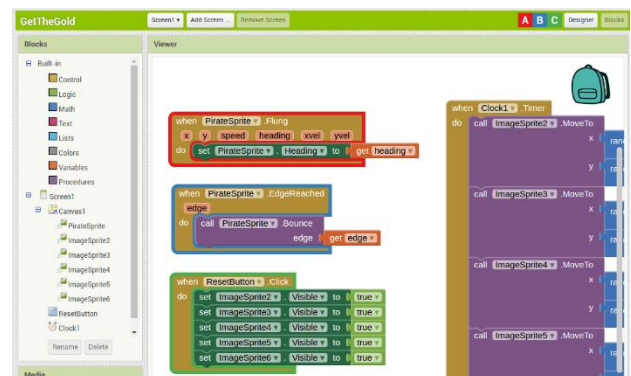


Figure 2. An example of collaborative block-based programming in MIT App Inventor. This project is shared within four users. The user can see the other three users, A, B and C, on the project title bar. The block that each user is editing is highlighted with the user's color.

3.2. Collaboration Server

In order to show others' changes immediately, we use publish-subscribe pattern to send updates from one user to others. Publish-subscribe pattern is a messaging pattern, where senders can send messages to a channel, and receivers who subscribe to that channel can receive the messages. We decided to build a NodeJS server for web clients to communicate about collaboration, which runs separately from the MIT App Inventor server, so it is easy to be managed. MIT App Inventor clients connect the collaboration server with sockets. We use Redis, an open source library for in-memory data structure store and publish-subscribe pattern, to publish and subscribe updates (Redis Contributors 2017), and all messages will be in JSON format. The client will translate changes into JSON documents and send them to the collaboration server over a specified channel. The server will apply operational transformations on JSON documents to make sure changes are published consistently to all subscribed clients. Then, clients translate JSON document into events that update the code and run the events on their individual systems. Therefore, the copies of the code of all clients will eventually be identical.

3.3. Channels

Each MIT App Inventor client is both publisher and subscriber in the system. Clients will subscribe to three kinds of channels:

1. User channel: The user channel is specified by the user email address. Each client

subscribes to only one user channel. When users share a project, they publish the project and user information to others' user channel. Therefore, other users will be notified that a user shares a project with them, and that project will appear in their project list.

2. Project channel: Project channel is specified by project id. (Each MIT App Inventor project has an id that is unique to the MIT App Inventor server.) When a collaborator opens a project, he will subscribe to that project channel. This channel is used for project-level messages, such as when other collaborators open or close the project, or when components are added, modified or removed. When a collaborator publishes changes to the project channel, all active collaborators on that project will be notified.
3. Screen channel: The screen channel is specified as combination of the project id and the screen name. This channel is used to publish changes about blocks. Each screen has its set of blocks. Users subscribe to this channel when they open the block editor of a screen. After subscribing this channel, all changes related to blocks in this screen will be published to the channel.

4. DISCUSSION

The collaborative programming environment within MIT App Inventor provides users a new approach to teach and learn. For example, it enables “teacher-student” or “mentor-mentee” roles inside MIT App Inventor. Teachers can share the projects with students in read-only mode to demonstrate ideas and demos. Students can work on group projects after school, because they can collaborate remotely. As MIT App Inventor is built for students and novice programmers, the collaborative programming environment gives them an opportunity to develop their teamwork skill at an early stage. Also, while developing applications collaboratively, users can learn how to resolve conflicts. In addition to the commonly used pair programming method, our collaborative programming environment introduced a new mode of cooperation between students. Instead of sitting shoulder-to-shoulder and working on the same

Goldman, M., Little, G., & Miller, R. C. (2011, October). Real-time collaborative coding in a web IDE. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 155-164). ACM.

machine, students can work on the different machines in distributed locations and review others' changes simultaneously.

This new collaboration mechanism for MIT App Inventor touches on all four of the key computational thinking practices of Brennan and Resnick (2012). Multiple users can incrementally and iteratively build small units either in isolation or together depending on the complexity of the tasks and expertise of the individuals. Users can explore different debugging techniques to assist one another in addressing problems in the code. Reuse and remix of code can happen on a much finer time granularity on the order of seconds or minutes. Lastly, users can work together to help one another understand and exploit abstraction and modularization techniques within a program.

One challenge for collaborating with visual programming language is that it is hard to understand others' thought process. With the text-based programming language, programmers can know others' plan via comments. However, it is hard to place comments in visual programming environment without disrupting actual programming logic. One way we can handle it is to add a screen for comments, so users can toggle the comments screen as they need. Another way to help users to understand others is adding a communication channel, so that users can exchange their ideas while they are programming.

Our technical approach is not restricted to MIT App Inventor, as it builds on Google's Blockly. It can therefore be applied to other visual programming languages, such as Scratch. It is easy to integrate socket and publish-subscribe pattern into the system.

5. CONCLUSIONS

We presented a collaborative programming environment within the MIT App Inventor software and provided technical details of an implementation of real-time collaboration. In future work, we will evaluate the effectiveness of the collaboration with novice and expert users of MIT App Inventor to better understand how students use the system to collaborate.

6. REFERENCES

- Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (pp. 1-25).
- Greenberg, S., & Gutwin, C. (2016). Implications of we-awareness to the design of distributed groupware tools. *Computer Supported Cooperative Work (CSCW)*, 25(4-5), 279-293.

- Gross, T. (2013). Supporting effortless coordination: 25 years of awareness research. *Computer Supported Cooperative Work (CSCW)*, 22(4-6), 425-474.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4), 16.
- Redis Contributors (2017). *Redis Publish-Subscribe message pattern*. Retrieved February 4, 2017 from <https://redis.io/topics/pubsub>.
- Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor*. O'Reilly Media, Inc.
- Guzzi, A., Bacchelli, A., Riche, Y., and Van Deursen, A. (2015). Supporting Developers' Coordination in the IDE. *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15*

Evidences of Self-Development of TAs in CT Education

Ray CHEUNG¹, Ron Chi-wai KWOK*, Matthew LEE, Robert LI, Chee-wei TAN

City University of Hong Kong, Hong Kong

¹r.cheung@cityu.edu.hk, isron@cityu.edu.hk, ismatlee@um.cityu.edu.hk,
Robert.Li@cityu.edu.hk, cheewtan@cityu.edu.hk

ABSTRACT

In the context of integrating Computational Thinking (CT) in Primary School Education, we examine the self-development of undergraduate students during their engagement as Teaching Assistants (TAs) in CT Education. More specifically, we propose to adopt the stress-adaptation-growth process of the Intercultural Transformation Theory (ITT) as a framework to provide evidences of the self-development of TAs in the CoolThink@JC project of Hong Kong. The collected data confirms the evidences of the stress-adaptation-growth process of TAs engagement, which helps transforming undergraduate students into co-teachers with commitment to future civic involvement.

KEYWORDS

Computational Thinking, Teaching Assistant, Co-Teaching, Stress-Adaptation-Growth, Service Learning

1. SUMMARY OF CITYU INVOLVEMENT IN CT EDUCATION

Coding and computing-related skills are vital in the information age both for personal and social development. In this project, the City University of Hong Kong (CityU) team aims to provide professional education support to enhance coding literacy among Hong Kong citizens through a series of elaborate teaching and learning activities, in particular targeting the primary school student group in our population.

Coding is now a global initiative in multiple countries, such as the “Hour of Code” campaign is first initialized by Code.org in the US in 2013, providing free educational resources for all ages. Now, over 100 million students worldwide have already tried an “Hour of Code”. In the UK and in Australia, Coding has been put into the primary education curriculum. In Hong Kong, CityU Apps Lab (CAL) (<http://appslab.hk>) is a leading University organization offering free workshops to public to learn to code, kicking off their first hour of coding. Over 2,000 hours of coding have been achieved in the previous “Hour of Code HK” workshops, and we, at CityU of Hong Kong, have offered over 10,000 hours of coding lessons to the beneficiaries by running “We Can Code” and “Go Code 2015” with the Sino Group.

In the world’s major economies, students from elementary school to postgraduate level are increasingly getting involved in understanding the fundamentals of computer programs and coding skills. In the UK, a new version of the relevant curriculum has been established a year earlier on 8 July 2013 by GOV.UK, putting a significant emphasis on computing skills. The new curriculum replaces basic word processing skills with

more demanding tasks such as coding and understanding algorithms. Primary school children are proposed to be taught how to write simple programs using computer languages.

In Singapore – Hong Kong’s Asian competitor of diverse areas - a plan is being fermented by its INFOCOMM Development Authority (IDA), which prescribes the progressive introduction of software programming classes into public schools. This would provide students with a unique opportunity to write code in classroom settings employing the teaching and educational resources which are available to other fundamental curriculums. A talk is now being initiated by the nation’s Ministry of Education regarding the necessity of incorporating programming into its national curriculum.

Estonia is beyond all doubt taking the lead in programming skill education by launching a nationwide scheme to teach school kids from the age of seven to nineteen the methodology of writing computer code. It is one of the first countries to have a government that was fully e-enabled. The ProgeTiger initiative was started in January of 2012 by the Estonian government, aiming at bringing programming into classrooms to help raise Estonia’s technical competency. This small country with a population of 1.3 million is the home of Skype and has been attracting sponsoring activities from well-known organizations such as the Mozilla Foundation.

It is of great significance that Hong Kong citizens could grasp the basic principles of mechanisms of the digital devices that play such a large role in modern life and be aware of the fundamentals of coding. It is also important to know that when running the “Hour of Code HK” Campaign, we observe that youth group can achieve the coding tasks in a much shorter time when compared with University students or adults. In this connection, it is identified that there is still a lack of momentum in Hong Kong in the present day to catch up with the world’s best.

We believe that students at their early age are able to understand and acquire computational thinking skill at a faster pace, therefore, in this project we provide them a three-year training from junior, to intermediate, and then to advanced in-class support. Each one of them will consist of 8 to 14 lessons, and each lesson is around 35-45 minutes long. On top of the in-class training, we will also provide them with mentoring support from our University students on a group basis (e.g. one 40-student class will be taken care of by 2 tutors). The University students involved will participate through our established campus internship and other co-curricular experiential learning schemes.

We propose this project on a 3-year basis in order to create a sustainable learning environment for the primary students to keep up their learning attitude. The main role for the CityU team is to provide in-class manpower support and also parent involvement support, and to facilitate effective learning in target schools. CityU Apps Lab, an education community at CityU consisting of more than 600 University students, is able to provide this manpower support throughout this project. It is expected that in 3 years' time, this community can grow up to 1,000 members on campus involving the CityU Alumni network. Students from HK major universities, who are passionate about coding education, will be recruited to join this project.

In order to provide interactions with primary school students, we will provide support to the whole project to create a structured curriculum with the partnering organizations on this project that eventually integrates learning existing subjects such as mathematics and sciences with the computational thinking skills that the students have picked up. This has the potential to galvanize knowledge sharing and learning among the students.

2. ROLES AND RESPONSIBILITIES OF TAS IN CT EDUCATION

In the CoolThink@JC project (<http://www.coolthink.hk/>), 97 teaching assistants (TA) are recruited by the CityU team from over 10 tertiary institutions of Hong Kong in the academic year of 2016/17. The 97 TAs have been trained and assessed based on their performances on a series of tests and teaching practices. They have passed the assessment criteria, and been assigned to serve the 12 pilot primary schools participating in the CoolThink@JC project of Hong Kong.

In general, the main roles of TAs are to assist teachers in answering students' enquiries in class, and the class matters related to CT teaching in the pilot primary schools. Also, they have to support the teacher in creating a joyful and innovative learning environment, and act as a role model in the classroom (e.g. passionate, responsive).

On the other hand, the major responsibilities of TAs are to provide a professional support to teachers in relation to teaching and learning. They have to praise students who have successfully completed the class exercises with creative ideas and behave well, and are able to assist other classmates. Also, they have to inspire students to generate creative ideas by encouraging students to finish their tasks by themselves with appropriate guidance. They have to report any concerns regarding student matters to their supervisors.

3. EVIDENCES OF SELF-DEVELOPMENT OF TAS IN CT EDUCATION

Data are being collected and presented in forms of reflective summary submitted by TAs. The extracted content of the reflective summary are also mapped with corresponding factors of the stress-adaptation-growth process of the Intercultural Transformation Theory (ITT) (Kim and Ruben (1988)). One TA case is presented in this paper (See Appendix). More elaboration of other cases will be presented in the conference.

4. REFERENCES

Kim, Y.Y., & Ruben, B.D. (1988). Intercultural transformation: A systems theory. In Y.Y. Kim., W.B. Gudykunst (Eds.) *Theories in intercultural communication*, Newbury Park, CA: Sage, 299-321.

5. APPENDIX

TA Case

ITT Factors	Extracted Reflective Summary
Stress	<p>Expectation before the co-teaching is simple and direct. To be part of a remarkable project that will enrich my life. Frankly, the work I am doing right now is more or less the same as I expected.</p> <p>The classroom experience is overwhelming. Witnessing our future generation build an astonishing program from scratch is definitely something beyond joy.</p>
Adaptation	<p>The young fellas have many questions regarding the lesson, sometimes I heard funny questions and sometimes, some of their questions even inspire me.</p> <p>Of all the awesome experience I had, one that stand out is a child asked me whether she can write her program in Japanese. I was not sure whether my answer should be yes or no at that time and still not sure at this moment. But I told her as long as she finished her classwork, she can program it in Japanese. Surprisingly, she finished her work within 15 minutes and start writing her Japanese program.</p>
Growth	<p>This experience made me realized the power of curiosity and sometimes it might be the best teacher a child can have.</p>

Author Index

Hal ABELSON	84	Nguyen-thinh LE	39
Gabriella ANTON	17	Chien-sing LEE	45, 150
Connor BAIN	17	Irene A. LEE	60
Gautam BISWAS	11, 28, 34	Matthew LEE	172
Ana C. CALDERON	6	Yunli LEE	133
Mei-ki CHAN	107	Kaitlyn D. LEIDL	116
Peng CHEN	94	Robert Kwok-yiu LI	64, 172
Yanru CHENG	55	Ling LIN	55
Ray CHEUNG	172	Yu-tzu LIN	50
Wai-chong CHIA	133	Liu LIU	164
Belinda CHNG	122	Meei-yen LONG	122
Sue-inn CH'NG	133	Chee-kit LOOI	164
Hyungshin CHOI	81	Samuel Hong-shan LOW	150
Bessie CHONG	139	Debora LUI	67
Tom CRICK	6	Joyce MALYN-SMITH	60
Xinyue DENG	168	Paul MEDLOCK-WALTON	160
Peh-yenc EE	45	Orni MEERBAUM-SALANT	23
Birgit EICKELMANN	103	Claudia MIHM	110, 116
Deborah A. FIELDS	67	Jesús MORENO-LEÓN	154
Chung-kit FUNG	55	Evan W PATTON	2, 145, 168
Divya GOPINATH	145	Niels PINKWART	39
Arjun GUPTA	145	Sarah POLLACK	23
Bruria HABERMAN	23	Gregorio ROBLES	154
Asif HASAN	28	Marcos ROMÁN-GONZÁLEZ	154
Wu-jing HE	107	Lisa L RUAN	2
Michael HORN	17	Peter SEOW	164
Ting-chia HSU	73	Josh SHELDON	77, 84, 145
Chiu-fan HU	50	Amanda A. SULLIVAN	110
Hsin-chung HU	73	Florence R. SULLIVAN	90
Ronghuai HUANG	94	Hillary SWANSON	17
Nicole M HUTCHINS	34	Chee-wei TAN	55, 64, 172
Jane IM	160	Ai-ling THIAN	122
Joseph IPPOLITO	60	Mike TISSENBAUM	2, 77, 84, 145, 160
Yasmin B. KAFAI	67	Catherine TRYFONA	6
P. Kevin KEITH	90	Marina UMASCHI BERS	110, 116
Mi-song KIM	81	Bimlesh WADHWA	164
Siu-cheung KONG	77, 84, 97	An-tsu WANG	50
Ron Chi-wai KWOK	172	Uri WILENSKY	17
Amelie LABUSCH	103	Jane Yat-ching WONG	64
Chun-kiu LAI	55	Jing-wen WONG	45
Ming LAI	77, 84	Kwong-cheong WONG	127
Karen LANG	84	Pam Hau-yung WONG	64
Andrew LAO	84	Wan-chi WONG	107
Natalie LAO	84	Cheng-chih WU	50

Longkai WU	164
Lee-seng YEONG	133
Pei-duo YU	55
Elaine ZHANG	145
Ningyu ZHANG	11, 34



<http://www.eduhk.hk/cte2017/>

Email: cte2017@eduhk.hk

Created & Funded by:



香港賽馬會慈善信託基金
The Hong Kong Jockey Club Charities Trust
同心 同步 同進 RIDING HIGH TOGETHER

Co-created by:



香港教育大學
The Education University
of Hong Kong



Massachusetts
Institute of
Technology



香港城市大學
City University of Hong Kong