Siu-Cheung Kong
Harold Abelson *Editors*

Computational Thinking Education

Springer Open

www.dbooks.org

Computational Thinking Education

Siu-Cheung Kong · Harold Abelson
Editors

Computational Thinking Education



www.dbooks.org

Editors
Siu-Cheung Kong
Department of Mathematics
and Information Technology
The Education University of Hong Kong
Hong Kong, Hong Kong

Harold Abelson
Computer Science and Artificial
Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, USA



ISBN 978-981-13-6527-0 ISBN 978-981-13-6528-7 (eBook)
https://doi.org/10.1007/978-981-13-6528-7

Library of Congress Control Number: 2019931527

© The Editor(s) (if applicable) and The Author(s) 2019. This book is an open access publication.
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Over the past few decades, Computational Thinking (CT) has gained widespread attention and been regarded as one of the essential skills required by those growing up in the digital era. To nurture the next generation to become creative problem-solvers, there is a growing need to implement CT education into the school curriculum. This book is an edited volume with a specific focus on CT education. The chapters were contributed by a group of world-renowned scholars and researchers, who pioneer research on CT education. To enable readers with various interests to advance their knowledge in this fresh yet important field, this book covers sub-themes that will be of interest to academics and educators, school teachers, policymakers and other readers. The sub-themes include CT and tool development, student competency and assessment, CT and programming education in K-12, CT in K-12 STEM education and non-formal learning, teacher and mentor development in K-12 education, and CT in educational policy and implementation. School teachers will be particularly interested in chapters in K-12 and K-12 STEM education; educators and academics will be interested in chapters in CT and tool development, student competency and assessment, and teacher and mentor development; policymakers will be particularly interested in chapters in policy and implementation; and readers, in general, will be interested in chapters in all sub-themes.

This edited volume was funded by CoolThink@JC, a cutting-edge 4-year initiative 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 younger generations to apply digital creativity in their daily lives and prepare them for tackling future challenges in many fields. Considering CT as an indispensable capability to empower students to move beyond mere technology consumption into problem-solving and innovation, with the belief that primary school education is the key in laying the foundations in CT to support future participation in a computing-rich society, CoolThink@JC educated over 16,500 upper primary students in Hong Kong at 32 pilot schools through CT and programming education. The Education University of Hong Kong and

www.dbooks.org

Massachusetts Institute of Technology, being the co-creators of this initiative, have tailored the teaching content and methods for Hong Kong students. An independent evaluator was also appointed to conduct rigorous and evidence-based research to assess students' benefits throughout the Initiative. Over time, the project team aspires to make greater impact by sharing insights and curricular materials with policymakers and educators in Hong Kong and across the territory.

Besides, through intensive train-the-trainer programme, CoolThink@JC developed the teaching capacity of more than 100 local teachers and equipped them with essential CT and programming pedagogies. The trained teachers formed communities of practice to identify best practices and contribute to the continuous improvement of the Initiative. In addition to building teacher capacity in the pilot schools, the Initiative shared CT teaching experience and practice to local educators via organizing seminars and lectures.

CoolThink@JC also recognizes the importance of parents' role in promoting CT education. The City University of Hong Kong, one of the co-creators of this Initiative, reached out to parents and schools through workshops and seminars. With the support of educators and members of the community, the Initiative hopes that all students in Hong Kong and the young people worldwide will be benefited from CT education in near future.

We would like to express our heartfelt gratitude to CoolThink@JC for its tremendous support to the successful publishing of this book in transferring knowledge and implementation of CT education. During the making of this book, we have planned for a second edited volume, which will cover more chapters on policy and implementation in different regions, and conceptual chapters such as different paradigms of CT education, unplugged activities, and parent education in CT. We wish to see scholars and researchers from different parts of the world to contribute their works to our second volume, so that our book series will be well representing CT education around the globe.

We sincerely hope that every reader will enjoy and get inspired by this edited volume.

Tai Po, Hong Kong — Siu-Cheung Kong
Boston, MA, USA — Harold Abelson

Contents

1 Introduction to Computational Thinking Education 1
Siu-Cheung Kong, Harold Abelson and Ming Lai

Part I Computational Thinking and Tool Development

2 Computational Thinking—More Than a Variant of Scientific Inquiry! 13
H. Ulrich Hoppe and Sören Werneburg

3 MIT App Inventor: Objectives, Design, and Development 31
Evan W. Patton, Michael Tissenbaum and Farzeen Harunani

Part II Student Competency and Assessment

4 Measuring Secondary School Students' Competence in Computational Thinking in ICILS 2018—Challenges, Concepts, and Potential Implications for School Systems Around the World 53
Birgit Eickelmann

5 Computational Thinking Processes and Their Congruence with Problem-Solving and Information Processing 65
A. Labusch, B. Eickelmann and M. Vennemann

6 Combining Assessment Tools for a Comprehensive Evaluation of Computational Thinking Interventions 79
Marcos Román-González, Jesús Moreno-León and Gregorio Robles

7 Introducing and Assessing Computational Thinking in the Secondary Science Classroom 99
Hillary Swanson, Gabriella Anton, Connor Bain, Michael Horn and Uri Wilensky

www.dbooks.org

8 Components and Methods of Evaluating Computational Thinking for Fostering Creative Problem-Solvers in Senior Primary School Education 119
Siu-Cheung Kong

Part III Computational Thinking and Programming Education in K-12

9 Learning Composite and Prime Numbers Through Developing an App: An Example of Computational Thinking Development Through Primary Mathematics Learning 145
Siu-Cheung Kong

10 Teaching Computational Thinking Using Mathematics Gamification in Computer Science Game Tournaments 167
Chee Wei Tan, Pei-Duo Yu and Ling Lin

11 Mathematics Learning: Perceptions Toward the Design of a Website Based on a Fun Computational Thinking-Based Knowledge Management Framework 183
Chien-Sing Lee and Pei-Yee Chan

Part IV Computational Thinking in K-12 STEM Education and Non-formal Learning

12 Defining and Assessing Students' Computational Thinking in a Learning by Modeling Environment 203
Ningyu Zhang and Gautam Biswas

13 Roles, Collaboration, and the Development of Computational Thinking in a Robotics Learning Environment 223
P. Kevin Keith, Florence R. Sullivan and Duy Pham

14 Video Games: A Potential Vehicle for Teaching Computational Thinking 247
Sue Inn Ch'ng, Yeh Ching Low, Yun Li Lee, Wai Chong Chia and Lee Seng Yeong

15 Transforming the Quality of Workforce in the Textile and Apparel Industry Through Computational Thinking Education 261
Bessie Chong and Ronald Wong

Part V Teacher and Mentor Development in K-12 Education

16 Teaching Computational Thinking with Electronic Textiles: Modeling Iterative Practices and Supporting Personal Projects in *Exploring Computer Science* 279
Deborah A. Fields, Debora Lui and Yasmin B. Kafai

17 A Study of the Readiness of Implementing Computational Thinking in Compulsory Education in Taiwan 295
Ting-Chia Hsu

18 Self-development Through Service-Oriented Stress-Adaption-Growth (SOSAG) Process in the Engagement of Computational Thinking Co-teaching Education 315
Mani M. Y. Wong, Ron C. W. Kwok, Ray C. C. Cheung, Robert K. Y. Li and Matthew K. O. Lee

Part VI Computational Thinking in Educational Policy and Implementation

19 Educational Policy and Implementation of Computational Thinking and Programming: Case Study of Singapore 345
Peter Seow, Chee-Kit Looi, Meng-Leong How, Bimlesh Wadhwa and Long-Kai Wu

20 Teaching-Learning of Computational Thinking in K-12 Schools in India 363
Sridhar Iyer

www.dbooks.org

Chapter 1
Introduction to Computational Thinking Education



Siu-Cheung Kong, Harold Abelson and Ming Lai

Abstract This chapter provides an overview of this edited volume on Computational Thinking Education (CTE). It starts with a historical review of CTE, beginning from the pioneering ideas of Seymour Papert on promoting the need to think computationally, the seminal work by Jeanette Wing, who argued that computational thinking (CT) should be an essential skill for everyone, and efforts to incorporate CT into K-12 education, such as those made by the National Research Council (Report of a workshop on the scope and nature of computational thinking, National Academies Press, 2010). With this background, the chapter introduces its conceptual framework of CTE and identifies six sub-themes. The section on the 'Computational Thinking and Tool Development' sub-theme includes an in-depth discussion of abstraction, a key concept of CT, and the development of a programming environment to facilitate CT development. 'Student Competency and Assessment' contains chapters that identify the key components, methods and tools for assessing CT. 'Computational Thinking and Programming Education in K-12' focuses on how CT can be taught and cultivated in K-12. 'Computational Thinking in K-12 STEM Education and Non-Formal Learning' discusses the combination of STEM and game activities with CT development. 'Teacher and Mentor Development in K-12 Education' sheds light on the capacity building of teachers and teaching assistants in implementing CT education. 'Computational Thinking in Educational Policy and Implementation' discusses the educational policy related to CT and a 10-year project with thinking skills embedded

S.-C. Kong (✉)
Department of Mathematics and Information Technology,
The Education University of Hong Kong, 10 Lo Ping Road, Tai Po,
N.T. Hong Kong, China
e-mail: sckong@eduhk.hk

H. Abelson
Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: hal@mit.edu

M. Lai
Centre for Learning, Teaching and Technology, The Education University of Hong Kong, Hong Kong, China
e-mail: mlai@eduhk.hk

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_1

in computer studies. Among the issues discussed in these chapters, the key focus of CTE is the importance of learning to think computationally.

Keywords Computation thinking · Computational thinking education · Computer science education · Programming education · STEM education

1.1 Introduction

This collection presents the latest research on and implementations of Computational Thinking Education (CTE). Our book includes contributions from educators and leading researchers around the world aimed at deepening the understanding of CTE theories and pedagogies, with an emphasis on education at the K-12 level. The term 'Computational Thinking Education' emphasizes the role of computing and computational ideas in facilitating learning, a perspective that is the legacy of Seymour Papert (discussed below). The term 'computational thinking' appeared as early as the nineteenth century in reference to the use of quantitative analysis in science, and appeared later regarding the emphasis on reasoning in teaching arithmetic (Childs, 2015). The modern association of the term with computers and education is due to Papert.

Since emerging from the laboratory in the 1950s, computers have become ever-present in modern life. With that growth has come increasing attention from the educational establishment, minuscule at first but building to a groundswell over the past decade. At the tertiary level, the first computer science degree programme began at the University of Cambridge in 1953, while the first programme in the US opened at Purdue University in 1962. There were 89 computer science Bachelor's degrees awarded in the US in 1966, compared to 60,000 such degrees in 2015 (National Academies of Sciences, Engineering, and Medicine, 2018).

Introducing K-12 students to CT has been slower and more sporadic. One seminal line of work stems from the invention of the BASIC programming language by John Kemeny and Thomas Kurtz at Dartmouth College in 1964. Their vision for BASIC (Beginner's All-purpose Symbolic Instruction Code) was that everyone, or at least every Dartmouth undergraduate, would learn to code. BASIC ran on terminals connected to the Dartmouth Time-Sharing system DTSS, which the university made available to Dartmouth undergraduates and to several universities and high schools. BASIC grew explosively with the introduction of the Apple II and other home computers in 1977, and it remains extremely popular with hobbyists and school computer clubs with a focus on learning to program.

A second stream in K-12 CT—and the direct precursor of present-day CTE—originated from the work of Seymour Papert and the 1967 creation of the Logo computing language by Papert, Cynthia Solomon and Wallace Feurzeig (see Solomon, 1986, for a comparative overview of BASIC, Logo and other early research in computing for K-12 learners).

www.dbooks.org

Logo was created at the Cambridge research firm Bolt Beranek and Newman and then moved to the MIT Artificial Intelligence Laboratory with the start of the MIT Logo project in 1969. Papert had worked at the University of Geneva with the renowned Swiss psychologist Jean Piaget, and he brought to Logo Piaget's constructivist theory of learning, which emphasizes that children construct meaning through the interaction between experience and ideas. Papert's extension of constructivism (and wordplay on the term), which he called constructionism, holds that learning happens 'especially felicitously in a context where the learner is engaged in constructing a public entity' (Papert, 1991).

For Papert, constructing entities with the Logo computer language could provide such a felicitous context. The perspective arose that computing could be a powerful intellectual tool for all children, and that technology, as Papert wrote, could become

> …something children themselves will learn to manipulate, to extend, to apply to projects, thereby gaining a greater and more articulate mastery of the world, a sense of the power of applied knowledge and a self-confidently realistic image of themselves as intellectual agents. (Papert, 1971)

The full expression of this idea and its characterisation as 'computational thinking' first appeared in Papert's book Mindstorms (Papert, 1980), although Papert also referred to the same idea as 'procedural thinking'. He wrote:

> In this book I have clearly been arguing that procedural thinking is a powerful intellectual tool and even suggested analogizing oneself to a computer as a strategy or doing it. … The cultural assimilation of the computer presence will give rise to computer literacy. This phrase is often taken as meaning knowing how to program, or knowing about the varied uses made of computer. But true computer literacy is not just knowing how to make use of computers and computational ideas. It is knowing when it is appropriate to do so. (Papert, 1980, p. 155)

Even here, in the first articulation of the idea, there was a concern to clarify the distinction between computational (or procedural) thinking and the knowledge of how to program or to use computational tools. This concern for clarification has persisted through the growth of the CT movement and is present in several papers in the present volume.

The appearance of the personal computer in the late 1970s produced an outburst of optimism about computing's potential to play a major role in K-12 education. Apple II BASIC appeared in 1978 and Apple Pascal in 1979. MIT worked with Texas Instruments to create a Logo implementation for the TI 99/4 home computer, piloting it in a 450-student Dallas elementary school in 1980 and later in several New York City public schools. Versions of Logo for the Apple II appeared in 1982 (Abelson, 1982a, b).

While the impact of these implementations on the emerging home hobbyist computer community was significant, there was little take-up in K-12 education. School adoption was meagre, with little adherence to the vision that computation could be a powerful learning framework for everyone, not only for students working towards careers involving computer programming. As several leaders of the school computing movement observed in 2003: 'while the literature points to the potential for

impact, the reality is sobering: to a first order approximation, the impact of computing technology over the past 25 years on primary and secondary education has been essentially zero' (Norris, Sullivan, Poirot, & Soloway, 2003).

However, the adoption of computers in K-12 began to increase with the tremendous increase in the impact of information technology in society and with the emergence of computing as a continuous presence in daily life. An important catalyst for change was Jeanette Wing's (2006) seminal essay 'Computational Thinking' (Wing, 2006). Wing reintroduced the term 'computational thinking' together with the notion in Papert's tradition that CT was not just programming and that it should be a fundamental skill for everyone. The appearance of Wing's essay, contemporaneous with the start of an enormous upwelling of the computing industry, led to a surge of interest in CT and computing in K-12 education that surprised many long-time observers of computing education. Even Wing herself observed:

> "Not in my lifetime." That's what I said when I was asked whether we would ever see computer science taught in K-12. It was 2009, and I was addressing a gathering of attendees to a workshop on computational thinking convened by the National Academies. I'm happy to say that I was wrong. (Wing, 2016)

Yet even with this burgeoning interest, there remains a widespread lack of clarity about what exactly CT is, and the struggle continues to articulate its fundamentals. The report of the 2009 National Academies workshop that Wing mentions above expressed the motivation behind it:

> Various efforts have been made to introduce K-12 students to the most basic and essential computational concepts, and college curricula have tried to provide students a basis for lifelong learning of increasingly new and advanced computational concepts and technologies. At both ends of this spectrum, however, most efforts have not focused on fundamental concepts.
>
> One common approach to incorporating computation into the K-12 curriculum is to emphasize computer literacy, which generally involves using tools to create newsletters, documents, Web pages, multimedia presentations, or budgets. A second common approach is to emphasize computer programming by teaching students to program in particular programming languages such as Java or C++. A third common approach focuses on programming applications such as games, robots, and simulations.
>
> But in the view of many computer scientists, these three major approaches—although useful and arguably important—should not be confused with learning to think computationally. (National Research Council, 2010)

It is sobering that the concern to distinguish CT from programming and from the use of computer tools is the same as that expressed by Papert in Mindstorms at the genesis of the CT movement 30 years previous.

Many of the papers in this volume grapple with this same concern, and readers will find several discussions of what 'computation thinking' means in the papers that follow. Much of the 2009 NRC workshop was devoted to a discussion of this same definitional question. The workshop participants, all experts in the field, did not come to any clear agreement, nor do the authors in this volume. Yet as in the

www.dbooks.org

NRC report, they agree that the cluster of ideas around CT is important in a world being increasingly transformed by information technology.

A second theme in the papers in this volume is the need to confront issues of educational computing at scale. One result of the increasing attention to CT is that jurisdictions are beginning to mandate computing education in K-12. Estonia, Australia, New Zealand, Taiwan, the United Kingdom and the US states of Virginia, Arkansas and Indiana have already taken this step, and other nations are formulating strategies to do so. This has led to serious issues of equipment availability and teacher education; several of the papers below present overviews of plans enacted or in progress. Key among the issues here is assessment, as the increasing mandates for computer learning are requiring increasing accountability from citizens and policy makers.

1.2 Conceptual Framework and Chapters in This Book

The chapters of the book were selected based on our conceptual framework of computational thinking education with six sub-themes, as illustrated in Fig. 1.1. At the top of Fig. 1.1 is 'Computational Thinking and Tool Development', the basic building block of CTE, which involves issues of the definition of CT and the design of the programming environment for facilitating CT. Students' CT development can occur in K-12 and can be combined with STEM education and non-formal learning, as captured by the sub-themes of 'Computational Thinking and Programming Education in K-12' and 'Computational Thinking in K-12 STEM Education and Non-formal Learning', respectively. To evaluate the effectiveness of students' CT development, we need to consider assessment issues, which include the articulation of the competencies involved, and the latest methods of assessing CT, as reflected in the sub-theme of 'Student Competency and Assessment'. Teacher and mentor development is a key factor to support the implementation of CTE, as captured by the sub-theme of 'Teacher and Mentor Development in K-12 Education'. From a broader perspective, policy matters can also play a supportive role in CTE, as illustrated in the sub-theme of 'Computational Thinking in Educational Policy and Implementation'. The chapters in this book were chosen according to these six sub-themes.

1.2.1 Sub-theme 1: Computational Thinking and Tool Development

The sub-theme of 'Computational Thinking and Tool Development' includes two chapters. Hoppe and Werneburg consider the abstraction involved in CT and in scientific inquiry, arguing that the former has more 'representational flexibility', and that therefore an important goal in CT education is to identify the specificity of CT

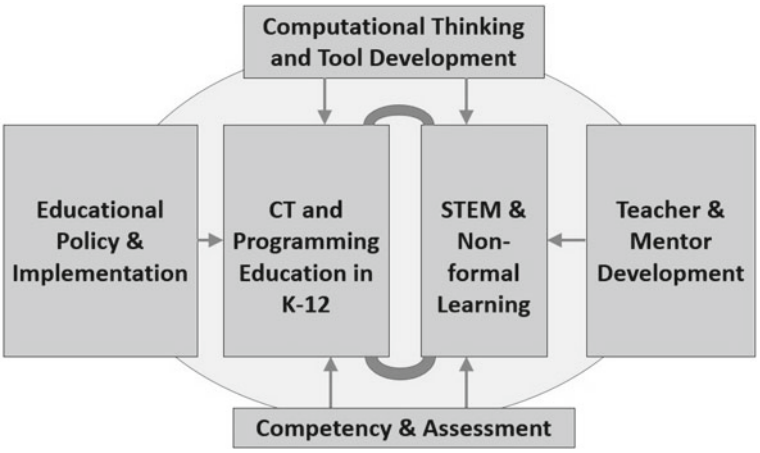


Fig. 1.1 Conceptual framework of computational thinking education

arising from its abstraction. The available abstractions, from either data representation or processing, form a repertoire of possible choices to generate computational artefacts. The programming environment supports different models of computation for users to choose from (e.g. visual programming interface), thus enabling programming flexibility. Furthermore, in terms of abstract operational mechanisms and data structure, computational media in inquiry learning contexts are of finite representational flexibility. In the other chapter, Patton, Tissenbaum and Harunani document the development of an online platform for facilitating CT, the App Inventor, at the Massachusetts Institute of Technology (MIT). They identify the logic and goals underlying the design of the platform and document how it can be used for educational purposes, focusing on empowerment through programming. These two chapters indicate the importance of the use of abstraction and a well-designed programming environment to facilitate students in learning to think computationally.

1.2.2 Sub-theme 2: Student Competency and Assessment

Among the key issues that require further exploration is how to measure students' CT ability and examine the effects of the teaching of CT, especially in terms of major assessment approaches and research methodologies. The sub-theme of 'Student Competency and Assessment' includes five chapters on related issues. Eickelmann presents a large-scale international comparative study on CT, with problem conceptualisation and solution operationalisation as the two main strands of constructs of students' achievements in CT to be assessed. The identification of these constructs echoes Papert's idea that to think computationally involves 'not just knowing how to make use of computers and computational ideas… [but] knowing when

www.dbooks.org

it is appropriate to do so' (Papert, 1980, p. 155). Labusch, Eickelmann and Vennemann align CT with problem solving and information processing, reporting on the design of a cross-national study of students' processes of CT with a focus on a cognitive approach. Roman-Gonzalez, Moreno-Leon and Robles review and classify tools for assessing CT, with categories of CT diagnostic, CT summative, CT formative-iterative, CT data-mining, CT skill-transfer, CT perceptions-attitude and CT vocabulary assessment. They present the findings of two convergent validity studies conducted using a variety of tools, and they put forward a comprehensive framework that involves the chronological uses of various tools to evaluate CT. Swanson, Anton, Bain, Horn and Wilensky evaluate the effectiveness of a computational biology curriculum among ninth-grade students and find that their modelling and simulation practices, which are an important strand of CT practices, improved significantly. Kong conducts a comprehensive review of the literature and identifies the key components and methods for evaluating students' CT development based on the differentiation of CT concepts, practices and perspectives proposed by Brennan and Resnick (2012). The consideration of computational identity and programming empowerment as important components of CT perspectives particularly merits research attention.

1.2.3 Sub-theme 3: Computational Thinking and Programming Education in K-12

There are three chapters under the sub-theme of 'Computational Thinking and Programming Education in K-12'. Kong, using an example of learning prime and composite numbers through the development of an app, illustrates how learners' CT can be developed in their learning of primary mathematics, highlighting the pedagogies that can be used. Tan, Yu and Lin, who regard CT as the cultivation of logical thinking and problem-solving skills, present a study on how CT can be taught using mathematical gamification. They develop mobile games to help students develop problem-solving skills and gain mathematical insights by solving linear equations. The difficulty at each level is calibrated based on the users' performance to ensure a reasonable growing curve and prevent users from becoming frustrated at early levels. They argue that gamification can be considered an effective educational approach to gaining arithmetic proficiency and computational skills. Lee and Chan document the design of an educational website guided by a fun-based CT framework integrated with a knowledge management approach, and they discuss how the different components in this website can facilitate students' mathematics learning. These chapters illustrate how CT and programming education can be implemented in K-12 classrooms.

1.2.4 Sub-theme 4: Computational Thinking in K-12 STEM Education and Non-formal Learning

The sub-theme of 'Computational Thinking in K-12 STEM Education and Non-formal Learning' contains four chapters. Zhang and Biswas extend a framework to evaluate students' synergistic learning of CT skills and science content. With the use of a computer-based learning environment, the authors illustrate students' learning gains in CT and science concepts, and they indicate that by focusing on the synergies between STEM and CT, the difficulties that students might encounter in their simulation tasks in the learning of science content could be overcome. Keith, Sullivan and Pham use a microgenetic case study approach to investigate the roles played by two groups of girls in educational robotics activities, finding that the emergence of distinct roles is related to the time of collaboration and the time of solo work, and that it thus affects students' engagement in CT. The authors also find that in the more collaborative group, students shared their major roles, while in the less collaborative group, the roles were adopted much earlier and adhered to thereafter, leaving some group members with no chance to engage in algorithmic thinking. Ch'ng, Low, Lee, Chia and Yeong examine the correlation between video gaming experience and individual CT skills, finding a significant correlation between the former and a specific category of CT skills, abstraction and problem decomposition. This finding can help address the concern that computer games might negatively influence students' introductory programming course performances. Non-formal learning happens outside of the formal education system, and training in a company can be seen as a perfect setting for equipping targeted people with specific knowledge, such as CT. Chong and Wong, who believe that employees in the Industry 4.0 era need to be equipped with computational and logical thinking, document the experience of a textile and apparel company in which employees were empowered to solve problems and use creative ideas to improve their daily work by developing mobile apps. These works are dedicated to providing more teaching and learning practices on CT and stimulate further questions regarding how to evaluate and validate non-formal learning outcomes.

1.2.5 Sub-theme 5: Teacher and Mentor Development in K-12 Education

Three chapters are included in the sub-theme of 'Teacher and Mentor Development in K-12 Education'. Sanford and Naidu (2016) argue that 'computational thinking does not come naturally and requires training and guidance' and thus that qualified teachers for future CT education are urgently needed. Fields, Lui and Kafai identify the teaching practices that can support students' iterative design in CT, including teachers' modelling of their own CT processes and mistakes and of students' mistakes in front of the whole class. They highlight an iterative design process as a crucial

www.dbooks.org

aspect of CT and the importance of revision and working through mistakes. Hsu considers the readiness of CT education from the perspective of school leaders, rating computer hardware readiness and leadership support readiness most favourably and instructor readiness and instructional resources readiness lower. Wong, Kwok, Cheung, Li and Lee discuss the supportive roles played by teaching assistants in CT classes and analyse their self-development through the process of service-oriented stress-adaption-growth.

1.2.6 Sub-theme 6: Computational Thinking in Educational Policy and Implementation

The sub-theme of 'Computational Thinking in Educational Policy and Implementation' contains two chapters. Seow, Looi, Wadhwa and Wu review the educational policy of CT in Singapore, which uses a pragmatic approach that depends on an eco-system with a focus on cultivating students' interests and allowing schools to opt in rather than making CT education compulsory. Singapore's policy offers good opportunities to educational stakeholders with interests in coding. Learning activities correspond to age and cognitive discrepancies, aiming for learning through playing in pre-school, interest cultivation in primary school and computing application in secondary school. In the final chapter, Sridhar describes a 10-year project called Computer Masti in India that embeds thinking skills such as CT into computer studies. The project involves the development of a curriculum and textbooks and the large-scale training of teachers to implement the curriculum. These chapters indicate the importance of good policies and good planning in facilitating everyone in learning to think computationally.

References

Abelson, H. (1982a). *Logo for the Apple II*. Byte Books.

Abelson, H. (1982b). *Apple Logo*. Byte Books.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *2012 Annual Meeting of the American Educational Research Association (AERA'12)*, Canada.

Childs, K. (2015). Computational thinking—The origins (part 1). *Computing & Digital Making*. Blog post, https://primaryicttech.wordpress.com/2015/12/29/the-origins-of-computational-thinking-part-1/.

National Academies of Sciences, Engineering, and Medicine (2018). *Assessing and responding to the growth of computer science undergraduate enrollments*. Washington, DC: The National Academies Press. https://doi.org/10.17226/24926.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking*. National Academies Press.

Norris, C., Sullivan, T., Poirot, J., & Soloway, E. (2003). No access, no use, no impact: Snapshot surveys of educational technology in K-12. *Journal of Research on Technology in Education, 36*(1), 15–27.

Papert, S. (1971). *Teaching children thinking*. MIT Artificial Intelligence Laboratory Memo no. 2247, Logo Memo no. 2. http://hdl.handle.net/1721.1/5835.

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

Papert, S. (1991). Situating constructionism. In S. Papert & I. Harel (Eds.), *Constructionism* (pp. 1–11). Norwood, NJ: Ablex.

Sanford, J. F., & Naidu, J. T. (2016). Computational thinking concepts for grade school. *Contemporary Issues in Education Research, 9*(1), 23–32.

Solomon, C. (1986). *Computer environments for children: A reflection on theories of learning and education*. Cambridge, MA: MIT press.

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

Wing, J. M. (2016). Computational thinking, 10 years later. *Microsoft Research Blog*. https://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later/.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Part I
Computational Thinking and Tool Development

Chapter 2
Computational Thinking—More Than a Variant of Scientific Inquiry!



H. Ulrich Hoppe and Sören Werneburg

Abstract The essence of Computational Thinking (CT) lies in the creation of "logical artifacts" that externalize and reify human ideas in a form that can be interpreted and "run" on computers. Various approaches to scientific inquiry (learning) also make use of models that are construed as logical artifacts, but here the main focus is on the correspondence of such models with natural phenomena that exist prior to these models. To pinpoint the different perspectives on CT, we have analyzed the terminology of articles from different backgrounds and periods. This survey is followed by a discussion of aspects that are specifically relevant to a computer science perspective. Abstraction in terms of data and process structures is a core feature in this context. As compared to a "free choice" of computational abstractions based on expressive and powerful formal languages, models used in scientific inquiry learning typically have limited "representational flexibility" within the boundaries of a predetermined computational approach. For the progress of CT and CT education, it is important to underline the nature of logical artifacts as the primary concern. As an example from our own work, we elaborate on "reactive rule-based programming" as an entry point that enables learners to start with situational specifications of action that can be further expanded into more standard block-based iterative programs and thus allows for a transition between different computational approaches. As an outlook beyond current practice, we finally envisage the potential of meta-level programming and program analysis techniques as a computational counterpart of metacognitive strategies.

Keywords Computational thinking · Computational abstraction · Computer science education

H. Ulrich Hoppe (✉) · S. Werneburg
Department of Computer Science and Applied Cognitive Science,
University of Duisburg-Essen, Lotharstraße 63, 47057 Duisburg, Germany
e-mail: hoppe@collide.info

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_2

www.dbooks.org

2.1 Introduction

2.1.1 Origins of the Current Debate

Although Papert (1996) had already used the term "Computational Thinking" (CT) ten years earlier, the current discussion of CT can be traced back to Wing (2006). Wing characterized CT by stating that it "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science". She emphasized that CT is not about thinking like a computer but about how humans solve problems in a way that can be operationalized with and on computers.

Science as well as humanities are influenced by CT since "computational concepts provide a new language for describing hypotheses and theories" in these fields of interest (Bundy, 2007). In addition, Barr and Stevenson (2011) formulated CT-related challenges for K-12 education in computer science, math, science, language and social studies.

Wing (2008) refined her first statement pointing out that the essence and key of CT is abstraction in a way that is more complex than in mathematics. If someone builds a computational model and wants to include all properties seen in the real environment, s/he cannot enjoy "the clean, elegant or easy definable algebraic properties of mathematical abstraction".

2.1.2 Computational Thinking for K-12

To bring CT to K-12, the International Society for Technology in Education and the Computer Science Teacher Association (ISTE and CSTA 2011) presented a definition of CT for K-12 Education

Computational Thinking is a problem-solving process that includes (*but is not limited to*) *the following characteristics*:

- *Formulating problems in a way that enables us to use a computer and other tools to help solve them*
- *Logically organizing and analyzing data*
- *Representing data through abstractions, such as models and simulations*
- *Automating solutions through algorithmic thinking (a series of ordered steps)*
- *Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources*
- *Generalizing and transferring this problem-solving process to a wide variety of problems.*

The first three bullet points highlight the importance of computational modeling as part of the problem-solving process. General approaches to computational modeling are typically facilitated by programming languages. However, we will see that the

creation of computational or formal models related to given problems can also make use of other abstract tools such as automata or grammars.

As part of the same proposal, it is claimed that CT problem-solving should be connected to realistic applications and challenges in science and humanities. This would link the learning experience and the ensuing creations to relevant real-world problems. However, realism in this sense can also lead to a "complexity overkill" that obstructs the pure discovery of important basic building blocks (the nature of which will be elaborated later). On the other hand, if models are simplified too much they lose fidelity and ultimately credibility. Many computer science concepts do not require an application to real complex environment in their basic version. Interactive game environments, for example, do not necessarily require an accurate modeling of physics but they can still promote the learning of CT concepts.

2.1.3 Model Progression: The Use-Modify-Create Scheme

CT activities typically result in the creation of logical artifacts that can be run, tested against the original intentions, and can be refined accordingly. The creation of an initial executable artifact can be very challenging for learners. Lee et al. (2011) presented a model with a low threshold for novices and promoted it as the "Use-Modify-Create Progression" (see Fig. 2.1).

In the initial phase of this process, learners *use* or *analyze* a predefined and executable (programming) artifact that typically contains a new construct or new type abstraction. In this phase, the learners will *modify* the artifact, which causes observable changes in the output. At the beginning, these modifications are often confined to *varying* predefined parameters. The consecutive *create* phase is essential for the appropriation of new CT skills, giving learners the opportunity to be creative and express themselves through programming. Iteratively, the students create new computational artifacts, execute these, and evaluate the ensuing outputs. This progression

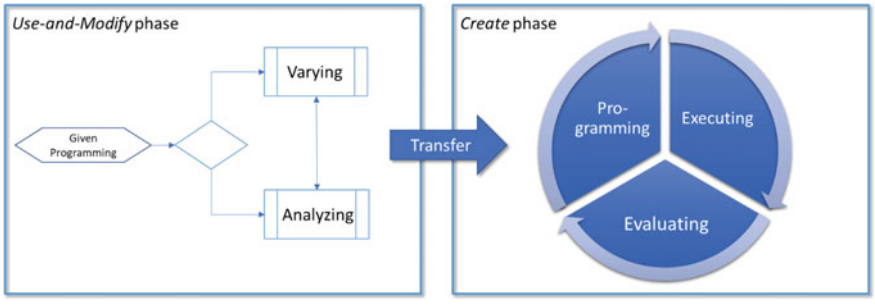


Fig. 2.1 Use-modify-create progression. Adapted from Lee et al. (2011)

www.dbooks.org

incorporates a smooth transition from reusing a predefined new type of artifact to learner generated creative construction.

The specification given by Lee et al. does not provide a clear distinction between the three phases of the cycle. Therefore, it is necessary to refine and rethink this part of the pedagogical model. It is certainly important for novices to have an "interactive starting point" that does not depend on their own original creation and a clear transfer between the *Use-Modify* phase and the *Create* phase is necessary. However, the creative-constructive activities of modeling, specifying and representing problems are also very important ingredients of computational thinking processes.

2.1.4 The CT Terminology

There are few meta-level studies on the existing CT literature. For example, Kalelioglu, Gülbahar, and Kukul (2016) unveiled that CT is very frequently related with "problem-solving", whereas Lockwood and Mooney (2017) point out that the definition of CT as well as the embedment of CT in curricula are still emerging, which make a final characterization difficult.

To characterize the discourse on CT considering different phases and perspectives, we have conducted a text analysis on selected corpora of scientific articles. We used text mining tools to extract important concepts connected to CT. First, we defined three and selected categories of articles[1]:

Articles about CT mainly related to …

(i) … computer science education before 2006 (corpus $n = 14$),
(ii) … computer science education 2006 and later (corpus $n = 33$),
(iii) … inquiry-based learning in science (corpus $n = 16$).

The distinction between the first two categories and corpora is based on a separation with respect to the Wing (2006) article. The third category is also limited to articles following up on Wing (2006) but with a focus in K-12 education in science classes and humanities classes. We will show how the concepts related to CT will differ.

To achieve this, we extracted terms from the three corpora based on the standard *tf-idf*-measure. Since the corpora differ in number of documents and volume, the extraction has been normalized to yield about 100 terms per category. The reduction of terms is based on sparsity of the document-term matrices. In the first step, the full paper texts (abstracts included) were loaded. After removing unnecessary symbols and stop words, the texts have been lemmatized using a dictionary of base terms. Relevant composite terms such as "computational thinking" (comp_think), "computer science", "problem solving" (prob_solv) or "computational model" (comp_model) are considered as possible target terms.

[1] The corpora of the three categories can be found in Appendix A on www.collide.info/textminingCT.

The sparsity criterion leads to a selection of terms that appear in at least p documents as a threshold t. The threshold has been adjusted per category to achieve the desired selection of 100 terms approximately. Table 2.1 shows the attributes of each matrix before and after removing of the sparse terms.

The frequency-based word clouds shown in Fig. 2.2 indicate differences in the corresponding portions of discourse for the three categories. Terms such as "learn", "model", "system", and "problem" appear in each category but with different weights. The earlier work in computer science education (first category) frequently refers to "mathematics" and "systems". Category (ii) contains an explicit notion of "computer science", whereas "science" in general and "simulation" are characteristic for category (iii). Category (ii) is also more concerned with special concepts related to computer science, such as "algorithm", "abstraction", and "computing". A special focus in this category appears to be on "games".

In Fig. 2.3, a comparison between selected frequent terms between all three categories is presented. Here, each bar represents the tf-idf-measure normalized to the selected term base.

Figure 2.3 corroborates the impressions gained from the word cloud diagrams. The profiles of categories (ii) and (iii) appear to be quite similar among each other as compared to category (i). "Algorithm" and "abstraction" are more important concepts

Table 2.1 Attributes of the document term matrix before and after removing of sparse terms

	Category (i)	Category (ii)	Category (iii)
Documents	14	33	16
Terms	4329	4411	2143
Average document length	1619.57	834.21	593.81
Sparsity (%)	84	91	88
Terms (with threshold)	88(t = 35%)	98(t = 55%)	104(t = 65%)
Sparsity (%)	21	40	55



Fig. 2.2 Word Clouds of the terms with the highest tf-idf value of category (i), (ii), and (iii)

www.dbooks.org

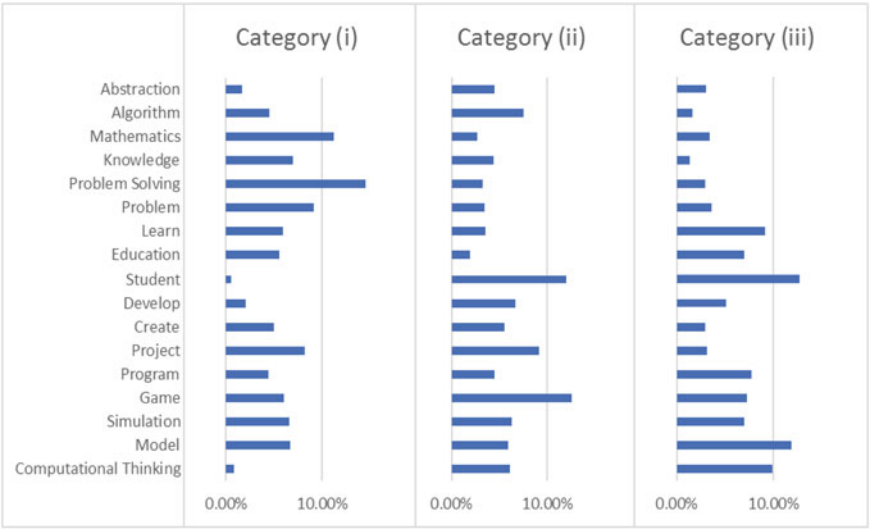


Fig. 2.3 Normalized frequencies of selected terms

in category (ii), whereas "education" is more prominent in category (iii). The latter observation corresponds to the disciplinary background of the contributions which is more often education and less computer science for category (iii) as compared to (ii). However, CT as a theme is even more frequently addressed in category (iii). Several papers in category (ii) use games as examples to illustrate general principles of CT, which explains the prominent role of this concept. For category (i), in addition to the focus on mathematics (cf. "the (Logo) turtle meets Euclid"—Papert 1996), this earlier discourse is much more centered on "knowledge" and "problem (solving[2])" as epistemological categories. In the later discourse for both (ii) and (iii), the epistemological perspective is substituted by instructional and cognitive frameworks.

2.2 Basic Concepts and Building Blocks

2.2.1 "Computational Models" and "Models of Computation"

In his definition of CT, the computer science pioneer Aho (2012) characterizes Computational Thinking as "the thought processes involved in formulating problems so their solution can be represented as computational steps and algorithms". Before

[2]Problem-solving itself is only present in some documents, so the sparsity threshold led to removing this term from the word cloud, even though the td-idf value was high.

starting the problem formulation and construction of a solution, it is necessary to specify an appropriate "model of computation" as a basis. This notion, sometimes also named "computational model" is very specific and important to computer science and very different from the notion of "computational model" that denotes the resulting computational artifact of a constructive CT activity. Aho's "models of computation" are the background and conceptual platforms of computational artifacts; they correspond to "abstract engines" of the type of Turing Machines, Finite State Machines or von Neumann architectures but also possibly Petri Nets, grammars or rewrite system. These models define the basic mechanisms of interpretation of computational artifacts (i.e., "computational models" of the other type).

The grounding of computation on such abstract engines, and accordingly the choice of such foundations, is not very much in the focus of the current more educationally inspired CT discourse, and it might seem that these choices have little to do with CT practice. However, there are relevant examples that explicitly address the choice of basic "models of computation".

The "Kara" microworld (Hartmann, Nievergelt, & Reichert, 2001) uses a programmable ladybug to introduce concepts of computer science and programming. It comes with different versions based on different abstract engines. The original version was based on Finite State Machines, but later versions based on programming languages (JavaKara, RubyKara) were added. That is, the microworld of Kara allows for solving the same of similar problems based on different "models of computation" in the sense of Aho. The FSM version allows for linking the Kara experience to automata theory in general, whereas the Java and Ruby versions may be used as introductions to programming.

Kafura and Tatar (2011) report on a Computational Thinking course for computer science students in which different abstract formalisms or engines (including Petri nets, BNF grammars, lambda expressions) with corresponding tools were employed to construct computational models in response to various problems. This example shows that relying on abstract models of computation can be an alternative to using programming in the construction of computational artifacts.

Curzon and McOwan (2016) describe a specific relation between computational modeling and algorithmic thinking: The algorithm simulates the transformation of an idea contextualized in a virtual or real world into a runnable computational representation, possibly as part of a game environment. However, computational modeling also refers the specification and handling of data and specification process structure. The choice of such structures is not only induced by the "external" problem but is also governed by a partially independent "logic of computation".

2.2.2 The Notion of "Abstraction"

A central element of CT is abstraction in the specific sense it has in the context of computational principles. Wing (2008) underlines the importance of abstraction for computational thinking

www.dbooks.org

> The essence of computational thinking is abstraction… First, our abstractions do not necessarily enjoy the clean, elegant or easily definable algebraic properties of mathematical abstractions, such as real numbers or sets, of the physical world… In working with layers of abstraction, we necessarily keep in mind the relationship between each pair of layers, be it defined via an abstraction function, a simulation relation, a transformation or a more general kind of mapping. We use these mappings in showing the observable equivalence between an abstract state machine and one of its possible refinements, in proving the correctness of an implementation with respect to a specification and in compiling a program written in a high-level language to more efficient machine code. And so the nuts and bolts in computational thinking are defining abstractions, working with multiple layers of abstraction and understanding the relationships among the different layers. Abstractions are the 'mental' tools of computing.

It is necessary to distinguish between the general understanding of abstraction and the specific nature of computational abstraction, or better "abstractions" (plural). The general idea of abstraction as a conceptual generalization and omission of coincidental details can of course also be found in the computational domain. Abstraction in this general sense if a process that supports the building of categories and thus the structuring of a domain. However, in computer science, abstractions can also be representational or operational constructs (i.e., mental tools in the words of Wing), which is not clear from a common sense point of view. Abstractions of this type include data structures, different concepts of variables depending on underlying models of computation or programming paradigms, procedural, and functional abstraction, recursion, as well lambda expressions in combination with higher order functions.

Hu (2011) elaborates on the relation between CT and mathematical thinking, arguing that, behind a quite different surface, CT skills are indeed quite similar to mathematical thinking. It is true that computational artifacts are governed by mathematical (especially logical) principles. Yet, where mathematical models focus on general structural properties, computational artifacts are operational in the sense that they have inherent behavior. This is reflected in the constructive nature of computational abstractions. CT as a creative activity involves choices and combinations of computational abstractions as mental tools. This calls for "representational flexibility", i.e., the provision of making choices between different representations and abstractions, as part of any kind of teaching and learning targeting CT.

2.2.3 Languages, Representations, and Microworlds

The building of computational artifacts requires a "medium" of representation that affords certain computational mechanisms in way susceptible to human imagination and creation. This idea is reflected in diSessa's notion of computational media as basis for "computational literacies" (diSessa, 2000). The notion of literacy sets the focus on the aspect of familiarity and possibly mastery with respect to the specific medium. The computational medium would include a "model of computation" in Aho's sense and would, on this basis, provide more or less easy access to different types of

abstractions. It is a natural choice to use a programming language for this purpose. For programming languages, it is well known that they resonate with computational abstractions (as constructs) in specific ways. For example, the concept of a variable as a storage or memory location is typical for imperative languages. It implies that variables can have mutable values, which is different from the concept of variables in pure functional or logical languages. However, as we have exemplified by referring to the Kara example (Hartmann et al., 2001) and to the CT course designed by Kafura and Tatar (2011), programming languages are not the only choice as a computational medium. Theory-inspired formal approaches provide "abstract engines" that can be used to build computational models, whereas "unplugged" approaches rely on physical or real-world models (e.g., teaching and explaining recursion through role-play).

Computational media for creative and constructive learning are often combined with concrete application domains (corresponding also to learning domains) for which the medium and its representational primitives are particularly designed. This corresponds to the notion of a "microworld", which was already one of the building blocks of the Logo approach. The educational affordances and usage patterns that originate from microworlds are immense and have been widely discussed from an educational technology point of view, see, e.g., (Rieber, 1996). The nature of a microworld as a computational construct and a tool of thought is nicely captured by diSessa (2000)

> A microworld is a type of computational document aimed at embedding important ideas in a form that students can readily explore. The best microworlds have an easy-to-understand set of operations that students can use to engage tasks of value to them, and in doing so, they come to understanding powerful underlying principles. You might come to understand ecology, for example, by building your own little creatures that compete with and are dependent on each other.

From a computer science perspective, microworlds in the sense described by diSessa can be conceived as *domain-specific languages* designed to facilitate constructive learning in certain domains. Compare the general characterization given by van Deursen, Klint, and Visser (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.*" This suggests that the principles of designing and implementing DSLs should be taken into account when we develop microworlds as computational media for learning.

2.2.4 CT from the Perspective of Inquiry Learning in Science

Inspired by the process of scientific discovery, Inquiry Learning (IL) is defined as "an approach to learning that involves a process of exploring the natural or material world, and that leads to asking questions, making discoveries, and rigorously testing

www.dbooks.org

those discoveries in the search for new understanding" (Ash, 2003). Inquiry learning leads students through various phases (Pedaste et al., 2015), typically starting with an orientation phase followed by a conceptualization with idea generation and the development of hypotheses. During the investigation phase, students engage in experimentation, taking measurements to test their hypotheses. Finally, the results are evaluated and discussed, which may lead to reformulation of hypotheses. This is usually understood as a spiral or cyclic process that allows for repeated revisions and refinements (Minner, Levy, & Century, 2010).

CT and IL practices overlap in the underlying cyclic phase models as learning process structures, as exemplified by the *use-modify-create progression* and various IL-related process models. However, the role of computational artifacts differs between CT and IL: In IL, the artifact or model serves as a proxy for a scientific target scenario (e.g., an ecosystem) and the main point is what the model can tell us about the original. In CT, the computational artifact is of primary interest per se, including the way it is built and its inherent principles. If a learner evaluates the computational artifact (or model) at hand in an IL context, this will typically involve a variation of parameters and possibly redefinition of behaviors. In a CT context, this encompasses the reflection and redesign of the underlying computational model and representation as well.

Sengupta, Kinnebrew, Basu, Biswas, and Clark (2013) elaborate in detail on relationships between IL and CT concepts using the CTSiM environment ("Computational Thinking in Simulation and Modeling") as a concrete point of reference. The environment as such is based on visual agent-based programming. Their approach is that, from educator's perspective, students learn best when they use design-based learning environments which is also an approach of "science in practice" that involves "engag(ing) students in the process of developing the computational representational practices " (Sengupta, Kinnebrew, Basu, Biswas, & Clark 2013). In this article, certain computational concepts and principles are related to aspects of the IL environment. Abstraction is discussed in relation to the classical definition given by the philosopher Locke as the process in which "ideas taken from particular beings become general representatives of all of the same kind" (Locke, 1700). As we have seen above, this is not sufficient for the understanding of abstraction in a computer science sense. The specified relationships between computer science concepts and structural and operational aspects found in the CTSiM environment are rich and relevant. Yet, we need to distinguish between representational choices made in the design and implementation of the environment and choices that are "handed over" to the learners operating in the environment using the visual agent-based programming interface. These choices are indeed limited.

Perkins and Simmons (1988) showed that novice misconceptions in mathematics, science, and programming exhibit similar patterns in that conceptual difficulties in each of these domains have both domain-specific roots (e.g., challenging concepts) and domain general roots (e.g., difficulties pertaining to conducting inquiry, problem-solving, and epistemological knowledge).

In this perspective, the development of scientific expertise is inseparably intertwined with the development of epistemic and representational practices, e.g., (Giere, 1988; Nersessian, 1992; Lehrer & Schauble, 2006; NRC, 2008). Basu et al. (2012) describe how students use computational primitives to "generate their computational models". The students can see "how their agents operate in the microworld simulation, thus making explicit the emergence of aggregate system behavior" (Basu et al., 2012). Their "computational model" is focused on the description of the system dynamics using computational tools. There is an overlap between inquiry learning, system thinking, and computational thinking. Although the overlap between the areas seems to be evident, CT involves competencies and skills that can be clearly distinguished from the other fields.

The discussion above suggests that although Wing (2006) defined CT as a "thought process", computational thinking becomes evident only in particular forms of epistemic and representational practice that involve the generation and use of external representations (i.e., representations that are external to the mind) by the learners (Sengupta et al., 2013).

2.2.5 Interim Summary

Regarding the structuring of learning processes and the enrichment of such processes with computational media, inquiry learning in science and CT education are quite closely related. However, a discourse that is primarily driven by pedagogical inspirations and interest tends to neglect the importance of genuine computer science concepts and their role in shaping CT. The essence of CT lies in the creation of "logical artifacts" that externalize and reify human ideas in a form that can be interpreted and "run" on computers. The understanding of the principles underlying and constituting such logical artifacts, including "models of computation" in the sense of Aho as well as specific "abstractions as constructs", are of central importance for CT. In contrast, in general scientific inquiry learning, computational models are instrumental for the understanding the domain if interest (e.g., the functioning of ecosystems or certain chemical reactions). Usually, the computational media used in scientific inquiry learning contexts are largely predetermined in terms of data structures and processing mechanisms. In this sense, they are of limited "representational flexibility" regarding the free choice of data representations and algorithmic strategies.

www.dbooks.org

2.3 Specific Approaches and Examples

2.3.1 From Reactive Rule-Based Programming to Block Structures

We have already seen that even for the practical activities CT cannot be reduced to programming only. However, there is no doubt that programming resonates with and requires CT. Programs are the most prominent examples of computational artifacts. Starting the construction and creation of a program is a difficult challenge especially for beginners. Guzdial (2004) discusses different approaches and tools for novices, such as the Logo family, the rule-based family, and traditional programming approaches. These tools provide different underlying models of computation as a basis to create computational artifacts. Although there is a wider variety of options block-based programming tools became a standard for introductory programming (Weintrop & Wilensky, 2015). However, depending on which tool best supports the implementation of an idea, students should be able to choose the way how they represent their ideas as computational artifacts.

We propose a "reactive rule-based programming" tool, in which the user/learner defines program elements as reactions of a programmable agent to situations and events in the learning environment. There is a close interplay between "running" and "teaching" the agent. Whenever the agent finds itself in a situation for which there is no applicable rule the learner will be prompted to enter such a rule. The condition part of the rule is generated by the system in correspondence to the given situational parameters (context). The user then specifies the actions to be applied. Once a suitable rule is entered the system will execute it. If more and more rules are provided the system will be able to execute chains of actions without further user input.

As can be seen in Fig. 2.4, the current context determines the conditions that must be considered for the rule. If none of the already defined rules applies the system requests the student to define actions for the current situation. Then, the defined rule can be executed and the cycle restarts in a new situation or ends in a predefined end. This loop puts the learner in a first person perspective of identifying herself/himself with the agent in the environment.

The reactive rule-based programming approach is the basis for the *ctMazeStudio*, a web-based programming environment to develop algorithms steering an agent

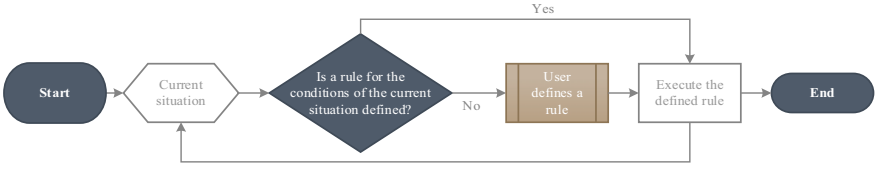


Fig. 2.4 Flow diagram for the reactive rule-based programming paradigm

who tries to find a way out of a maze. The goal for the students is to implement a universal solution that works on any maze. This learning environment contains three components: the rule editor, the behavior stage and a rule library (Fig. 2.5).

As can be seen in Fig. 2.5, the rule editor (d) provides a visual programming interface, which is available when a new situation is encountered. The editor comprises a condition component (IF part) and an action component (THEN part). For the given conditions, the students can select the desired actions for the current and corresponding situations with the same conditions to define a local "reactive" behavior. The users can also delete conditions, which implies that the corresponding rule will be applied more generally (generalization).

The rule library (c) manages the collection of all defined rules. In this user interface, the students can edit or delete already defined rules, directly enter new rules and change the order (and thus priority) of the rules to be checked. In the behavior stage (a), the behavior of the agent is visualized. Depending on the entries in the

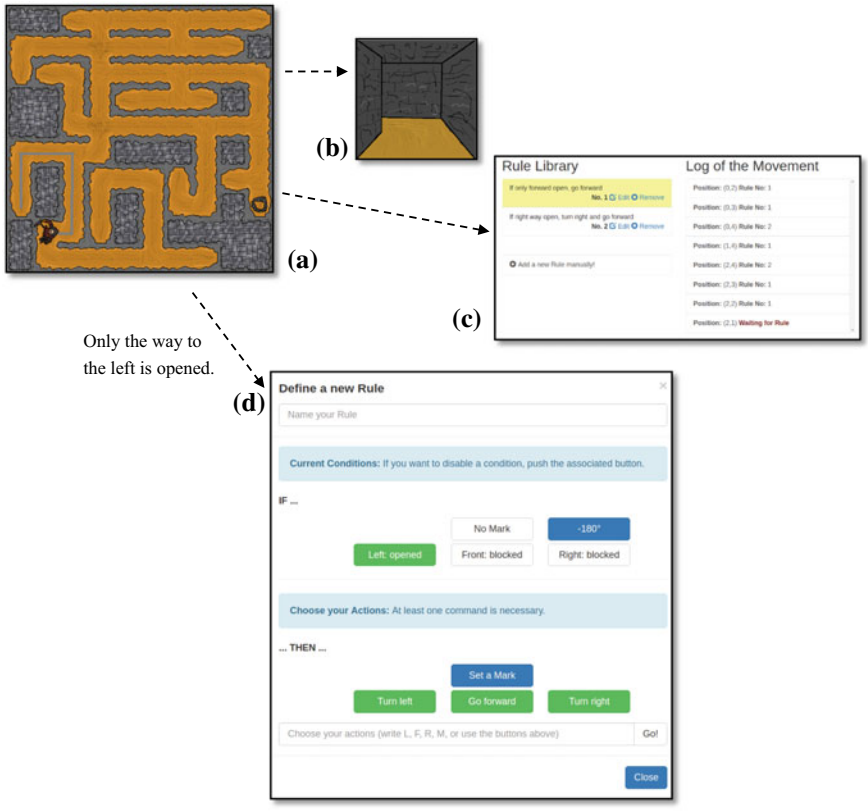


Fig. 2.5 Visualization of the reactive approach in *ctMazeStudio*

www.dbooks.org

rule library, the corresponding actions are executed and the specific entry in the rule library is highlighted.

To support CT, it is possible to apply different strategies to improve the programming code. When learners investigate and test their rulesets in consecutive situations, they may revise formerly defined rule sets through generalization (of conditions) or reordering. The challenge is to create a maximally powerful ruleset with a minimum number of rules. This requires a level of understanding that allows for predicting global behavior based on locally specified rules. In the maze example, as one of the first steps, a small set of rules will be created to implement a wall-following strategy. This strategy will be later refined, e.g., to avoid circling around islands.

The *ctMazeStudio* environment can also be programmed through a block-structured interface to represent conditions and loops governed by a global control strategy. In this sense, it provides a choice between different models of computation and thus supports "representational flexibility". Based on these options, we are currently studying transitions from rule-based reactive to block-structured iterative programs.

2.3.2 "Computational Metacognition"

CT is based on the principle of externalizing and "reifying" mental constructs or ideas in a computational medium. Accordingly, there is a close correspondence between the mental and the computational model. However, not all mental or cognitive constructs may lend themselves to such a mapping. How far can this approach be extended? In the sequel, we discuss the potential of extending this correspondence towards second order, "reflexive" constructs:

From a pedagogical point of view, thinking skills or cognitive skills comprise metacognition and reflection as important ingredients (see, e.g., Schraw, 1998). This raises the question if such a mapping from the mental to computational realm is also possible for metacognition. Indeed, advanced abstraction techniques in programming and program analysis allow for such mappings.

A first example of adding "computational metacognition" to CT environments is the application of analytic (computational) techniques to learner generated computational artifacts, especially to programs. We have to distinguish between static analyses based on the source code (as is the case for certain types of software metrics) and dynamic analyses based on the run-time behavior and results of programs (including "testing" approaches such as JUnit for Java).

Matsuzawa et al. (2017) used coding metrics to analyze characteristics of programming exercises and visualized these through a dashboard. The provision of this visualization could improve the teaching and understanding of classroom exercises in introductory programming. Manske and Hoppe (2014) have used static and dynamic analysis techniques to assess "creativity" in student programs created as solutions to so-called "Euler Problems" (see https://projecteuler.net/). Usually, it is easy to provide a brute force solution but additional insight is needed to make it more elegant

and efficient. An example challenge is finding the sum of all even Fibonacci numbers with values below 4 million (Problem 2). These Euler problems define challenges of both mathematical and computational nature. Manske and Hoppe introduced several metrics to capture different features of the programming solutions. Static and dynamic code metrics (including lines of code, cyclomatic complexity, frequency of certain abstractions, and test results) cover structural quality and compliance, while similarity based metrics address originality. In a supervised machine learning approach, classifiers have been generated based on these features together with creativity scores from expert judgments. The main problem encountered in this study was the divergence of human classification related to creativity in programming.

Logic programming in combination with so-called meta-interpreters allows for dynamically reflecting deviations of actual program behavior from intended results or specifications (Lloyd, 1987). The method of deductive diagnosis (Hoppe, 1994) uses this technique in the context of a mathematics learning environment to identify and pinpoint specific mistakes without having to provide an error or bug library. From a CT point view, these meta-level processing techniques are relevant extensions of the formal repertoire. In the spirit of "open student modeling" (Bull & Kay, 2010), not only the results but also the basic functioning of such meta-level analyses could be made available to the learners to improve reflection on their computational artifacts and to extend their understanding of computational principles.

Of course, metacognition has also been addressed and discussed in several approaches to inquiry learning (White, Shimoda, & Frederiksen, 1999; Manlove, Lazonder, & Jong, 2006; Wichmann & Leutner, 2009). In these approaches, metacognition is conceived as an element of human learning strategies, possibly supported by technology but not simulated on a computational level. The examples above show that "second order" computational reflection techniques can be applied to "first order" computational artifacts in such way as to reveal diagnostic information related to the underlying human problem-solving and construction processes. In this sense, we can identify computational equivalents of metacognitive strategies. Making these second order computations susceptible to human learning as tools of self-reflection is a big challenge, but certainly of interest for CT education.

2.4 Conclusion

The current scientific discourse centered around the notion of "Computational Thinking" is multi-disciplinary with contributions from computer science, cognitive science, and education. In addition, the curricular application contexts of CT are manifold. Still, it is important to conceive the basic computational concepts and principles in such a way as to keep up with the level of understanding developed in modern computer science. This is especially the case for the notion of "abstraction".

Our comparison of the perspectives on CT from computer science education and Inquiry Learning in science has brought forth the following main points:

www.dbooks.org

1. The essence of Computational Thinking (CT) lies in the creation of "logical artifacts" that externalize and reify human ideas in a form that can be interpreted and "run" on computers. Accordingly, CT sets a focus on computational abstractions and representations—i.e., the computational artifact and how it is constituted is of interest as such and not only as a model of some scientific phenomenon.
2. Beyond the common sense understanding of "abstraction", computational abstractions (plural!) are constructive mind tools. The available abstractions (for data representation and processing) form a repertoire of possible choices for the creation of computational artifacts.
3. Inquiry learning uses computational artifacts and/or systems as models of natural phenomena, often in the form of (programmable) simulations. Here, the choice of the computational representation is usually predetermined and not in the focus of the learners' own creative contributions.
4. Inquiry learning as well as CT-related learning activities both exhibit and rely on cyclic patterns of model progression (cycle of inquiry steps - creation/revision cycle).

There is a huge potential for a further productive co-development of CT-centric educational environments and scenarios from multiple perspectives. "Representational flexibility" as the handing over of choices related to data structuring and other abstractions to the learners is desirable from a computer science point of view. This does not rule out the meaningful and productive use of more fixed computational models in other learning contexts. Yet, this future co-development of CT should benefit from taking up and exploring new types of abstractions and models of computation (including, e.g., different abstract engines or meta-level reasoning techniques) to enrich the learning space of CT. This may also reconnect the discourse to epistemological principles.

Acknowledgements This article is dedicated to the memory of Sören Werneburg who made important and essential contributions to this work, including the full responsibility for the design and implementation of the *ctMazeStudio* and *ctGameStudio* environments.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Ash, D. (2003). Inquiry Thoughts, Views and Strategies for the K-5 Classroom. In *Foundations: A monograph for professionals in science, mathematics and technology education*.

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., Kinnebrew, J., Dickes, A., Farris, A., Sengupta, P., Winger, J., … Biswas, G. (2012). A science learning environment using a computational thinking approach. In *Proceedings of the 20th International Conference on Computers in Education*.

Bull, S., & Kay, J. (2010). Open Learner Models. In *Advances in intelligent tutoring systems* (pp. 301–322): Springer.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67–69.

Curzon, P., & McOwan, P. W. (2016). *The power of computational thinking: Games, magic and puzzles to help you become a computational thinker*: World Scientific

diSessa, A. (2000). Changing minds: Computers. *Learning and Literacy*.

Giere, R. (1988). Laws, theories, and generalizations.

Guzdial, M. (2004). Programming environments for novices. *Computer Science Education Research, 2004,* 127–154.

Hartmann, W., Nievergelt, J., & Reichert, R. (2001). Kara, finite state machines, and the case for programming as part of general education. In *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments*.

Hoppe, H. U. (1994). Deductive error diagnosis and inductive error generalization for intelligent tutoring systems. *Journal of Interactive Learning Research, 5*(1), 27.

Hu, C. (2011). Computational thinking: What it might mean and what we might do about it. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*.

ISTE, & CSTA. (2011). *Computational thinking in K–12 education leadership toolkit*.

Kafura, D., & Tatar, D. (2011). Initial experience with a computational thinking course for computer science students. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*.

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.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., et al. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

Lehrer, R., & Schauble, L. (2006). Scientific thinking and science literacy. In *Handbook of Child Psychology*.

Lloyd, J. W. (1987). Declarative error diagnosis. *New Generation Computing, 5*(2), 133–154.

Locke, J. (1700). *An essay concerning human understanding*.

Lockwood, J., & Mooney, A. (2017). Computational thinking in education: Where does it fit? *A Systematic Literary Review*.

Manlove, S., Lazonder, A. W., & Jong, T. D. (2006). Regulative support for collaborative scientific inquiry learning. *Journal of Computer Assisted Learning, 22*(2), 87–98.

Manske, S., & Hoppe, H. U. (2014). Automated indicators to assess the creativity of solutions to programming exercises. In *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*.

Matsuzawa, Y., Tanaka, Y., Kitani, T., & Sakai, S. (2017). A demonstration of evidence-based action research using information dashboard in introductory programming education. In *IFIP World Conference on Computers in Education*.

Minner, D. D., Levy, A. J., & Century, J. (2010). Inquiry-based science instruction—what is it and does it matter? Results from a research synthesis years 1984 to 2002. *Journal of Research in Science Teaching, 47*(4), 474–496.

Nersessian, N. J. (1992). How do scientists think? Capturing the dynamics of conceptual change in science. *Cognitive Models of Science, 15,* 3–44.

NRC. (2008). *Public participation in environmental assessment and decision making*: National Academies Press.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning, 1*(1), 95–123.

Pedaste, M., Mäeots, M., Siiman, L. A., De Jong, T., Van Riesen, S. A., Kamp, E. T., et al. (2015). Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review, 14,* 47–61.

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.

www.dbooks.org

Rieber, L. P. (1996). Microworlds. In *Handbook of research for educational communications and technology* (Vol. 2, pp. 583–603).

Schraw, G. (1998). Promoting general metacognitive awareness. *Instructional Science, 26*(1–2), 113–125.

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.

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

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*.

White, B. Y., Shimoda, T. A., & Frederiksen, J. R. (1999). Enabling students to construct theories of collaborative inquiry and reflective learning: Computer support for metacognitive development. *International Journal of Artificial Intelligence in Education (IJAIED), 10,* 151–182.

Wichmann, A., & Leutner, D. (2009). Inquiry learning: Multilevel support with respect to inquiry, explanations and regulation during an inquiry cycle. *Zeitschrift für Pädagogische Psychologie, 23*(2), 117–127.

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.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 3
MIT App Inventor: Objectives, Design, and Development



Evan W. Patton, Michael Tissenbaum and Farzeen Harunani

Abstract MIT App Inventor is an online platform designed to teach computational thinking concepts through development of mobile applications. Students create applications by dragging and dropping components into a design view and using a visual blocks language to program application behavior. In this chapter, we discuss (1) the history of the development of MIT App Inventor, (2) the project objectives of the project and how they shape the design of the system, and (3) the processes MIT uses to develop the platform and how they are informed by computational thinking literature. Key takeaways include use of components as abstractions, alignment of blocks with student mental models, and the benefits of fast, iterative design on learning.

Keywords Computational thinking · Computational action · Educational technology · Programming languages · Block-based programming · Mobile learning

3.1 Introduction

The smartphone is an information nexus in today's digital age, with access to a nearly infinite supply of content on the web, coupled with rich sensors and personal data. However, people have difficulty harnessing the full power of these ubiquitous devices for themselves and their communities. Most smartphone users consume technology without being able to produce it, even though local problems can often be solved with mobile devices. How then might they learn to leverage smartphone capabilities to solve real-world, everyday problems? MIT App Inventor is designed to democratize this technology and is used as a tool for learning computational thinking in a variety

E. W. Patton (✉) · M. Tissenbaum · F. Harunani
Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: ewpatton@csail.mit.edu; ewpatton@mit.edu

M. Tissenbaum
e-mail: miketissenbaum@gmail.com

F. Harunani
e-mail: farzeen@mit.edu

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_3

www.dbooks.org

of educational contexts, teaching people to build apps to solve problems in their communities.

MIT App Inventor is an online development platform that anyone can leverage to solve real-world problems. It provides a web-based "What you see is what you get" (WYSIWYG) editor for building mobile phone applications targeting the Android and iOS operating systems. It uses a block-based programming language built on Google Blockly (Fraser, 2013) and inspired by languages such as StarLogo TNG (Begel & Klopfer, 2007) and Scratch (Resnick et al., 2009; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), empowering anyone to build a mobile phone app to meet a need. To date, 6.8 million people in over 190 countries have used App Inventor to build over 24 million apps. We offer the interface in more than a dozen languages. People around the world use App Inventor to provide mobile solutions to real problems in their families, communities, and the world. The platform has also been adapted to serve requirements of more specific populations, such as building apps for emergency/first responders (Jain et al., 2015) and robotics (Papadakis & Orfanakis, 2016).

In this chapter, we describe the goals of MIT App Inventor and how they have influenced our design and development—from the program's inception at Google in 2008, through the migration to MIT, to the present day. We discuss the pedagogical value of MIT App Inventor and its use as a tool to teach and encourage people of all ages to think and act computationally. We also describe three applications developed by students in different parts of the world to solve real issues in their communities. We conclude by discussing the limitations and benefits of tools such as App Inventor and proposing new directions for research.

3.2 MIT App Inventor Overview

The MIT App Inventor user interface includes two main editors: the design editor and the blocks editor. The design editor, or designer (see Fig. 3.1), is a drag and drop interface to lay out the elements of the application's user interface (UI). The blocks editor (see Fig. 3.2) is an environment in which app inventors can visually lay out the logic of their apps using color-coded blocks that snap together like puzzle pieces to describe the program. To aid in development and testing, App Inventor provides a mobile app called the App Inventor Companion (or just "the Companion") that developers can use to test and adjust the behavior of their apps in real time. In this way, anyone can quickly build a mobile app and immediately begin to iterate and test.

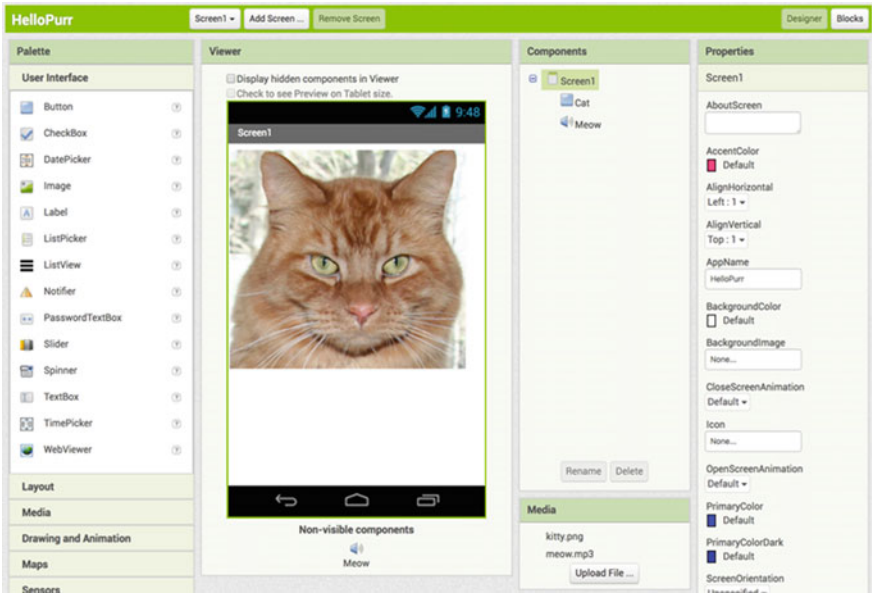


Fig. 3.1 App Inventor's design editor. App inventors drag components out from the palette (far left) to the viewer (center left) to add them to the app. Inventors can change the properties of the components (far right). An overview of the screen's components and project media are also displayed (center right)

3.3 MIT App Inventor Design Goals

In the design of MIT App Inventor, introducing mobile app development in educational contexts was a central goal. Prior to its release, most development environments for mobile applications were clunky, only accessible with expertise in systems level or embedded programming, or both. Even with Google's Android operating system and the Java programming language, designing the user interface was a complex task. Further, use of the platform required familiarity with Java syntax and semantics, and the ability to debug Java compilation errors (e.g., misspelled variables or misplaced semicolons) for success. These challenges presented barriers to entry for individuals not versed in computer science, App Inventor's target demographic. We briefly highlight and discuss design goals for the App Inventor project, specifically, the use of *components* to abstract some of the complexity of platform behavior, and the use of *blocks* to eliminate complexity of the underlying programming language. These goals can be further explained as aligning the visual language to the mental models of young developers and enabling exploration through fast, iterative design.

www.dbooks.org

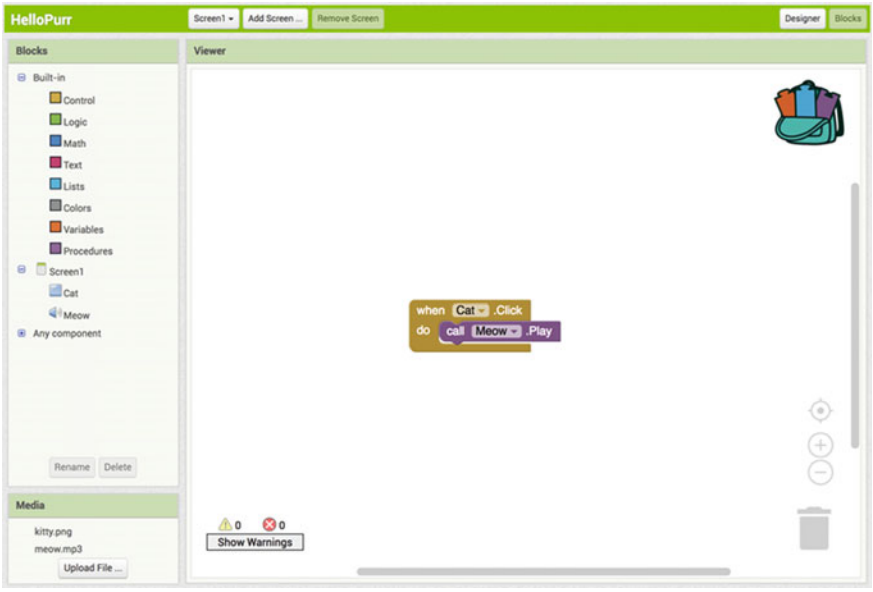


Fig. 3.2 App Inventor's blocks editor. Blocks code is typically read left to right, top to bottom. In this example, one would read "when Cat click, do call Meow play," that is, play the meow sound when the cat is clicked

3.3.1 Component Abstraction for Platform Behavior

Components are core abstractions in MIT App Inventor. Components reduce the complexity of managing interactions with platform-specific application programming interfaces (APIs) and details concerning state management of device hardware. This allows the user to think about the problem at hand rather than the minutia typically required of application developers. For example, someone planning to use MIT App Inventor to build an app to use the global positioning system (GPS) to track movement need not be concerned with application lifecycle management, GPS software and hardware locks, or network connectivity (in case location detection falls back to network-based location). Instead, the app developer adds a location sensor component that abstracts away this complexity and provides an API for enabling and processing location updates. More concretely, this implementation reduces 629 lines of Java code to 23 blocks, of which only two are required to accomplish location tracking. This reduction in complexity enables app inventors to focus on the problem at hand and quickly accomplish a goal.

Components are made up of three major elements: properties, methods, and events. *Properties* control the state of the component and are readable and/or writable by the app developer. For example, the enabled property of the location sensor includes the functionality required to configure the GPS receiver and to manage its state while the app is in use. *Methods* operate on multiple inputs and possibly return a result. *Events*

respond to changes in the device or app state based on external factors. For example, when the app user changes their location, the location changed event allows the app logic to respond to the change.

3.3.2 Blocks as Logic

In MIT App Inventor, users code application behavior using a block-based programming language. There are two types of blocks in App Inventor: built-in blocks and component blocks. The built-in blocks library provides the basic atoms and operations generally available in other programming languages, such as Booleans, strings, numbers, lists, mathematical operators, comparison operators, and control flow operators. Developers use component blocks (properties, methods, and events) to respond to system and user events, interact with device hardware, and adjust the visual and behavioral aspects of components.

3.3.2.1 Top-Level Blocks

All program logic is built on three top-level block types: global variable definitions, procedure definitions, and component event handlers. Global variables provide named slots for storing program states. Procedures define common behaviors that can be called from multiple places in the code. When an event occurs on the device, it triggers the corresponding application behavior prescribed in the event block. The event handler block may reference global variables or procedures. By limiting the top-level block types, there are fewer entities to reason about.

3.3.3 Mental Modeling

The development team for App Inventor considered a number of restrictions when designing the environment. We examine a few design decisions, the rationale behind them, and their effects on computational thinking within App Inventor.

3.3.3.1 What You See Is What You Get (WYSIWYG)

The design editor for App Inventor allows developers to see how the app will appear on the device screen and adjust the form factor of the visualized device (e.g., phone or tablet). Adjustments to properties of the visual components, for example, background color and size, are reflected in real time. Apps can also be run in a *live development* mode using the Companion, which we will be discussed in more detail below.

www.dbooks.org

The App Inventor team recently added capability for creating map-based applications. The functionality allows app inventors to drag, drop, and edit markers, lines, polygons, rectangles, and circles in their maps, as well as integrate web-based data from geographic information systems (GIS) to build content-rich apps. This way, the user can move the content around easily to achieve great results without needing to provide most of the logic for this in code.

3.3.3.2 Design Time Component Creation

Unlike many programming languages, App Inventor limits runtime creation of new entities. This provides multiple benefits. First, by explicitly positioning all components in the app, the user can visualize it clearly rather than having to reason about things that will not exist until a future time. Second, it reduces the chances of users introducing cyclic memory dependencies in the user interface that would eventually cause the app to run out of memory. This encourages app inventors to think about how to appropriately structure their applications and reuse components to avoid overloading the system or their end users.

3.3.3.3 Natural Numbering

The number system in App Inventor assumes a starting value of 1, in line with children's counting skills (Gelman & Gallistel, 1978). This is unlike most programming languages, which are more aligned with machine architecture and therefore start at 0.

3.3.4 Fast Iteration and Design Using the Companion

A key feature of MIT App Inventor is its live development environment for mobile applications. App Inventor provides this by means of a companion app installed on the user's mobile device. The App Inventor web interface sends code to the companion app, which interprets the code and displays the app in real time to the developer (Fig. 3.3). This way, the user can change the app's interface and behavior in real time. For example, a student making a game involving the ball component may want to bounce the ball off the edge of the play area. However, an initial implementation might have the ball collide with the wall and then stop. After discovering the Ball.EdgeReached event, the student can add the event and update the direction of the ball using the Ball.Bounce method. By testing the app and adjusting its programming in response to undesired behavior, students can explore more freely.

The traditional build cycle for an Android app involves writing code in a text editor or integrated development environment, and rebuilding the application for testing may often take minutes, whereas making a change in the live development

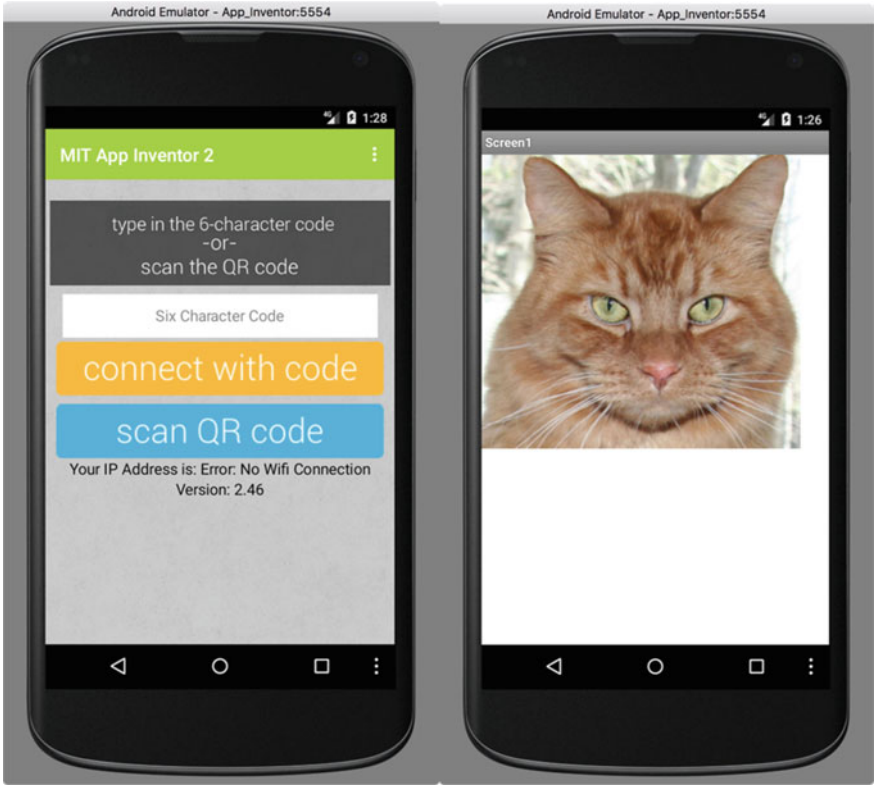


Fig. 3.3 The MIT Companion app interface for Android (left). After establishing a connection with the user's browser session, the active project is displayed in the companion app (right). See Fig. 3.1 for the designer view of the same project

environment typically takes effect in 1–2 s. Seeing changes reflected in the app quickly means that students can explore and even make mistakes while exploring, because the time cost of those mistakes is relatively small.

3.4 The History of MIT App Inventor

The App Inventor project began at Google in 2007 when Prof. Hal Abelson of MIT went on sabbatical at Google Labs. The project leads were inspired by increased interest in educational blocks programming languages, such as Scratch, and the release of the new Android operating system. This educational project was migrated to MIT when Google closed Google Labs in 2011. In this section, we briefly cover inception and early development of the App Inventor platform, first at Google, and then at MIT.

www.dbooks.org

3.4.1 Inception at Google

Hal Abelson conceived the idea of App Inventor while on sabbatical at Google Labs in 2007. Abelson had previously taught a course at MIT on mobile programming, but at the time mobile app development required significant investment on the part of developers and development environments. Also in 2007, Google publicly announced the Android operating system. Abelson and Mark Friedman of Google began developing an intermediate language between the blocks language and Java APIs for Android, called Yet Another Intermediate Language (YAIL). The project was intended to help younger learners program for Android. Abelson and Friedman generated YAIL from a block-based language based on OpenBlocks (Roque, 2007), and the design of which was drawn from StarLogo TNG (Begel & Klopfer, 2007). The user interface and related components embodied Papert's idea of "powerful ideas in mind-size bites" (Papert, 1993). The Google version of the project terminated at the end of 2011, but the educational technology was transferred to MIT so that development and educational aspects could continue (Kincaid, 2011). Prof. Abelson joined Prof. Eric Klopfer of the Scheller Teacher Education Program lab and Prof. Mitch Resnick of the MIT Media Lab, forming a group called the MIT Center for Mobile Learning to carry on the App Inventor vision.

3.4.2 Educational Expansion at MIT

In late 2011, Google transferred stewardship of the App Inventor project to MIT. Much of the development focused on increasing capabilities to support educational goals of the project. At this time, the team developed additional curricula, making them freely available to teachers for computer science and computational thinking education. The MIT team also hosted a number of 1-day workshops, primarily around the northeast United States, training teachers in the pedagogy of App Inventor. We now focus on guided and open exploration in our materials rather than presenting students with step-by-step instructions in order to encourage self-guided learning. By making mistakes, students have the opportunity to practice more of the computational thinking principles, such as debugging, described by Brennan and Resnick (2012).

Technical development at MIT focused on development of new components including robotics (LEGO™ EV3), cloud-oriented data storage (CloudDB), and geographic visualization (Map). App Inventor team also developed Internet of Things related extensions so learners could interact with physical hardware external to their mobile devices, and to leverage the growing collection of small computer boards, such as Arduino, BBC micro:bit, and Raspberry Pi. To this day, the team continues its work of development, creating complementary educational materials in parallel.

3.5 MIT App Inventor in Education

The primary aim of MIT App Inventor is providing anyone with an interest in building apps to solve problems with the tools necessary to do so. Instructional materials developed by the team are primarily oriented toward teachers and students at the middle- and high-school levels, but app inventors come in all ages from around the world. In this section, we describe a few of the key components of the MIT App Inventor educational strategy, including massively online open courses (MOOCs) focused on MIT App Inventor, the Master Trainer (MT) program, the extensions functionality of App Inventor that allows incorporation of new material for education, and research projects that have leveraged App Inventor as a platform for enabling domain-specific computing.

3.5.1 Massive Open Online Courses

A desire to learn computational thinking has driven a proliferation of online educational material that anyone can access to increase their knowledge and understanding. As we continue to integrate information technology into our daily lives, mobile devices, and other new technologies, we can observe that a deeper understanding of computing is necessary to be an effective member of society, and those who learn computational thinking will have an advantage in our knowledge-driven economy.

Many massive open online courses have been developed wholely or in part using App Inventor. For example, an App Inventor EdX course closely integrates with the AP CS Principles course and incorporates many computational thinking elements. Students therefore can both build their own mobile apps and learn core competencies related to computation.

3.5.2 MIT Master Trainers Program

MIT provides special instruction to educators through the Master Trainers program.[1] A prototype of the Master Trainers program began during a collaboration with the Verizon App Challenge in 2012. Skilled App Inventor educators were recruited and given a small amount of special training to help mentor and train teams who subsequently won the App Challenge. The current Master Trainers program was conceived in 2015, to "grow a global community of experts on mobile app development who are available to guide others through the exploration of mobile app creation…, thus providing a pathway into computer science, software development, and other disciplines relevant in today's digital world."

[1] http://appinventor.mit.edu/explore/master-trainers.html.

www.dbooks.org

In order to become a Master Trainer, one must demonstrate proficiency in App Inventor, for example, through taking the App Inventor EdX MOOC. The MOOC is highly integrated with computational thinking concepts, giving students a strong foundation in the concepts and practices associated with computational thinking. Aspiring Master Trainers then complete a 10-week online reading course covering topics such as App Inventor's mission and philosophy, pedagogy of teaching children and adults, constructionism, and design thinking. Lastly, there is an on-site 3-day workshop at MIT where participants dive into App Inventor features and learn to use App Inventor in a classroom to foster creativity, collaboration, and problem-solving. At the time of writing, there were 57 master trainers in 19 countries.

3.5.3 Extensions

Anyone with Java and Android programming experience can write their own components for App Inventor using our extension mechanism. For example, MIT recently published a suite of Internet of things (IOT)-related extensions[2] for interfacing with Arduino 101 and BBC micro:bit microcontrollers, with support for other platforms in development. Using these extensions, teachers can assemble custom curricula to leverage these technologies in the classroom and encourage their students to explore the interface between the world of software and the world of hardware.

We foresee the development of extensions related to artificial intelligence technologies, including deep learning, device support for image recognition, sentiment analysis, natural language processing, and more. Ideally, these complex technologies could be leveraged by anyone looking to solve a problem with the smartphone as a platform.

3.5.4 Research Projects

In addition to its pedagogical applications, App Inventor offers excellent opportunities for research in education and other areas. Early work focused on understanding how to appropriately name components for educational use (Turbak, Wolber, & Medlock-Walton, 2014). Usability in domain-specific contexts, such as humanitarian needs (Jain et al., 2015) and educational settings (Morelli, De Lanerolle, Lake, Limardo, Tamotsu, & Uche, 2011; Xie, Shabir, & Abelson, 2015), is also an area of interest. More recently, App Inventor has been used as a mechanism for data collection and visualization (Harunani, 2016; Mota, Ruiz-Rube, Dodero, & Figueiredo, 2016; Martin, Michalka, Zhu, & Boudelle, 2017). We are currently exploring expanding App Inventor's capabilities to include real-time collaboration between students, which should yield additional educational opportunities (Deng, 2017).

[2] http://iot.appinventor.mit.edu.

3.6 Empowerment Through Programming

By placing the output of student programming on mobile devices, App Inventor allows students to move their work out of traditional computer labs, and into their everyday lives and communities. This transition has powerful implications for what students create and how they envision themselves as digital creators. It allows students to shift their sense of themselves from individuals who "know how to code" to members of a community empowered to have a real impact in their lives and those of others. Below, we outline how App Inventor moves computing education from a focus on the theoretical to a focus on the practical, how we can reconceptualize computing education through a lens of computational action, and how we support students to engage in a broader community of digitally empowered creators.

3.6.1 From Theoretical to Practical

Traditional computer science curricula at the university level often focus on theory and include evaluation tools (e.g., Big-O notation of algorithms) and comprehension of the space and time complexity of data structures. Instead, App Inventor curricula focus on using a language practically to solve real-world problems. Rather than placing emphasis on learning concepts such as linked lists or key–value mappings, App Inventor hides the complexity of these data structures behind blocks so that students can spend more time designing apps that perform data collection and analysis, or integrate with a range of sensors and actuators interacting with external environments. This allows for a top-down, goal-based decomposition of the problem rather than a bottom-up approach, although App Inventor does not preclude such a strategy.

3.6.2 Computational Thinking

The concept of computational thinking was first used by Seymour Papert in his seminal book Mindstorms: Children, computers, and powerful ideas (1993); however, it was largely brought into the mainstream consciousness by Jeannette Wing in 2006. For Wing, computational thinking is the ability to think like a computer scientist. In the decade since, many educational researchers have worked to integrate computational thinking into modern computing and STEM curricula (Tissenbaum, Sheldon, & Sherman, 2018). However, the explosive growth of computational thinking has also resulted in a fragmentation of its meaning, with educational researchers, curriculum designers, and teachers using different definitions, educational approaches, and methods of assessments (Denning, 2017). There have been attempts to reconcile these differences (National Academy of Sciences, 2010) and to bring leading

www.dbooks.org

researchers together to compare and contrast these perspectives (Tissenbaum et al., 2018).

For most educational practitioners and researchers, computational thinking is dominated by an epistemological focus on computational thinking, in which students learn programming concepts (such as loops, variables, and data handling) and the use of abstractions to formally represent relationships between computing and objects in the real world (Aho, 2012). While this view has become the most prominent view of computational thinking, Papert critiqued mainstream schooling's emphasis on these "skills and facts" as a bias *against ideas* (Papert, 2000). Papert went further, arguing that students should be encouraged to follow their own projects and that learning the necessary skills and knowledge would arise as students encountered new problems and needed to solve (or not solve) them. This position of computational thinking and computing education fits more naturally with the ways that professionals engage in computer science: in pursuit of finishing a project, problems naturally come up and computer scientists reach out to the community through sites like Stack Overflow, or search the web for tutorials or other support. This disconnect between how we teach computing and how it is practiced in the real world requires us to critically reexamine theoretical and practical approaches. Below, we argue for an approach to computing education, termed computational action, that we believe matches these broader ideals.

3.6.3 Computational Action

While the growth of computational thinking has brought new awareness to the importance of computing education, it has also created new challenges. Many educational initiatives focus solely on the programming aspects, such as variables, loops, conditionals, parallelism, operators, and data handling (Wing, 2006), divorcing computing from real-world contexts and applications. This decontextualization threatens to make learners believe that they do not need to learn computing, as they cannot envision a future in which they will need to use it, just as many see math and physics education as unnecessary (Flegg et al., 2012; Williams et al., 2003).

This decontextualization of computing education from the actual lives of students is particularly problematic for students underrepresented in the fields of computing and engineering, such as women and other learners from nondominant groups. For these students, there is a need for their work to have an impact in their community and for it to help them develop a sense of fit and belonging (Pinkard et al., 2017). Lee and Soep (2016) argue that a critical perspective for computing is essential for students to develop a critical consciousness around what they are learning and making, moving beyond simply programming, instead of asking the students what they are programming and why they are programming it.

In response, the App Inventor team advocates for a new approach to computing education that we call *computational action*. The computational action perspective on computing argues that while learning about computing, young people should also

have opportunities to create with computing which have direct impact on their lives and their communities. Through our work with App Inventor, we have developed two key dimensions for understanding and developing educational experiences that support students in engaging in computational action: (1) computational identity and (2) digital empowerment. Computational identity builds on prior research that showed the importance of young people's development of scientific identity for future STEM growth (Maltese & Tai, 2010). We define computational identity as a person's recognition that they can use computing to create change in their lives and potentially find a place in the larger community of computational problem-solvers. Digital empowerment involves instilling in them the belief that they can put their computational identity into action in authentic and meaningful ways.

Computational action shares characteristics with other approaches for refocusing computing education toward student-driven problem-solving, most notably *computational participation* (Kafai, 2016). Both computational action and computational participation recognize the importance of creating artifacts that can be used by others. However, there is a slight distinction between the conceptualizations of community in the two approaches. In computational participation, community largely means the broader community of learners engaging in similar computing practices (e.g., the community of Scratch programmers that share, reuse, and remix their apps). While such a learning community may be very beneficial to learners taking part in a computational action curriculum, the community of greater importance is the one that uses or is impacted by the learners' created products (e.g., their family, friends, and neighbors). This computational identity element of computational action acknowledges the importance of learners feeling a part of a computing community (i.e., those that build and solve problems with computing), but it is not a requirement that they actively engage with this larger community. A small group of young app builders, such as those described below, may develop significant applications and believe they are authentically part of the computing community, without having connected with or engaged with it in a deep or sustained way as would be expected in computational participation.

Through students' use of App Inventor, we have seen this computational action approach produce amazing results. Students in the United States have developed apps to help a blind classmate navigate their school (Hello Navi[3]); students in Moldova developed an app to help people in their country crowdsource clean drinking water (Apa Pura[4]); and as part of the CoolThink@JC project, students in Hong Kong created an app, "Elderly Guardian Alarm," to help the elderly when they got lost. Across these projects, we see students engaging with and facilitating change in their communities, while simultaneously developing computational identities.

[3] https://www.prnewswire.com/news-releases/321752171.html.

[4] The Apa Pura Technovation pitch video is available online at https://youtu.be/1cnLiSySizw.

www.dbooks.org

3.6.4 Supporting a Community Around Computation and App Creation

We started the App of the Month program in 2015 in order to encourage App Inventors to share their work with the community. Any user can submit their app to be judged in one of four categories: Most Creative, Best Design, Most Innovative, and Inventor. Submissions must be App Inventor Gallery links, so that any user can remix winning apps. Furthermore, apps are judged in two divisions: youth and adult.

Now, 3 years after the program's inception, approximately 40 apps are submitted each month. More youth tend to submit than adults, and significantly more male users submit than female users, especially in the adult division. While submissions come in from all over the world, India and the USA are most highly represented.

Themes of submitted apps vary widely. Many students submit "all-in-one" apps utilizing the Text to Speech and Speech Recognizer components. Adults often submit learning apps for small children. Classic games, such as Pong, also get submitted quite frequently. Teachers tend to submit apps that they use in their classrooms.

Perhaps most importantly, students and adults alike submit apps designed to solve problems within their own lives or their communities. For example, a recent submitter noticed that the Greek bus system is subject to many slowdowns, so he built an app that tracks buses and their routes. Similarly, a student noticed that many of her peers were interested in reading books, but did not know how to find books they would like, so she built an app that categorizes and suggests popular books based on the Goodreads website.

However, not all users fit the same mold. One student found that he enjoys logic- and math-based games, and after submitting regularly for about a year, his skill improved tremendously. Hundreds of people have remixed his apps from the Gallery, and even downloaded them from the Google Play Store, encouraging the student to pursue a full-time career in game development.

The App of the Month program, as a whole, encourages users to think of App Inventor as a tool they can use in their daily lives and off-the-screen communities. It also provides incentive to share their apps and recognition for their hard work. Users go to App Inventor to solve problems—which makes them App Inventors themselves.

3.7 Discussion

We have seen in detail many aspects of the MIT App Inventor program from the development and educational perspective. There are some misconceptions, limitations, and benefits that are important to highlight.

3.7.1 Common Misconceptions

One common position detractors take is that blocks programming is not real programming (often comparing blocks languages to text languages). This is a false dichotomy if one understands programming to be the act of describing to a computer some realization of a Turing machine. The examples presented in earlier sections highlight how people use MIT App Inventor to solve real problems they face in their communities. To this end, younger individuals recognize that through tools such as App Inventor they can effect real change in their community, if not the whole world. Novice users who begin learning programming with blocks languages also tend to go further and continue more often than learners of textual languages (Weintrop & Wilensky, 2015).

Another common misconception is that creating mobile applications is something that only experts and those who have a lot of experience programming can do. However, students across the K-12 spectrum use App Inventor to develop their own mobile applications with little to no prior experience. For instance, the CoolThink@JC curriculum targets over 15,000 students in Hong Kong from grades 4–6. This intervention has enabled these elementary students to learn both to think computationally and to develop their own apps to address local issues (Kong et al., 2017).

3.7.2 Limitations

Computational proficiency is often assessed in traditional textual representations; for example, the AP Computer Science A exam is assessed in the Java programming language. For students who learn in block-based representations, it can be difficult to transition to textual representations. Therefore, it is important to help students transition to textual languages, while ensuring that knowledge gained in the visual language is not lost. Prof. Dave Wolber and a team at USF are actively addressing this through the development of the App Inventor Java Bridge.[5] The Java bridge allows anyone to translate an App Inventor application into a Java application compatible with Android Studio, the official text-based development environment used to build native Android applications. This enables students to transition from the AP Computer Science 0 curriculum to AP Computer Science A.

Another current limitation of App Inventor is that its design inhibits code reuse. Code reuse is one of the key computational thinking concepts in Brennan and Resnick's framework (2012). Many text-based languages provide robust support for code libraries and dependency management, allowing app developers to build on each other's work more easily. While App Inventor provides a gallery for publishing completed app source code, the community has yet to develop the granularity of libraries common in other programming languages. This presents an opportunity to

[5] http://appinventortojava.com/.

www.dbooks.org

continue to grow the platform and user community and is a worthy subject for further exploration.

3.7.3 Benefits of Visual Programming for Mobile

Users of the App Inventor platform benefit from being able to repurpose the computational thinking skills they learn to interface with physical space in the external world. The visual programming of App Inventor and the abstraction and compartmentalization of concepts into components and blocks allow the app inventor to focus more on decomposing their problems into solvable elements. The facility of running apps on mobile devices allows the students to experience their own apps as part of an ecosystem they interact with daily, and with which they are intimately familiar. Since this encapsulation reduces the time it takes to build an app, even a straightforward prototype, app inventors can quickly grasp and iterate without paying a significant cost in terms of a compile-load-run cycle that is typical with mobile app development.

3.8 Conclusions

The MIT App Inventor project continues to push the boundaries of education within the context of mobile app development. Its abstraction of hardware capabilities and the reduction of complex logic into compact representations allows users to quickly and iteratively develop projects that address real-world problems. We discussed how App Inventor's curriculum development incorporates elements of computational thinking and encourages computational action with real-world effects. We also presented a number of projects that effectively accomplish this mission. We continue to grow the platform to democratize access to newer technologies, preparing future generations for a world in which computational thinking is a central part of problem-solving.

3.8.1 Future Vision

We already observe a rise in the growth of machine learning technologies. These technologies offer new ways of engaging with the world and could dramatically affect the future of technology and society. To support educating youth in this family of technologies, we are actively developing artificial intelligence and machine learning components, as well as curricula teachers can use to instruct students in these technologies.

In the future, we expect that households will be increasingly computationally literate. Already we are observing toddlers making use of devices such as phones and tablets to learn and engage the world in different ways. These technologies will become nearly universal in the near future, mandating increased pedagogy around computational thinking as well as the creation of environments to aid young children in using these tools to solve problems is critical. We must help them become producers and change makers rather than simply consumers. Increasingly, we move toward a world where functionality is abstracted, or provided as pieces that the computationally literate can combine for novel solutions for any problem. App Inventor will continue to push these boundaries by exploring bleeding edge technologies and integrating them within a mobile context.

Lastly, we are moving toward economies of knowledge. In these economies, access to new innovations and ways of solving problems will differentiate individuals competing globally (Powell & Snellman, 2004). Tools that provide increased abstraction for solving problems will offer more advantages to individuals than traditional engineering approaches.

Acknowledgements The authors would like to thank Prof. Hal Abelson, Karen Lang, and Josh Sheldon for their input, and discussions of material in the manuscript. App Inventor has received financial support from Google, NSF grant #1614548, Hong Kong Jockey Club, Verizon Foundation, and individual contributors.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Begel, A., & Klopfer, E. (2007). Starlogo TNG: An introduction to game development. *Journal of E-Learning, 53,* 146.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In: *Proceeding of the 2012 AERA*.

Deng, X. (2017). *Group collaboration with App Inventor* (Masters thesis, Massachusetts Institute of Technology).

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33–39.

Flegg, J., Mallet, D., & Lupton, M. (2012). Students' perceptions of the relevance of mathematics in engineering. *International Journal of Mathematical Education in Science and Technology*, *43*(6), 717–732.

Fraser, N. (2013). Blockly: A visual programming editor. https://developers.google.com/blockly/.

Gelman, R., & Gallistel, C. R. (1978). *The child's understanding of number*. Cambridge, MA: Harvard University Press.

Harunani, F. (2016). AppVis: Enabling data-rich apps in App Inventor. Masters Thesis, University of Massachusetts, Lowell.

Jain, A., Adebayo, J., de Leon, E., Li, W., Kagal, L., Meier, P., et al. (2015). Mobile application development for crisis data. *Procedia Engineering, 107,* 255–262.

Kafai, Y. B. (2016). From computational thinking to computational participation in K-12 education. *Communications of the ACM, 59*(8), 26–27.

www.dbooks.org

Kincaid, J. (2011). Google gives Android App Inventor a new home at MIT Media Lab. Techcrunch. Retrieved March 04, 2018, from https://techcrunch.com/2011/08/16/google-gives-android-app-inventor-a-new-home-at-mit-media-lab/.

Kong, S., Abelson, H., Sheldon, J., Lao, A., Tissenbaum, M., & Lai, M. (2017). Curriculum activities to foster primary school students' computational practices in block-based programming environments. In *Proceedings of Computational Thinking Education* (p. 84).

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., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16.

Maltese, A. V., & Tai, R. H. (2010). Eyeballs in the fridge: Sources of early interest in science. *International Journal of Science Education, 32*(5), 669–685.

Martin, F., Michalka, S., Zhu, H., & Boudelle, J. (2017, March). Using AppVis to build data-rich apps with MIT App Inventor. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 740–740). ACM.

Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2011, March). Can Android App Inventor bring computational thinking to k-12. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)* (pp. 1–6).

Mota, J. M., Ruiz-Rube, I., Dodero, J. M., & Figueiredo, M. (2016). Visual environment for designing interactive learning scenarios with augmented reality. *International Association for Development of the Information Society*.

National Academies of Science. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington DC: National Academies Press.

Papadakis, S., & Orfanakis, V. (2016, November). The combined use of Lego Mindstorms NXT and App Inventor for teaching novice programmers. In *International Conference EduRobotics 2016* (pp. 193–204). Springer, Cham.

Papert, S. (1990). *A critique of technocentrism in thinking about the school of the future*. Cambridge, MA: Epistemology and Learning Group, MIT Media Laboratory.

Papert, S. (1993). *Mindstorms: Children, computers, and powerful ideas* (2nd ed.). Basic Books.

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal, 39*(3–4), 720–729.

Pinkard, N., Erete, S., Martin, C. K., & McKinney de Royston, M. (2017). Digital Youth Divas: Exploring narrative-driven curriculum to spark middle school girls' interest in computational activities. *Journal of the Learning Sciences*, (just-accepted).

Powell, W. W., & Snellman, K. (2004). The knowledge economy. *Annual Reviews in Sociology, 30,* 199–220.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60–67.

Roque, R. V. (2007). OpenBlocks: *An extendable framework for graphical block programming systems*. Doctoral dissertation, Massachusetts Institute of Technology.

Tissenbaum, M., Sheldon, J., & Sherman, M. (2018). The state of the field in computational thinking assessment. In *To Appear in the Proceedings of the 2018 International Conference of the Learning Sciences*. London.

Turbak, F., Wolber, D., & Medlock-Walton, P. (2014, July). The design of naming features in App Inventor 2. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 129–132). IEEE.

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC'15)* (pp. 199–208).

Williams, C., Stanisstreet, M., Spall, K., Boyes, E., & Dickson, D. (2003). Why aren't secondary students interested in physics? *Physics Education, 38*(4), 324.

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

Xie, B., Shabir, I., & Abelson, H. (2015, October). Measuring the usability and capability of app inventor to create mobile applications. In *Proceedings of the 3rd International Workshop on Programming for Mobile and Touch* (pp. 1–8). ACM.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Part II
Student Competency and Assessment

Chapter 4
Measuring Secondary School Students' Competence in Computational Thinking in ICILS 2018—Challenges, Concepts, and Potential Implications for School Systems Around the World



Birgit Eickelmann

Abstract Focusing on increasing relevance of researching teaching and learning computational thinking, this chapter elaborates on the international study ICILS 2018 (*International Computer and Information Literacy Study*, second cycle). In the scope of this international comparative study, a research module on computational thinking is, for the first time, being realized as an international option. Countries with education systems which are taking part in ICILS 2018 were able to choose whether they wanted to take part in this additional module. The option comprises computer-based tests, two test modules for each student, in the domain of computational thinking for Grade 8 students as well as additional computational-thinking-related questions and items in the study's questionnaires for students, teachers, school principals, and IT coordinators. This chapter introduces the research approach of the study and especially its approach to computational thinking from the perspective of educational school-related research. Since findings of the study will not be available until the end of 2019, the current chapter illustrates the study's theoretical and empirical approach and outlines what kind of results will for the first time feature within the scope of an international large-scale assessment. With regard to the aim of the study to provide, apart from basic research knowledge toward an in-depth understanding of computational thinking, information on the current situation and future perspectives of education systems around the world, examples of potential implications for schools and school systems will also be given.

Keywords Computational thinking · ICILS 2018 · Students' competences · International comparison · Implications for school systems

B. Eickelmann (✉)
Institute for Educational Science, Paderborn University, Paderborn, Germany
e-mail: birgit.eickelmann@upb.de

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_4

www.dbooks.org

4.1 Introduction: The Relevance of Researching Teaching and Learning Computational Thinking in Schools

Comparatively new in the discussion about what kind of competences young people need in order to participate effectively in the digital society and to be prepared for work as well as for everyday life, competences related to computational thinking are attracting increasing attention. The corresponding research can be allocated in a broader understanding of researching ICT literacy (e.g., Ainley, Schulz, & Fraillon, 2016; Siddiq, Hatlevik, Olsen, & Throndsen, 2016; ETS, 2007). From this perspective, computational thinking adds to a new understanding of computer-related problem-solving. It not only broadens the previous definitions of ICT literacy but indeed opens up a new perspective. In this scope, teaching and learning about how to solve problems and how computer systems work means competence in computational thinking can be applied in different contexts (Ainley, Schulz, & Fraillon, 2016). Accordingly, the current discussion grows around the question as to where these competences should and could be taught. Answering this question requires developing an understanding of teaching and learning computational thinking and establishing common ideas of computational concepts, practices, and perspectives within a school system (Kong, 2016). Currently, computational thinking challenges the work of education systems all over the world, especially with regard to the development of competence models, teacher education, and curriculum integration of computational thinking (Kafai, 2016; Bescherer & Fest, 2018). Looking at the current developments, three approaches to support the acquisition of competences in computational thinking and improve students' achievement in computational thinking can be identified: (1) *Cross-curricular approach*: The first approach is to understand computational thinking as a cross-curricular competence which can be taught within different subjects, acknowledging that each subject has a particular view and contribution to students' acquisition of competences in computational thinking (Barr & Stephenson, 2011). (2) *Computer science approach*: The second discussion refers to the understanding of computational thinking as being a substantial part of computer science (e.g., Kong, 2016). In this understanding, thought processes related to computational thinking and involved in formulating and solving problems can be represented as computational steps and algorithms (Aho, 2011). This, to a certain extent, leads to promoting computational thinking best in the context of teaching computer science and is highly correlated with its contribution to modeling, programming, and robotics. Following on from this school of thought, Kong (2016) proposes a computational thinking framework, based on a framework by Brennan and Resnick (2012), to develop a curriculum in K-12 that promotes computational thinking through programming. The special thing about this framework is that although it is assumed that computational thinking draws "on the fundamental knowledge and skills of computer science" (Kong, 2016, p. 379), it is supposed that computational thinking is broader than computer science and refers to problem-solving, system design, and human behavior. Based on this, computational thinking is promoted in a separate 3-year curriculum. This results in a third approach, which is to develop a new learn-

ing area in terms of a separate subject. (3) *Computational thinking as an individual subject/learning area*: From this perspective, computational thinking is seen as a key competence in the field of using new technologies competently and reflectively. This understanding can be elaborated on and realized in different ways (Rich & Hodges, 2017). A number of countries have already implemented a computational thinking learning area either as a compulsory subject (e.g., in England, Department of Education, 2013) or as part of an optional subject (e.g., Denmark, EMU, 2017).

Regardless of the approach taken, including computational thinking in formal educational learning is already being realized, has been embarked on or is planned, while knowledge about the nature of computational thinking and factors and conditions related to its acquisition in the school context is still pending. This kind of knowledge can be understood as meta-knowledge, which is also important for improving learning in schools and for the development of school systems, the latter especially when it comes to ICT-related educational policies. In this understanding, research knowledge related to computational thinking, its implementation in schools and school systems, and the effectiveness of different approaches seems to be crucial in order for decision-making to bring educational systems into the digital age (Eickelmann, 2018). The urgent questions arising from this are what exactly should be taught in schools in future in the context of computational thinking and which conditions in classrooms, schools, and school systems are supportive in terms of facilitating and contributing to students' acquisition of competences in the field of computational thinking. Against this background, the international comparative study ICILS 2018 (International Computer and Information Literacy Study, 2nd cycle), which is for the first time closing the aforementioned research gap on the basis of representative data from school systems from all over the world, is introduced below.

4.2 Researching Students' Achievement in Computational Thinking in the Context of ICILS 2018

In the following section, the theoretical framework and the empirical approach of measuring students' achievements in computational thinking in the context of the international comparative large-scale study ICILS 2018 will be presented. The first section provides information on the study and its scope. The second section describes the approach of researching computational thinking in the context of the study. It provides information on the research design, on the understanding of computational thinking in the scope of the study as a construct that can be measured by conducting computer-based tests. Furthermore, the research questions that are addressed in the study ICILS 2018 as well as information on the relevant context factors are presented. The last subsection deals with insights into national extensions implemented by individual countries participating in the computational thinking option of ICILS 2018.

www.dbooks.org

4.2.1 *ICILS 2018—Assessing Students' Readiness for the Digital World in the Scope of an International Comparative Study*

With ICILS 2018 (*International Computer and Information Literacy Study*), the IEA (*International Association for the Evaluation of Educational Achievement*) is completing the second cycle of ICILS. The study is an international comparative assessment with participating countries from all around the world. As for ICILS 2013, the international study center of ICILS 2018 is allocated at the *Australian Council for Educational Research* (ACER). ICILS 2013 for the first time focused on computer and information literacy (CIL) as a competence area measured in international comparisons by conducting computer-based student tests for Grade 8 in 21 education systems around the world (Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014). After having successfully run this first cycle of ICILS, the IEA decided to conduct a second cycle (ICILS 2018). Acknowledging the rapid changes affecting ICT in teaching and learning and the aspiration to conduct a future-oriented study, ACER suggested adding computational thinking as an extension of the study. The core and trend part of both ICILS 2013 and ICILS 2018 comprises student tests for CIL and questionnaires on teaching and learning with ICT and individual, classroom and school factors with regard to the acquisition of CIL. Within the scope of ICILS 2018, nine education systems (Denmark, France, Germany, Luxembourg, Portugal, the U.S., the Republic of Korea and the benchmarking participants Moscow (Russia) and the German federal state North Rhine-Westphalia) are making use of the international option and are participating in the additional module focusing on computational thinking. Each student of the representative student sample takes, in addition to two 30-min CIL tests, two 25-min computer-based tests on computational thinking. From the research perspective, the development of the computer-based tests, covering all aspects of computational thinking and making it work for Grade 8 students, has probably been the most challenging part of the current cycle of the study. The aforementioned student tests are complemented by questionnaires addressing the tested student, teachers in the participating schools and the school principals and ICT coordinators of the schools which are selected for participation in the study. In this context, questionnaire items of particular interest in the context of computational thinking are added in the student and teacher questionnaires. Furthermore, all participating countries are asked to provide data about the education system and its approach to teaching and learning with ICT by filling in a so-called national context survey. This country-related questionnaire refers, for instance, to aspects of educational goals, curricula, and teacher education related to the scope of the study.

Data collection took place in spring 2018 for the Northern Hemisphere and in autumn 2018 for countries from the Southern Hemisphere. All education systems participated with a representative school sample, comprising representative teacher and student samples. Therefore, the results of the study allow for interpreting the status quo of Grade 8 student achievement in CIL and in addition, for those education

systems which are taking part in the international option, also for the domain of computational thinking.

4.2.2 Computational Thinking as Part of ICILS 2018

In the following, in-depth information is provided on the international option on computational thinking in the context of ICILS 2018. This includes the description of the theoretical understanding of computational thinking in terms of the definition of the construct used as a base for developing the student tests for ICILS 2018. Furthermore, initial insights into the questionnaires are given to provide examples of information on which factors and aspects are assumed to be relevant for student acquisition of computational thinking considering different perspectives.

4.2.2.1 The Construct of Computational Thinking and How Student Achievements in Computational Thinking Are Measured

The study starts where most research on computational thinking begins, basing itself on an adaptation of Wing's (2006) statements on computational thinking. In her understanding, computational thinking comprises fundamental skills which allow individuals to solve problems by using computers: "Computational thinking is a way humans solve problems; it is not trying to get humans to think like computers" (Wing, 2006, p. 35). Wing's idea relates to Papert (1980) who developed the essential features of computational thinking. In recent years, the field of computational thinking has been continuously developed (e.g., Dede, Mishra, & Voogt, 2013; Mannila et al., 2014; Voogt, Fisser, Good, Mishra, & Yadav, 2015).

In the context of ICILS 2018, computational thinking is defined as "an individual's ability to recognize aspects of real-world problems which are appropriate for computational formulation and to evaluate and develop algorithmic solutions to those problems so that the solutions could be operationalized with a computer" (Fraillon, Schulz, Friedman, & Duckworth, 2019). In this context, the understanding of computational thinking and its relevance for future generation lead to new tasks for schools and school systems in order to offer the possibility for every child to participate effectively in the digital world. In this context, it is stressed that this approach sees young people not only as consumers in a digital world but also their need for competence as reflective creators of content (IEA, 2016).

Apart from this broader understanding of computational thinking, the subject of the study is a more detailed definition of the construct "computational thinking." This has been developed by taking previous research findings, relevant approaches, and understandings of computational thinking into account. The study's understanding of computational thinking skills also corresponds to international standards such as the ISTE standards for students (2016). These standards focus on the understanding of "Computational Thinkers" that "students develop and employ strategies for

www.dbooks.org

understanding and solving problems in ways that leverage the power of technological methods to develop and test solutions" (p. 1), including skills such as problem formulation, data collection and analysis, abstraction, modeling, algorithmic thinking, solution finding, use of digital tools, representation of data, decomposition, and automation. The construct as it is addressed in ICILS 2018 consists of two strands which are both subdivided into subareas (Fraillon, Schulz, Friedman, & Duckworth, 2019).

Strand I: Conceptualizing problems: The first strand refers to the conceptualization of problems. Conceptualizing problems acknowledges that before solutions can be developed, problems must first be understood and framed in a way that allows algorithmic or system thinking to assist in the process of developing solutions. As subareas it includes three aspects: 1. Knowing about and understanding computer systems; 2. Formulating and analyzing problems; and 3. Collecting and representing relevant data. A task that provides evidence of an individual's ability to know about and understand computer systems includes, for example, operating a system to produce relevant data for analysis or explaining why simulations help to solve problems. Formulating problems entails the decomposition of a problem into smaller manageable parts and specifying and systematizing the characteristics of the task so that a computational solution can be developed—possibly with the help of a computer. Analyzing consists of making connections between the properties of and developing solutions to previously experienced problems and new problems to establish a conceptual framework to underpin the process of breaking down a large problem into a set of smaller, more manageable parts. Collecting and representing relevant data comprises making effective judgements about problem-solving within systems. This requires knowledge and understanding of the characteristics of the relevant data and of the mechanisms available for collection, organization, and representation of the data for analysis. This could, for instance, involve creating or using a simulation of a complex system to produce data that may show specific patterns or characteristics.

Strand II: Operationalizing solutions: The second strand concerns operationalizing solutions. Operationalizing solutions comprise the processes associated with creating, implementing, and evaluating computer-based system responses to real-world problems. It includes the iterative processes of planning for, implementing, testing, and evaluating algorithmic solutions to real-world problems. The strand includes an understanding of the needs of users and their likely interaction with the system under development. The strand comprises two aspects: 1. Planning and evaluating solutions and 2. Developing algorithms, programs, and interfaces. Examples of tasks that, for instance, provide evidence of an individual's ability to develop algorithms, programs, and designs can be processed such as creating a simple algorithm or modifying an existing algorithm for a new purpose.

This understanding of computational thinking acted as a basis for the development of the student tests. Each aspect is covered in at least one of the two computational thinking test modules. Student test data is analyzed using IRT scaling and student achievement data is analyzed in relationship to the context data which is gathered via the various types of questionnaires. The analyses are guided by the research questions, which are presented in the following section.

4.2.2.2 Research Questions Related to Computational Thinking in the Context of ICILS 2018

Taking the aforementioned research gap and the aims of the study into account, the following three overarching research questions are addressed in the study. The questions refer to different levels within education systems: the school system level, the school and classroom level, and the individual student level.

(1) *First, what variations exist in and across different countries in student achievement in computational thinking and what aspects of students' personal and social background are related to it?*

This question is answered by gathering data on student achievement in computational thinking, using computer-based computational thinking tests. The student test data enables compilation of national averages as well as for comparison of student achievement between countries. As in other international comparative studies, the student achievement data also allows for in-depth analyses within countries, e.g., comparing student achievement between boys and girls, between students with and without migration background or between students from different socioeconomic backgrounds. If countries have chosen to stratify the student sample to differentiate between different school types or tracks, analysis can also reveal differences and similarities between groups of students from different school types. These are just a few examples for potential analysis that are useful for the purpose of obtaining information on student achievement in computational thinking and gathering data and information to describe efforts made in teaching and learning computational thinking.

(2) *Second, what aspects of education systems, schools, and classroom practice explain variation in student achievement in computational thinking?*

This question focuses on the way in which computational thinking is implemented in education systems, in schools, and in classroom practice. Data relevant to this question will be collected via the questionnaires. This type of data and results, for instance, enable interpretation and framing of findings to address the first question. Furthermore, information on educational practice and settings is gathered and provides interesting insights into teaching and learning with ICT in the context of computational thinking.

(3) *Third, how is student achievement in computational thinking related to their computer and information literacy (CIL) and to their self-reported proficiency in using computers?*

This research question connects the core part of the study referring to CIL and the additional part referring to computational thinking. By applying both the student test on computational thinking and the student test on CIL within the same student sample, initial correlations between the two constructs can be examined (For a detailed overview of the underlying construct of CIL in ICILS 2013, see Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014. The exact constructs of CT and CIL in ICILS

www.dbooks.org

2018 can be found in the study's framework. It also provides information on how the study envisages the differences between these two areas). This also leads to new fundamental theoretical knowledge as well as to important information on how teaching of both competences can be combined and how student learning might be better supported.

A comprehensive and more detailed list of research questions as well as further details on instrument preparation and content can be found in the study's assessment framework (Fraillon, Schulz, Friedman, & Duckworth, 2019). A detailed overview of all instruments and objects will be published by ACER (*Australian Council for Educational Research*) in 2020 with the so-called ICILS 2018 user guide for the international database.

4.2.2.3 Insights into the Structure and Content of the Study's Instruments

As mentioned above, the ICILS 2018 applies computer-based student tests for Grade 8 students and adds questionnaires for students, teachers, school principals, and IT coordinators. Furthermore, a so-called national context survey questionnaire is applied and filled in by the national study centers of the countries participating in the study.

In the scope of the study, assessing students' achievement in computational thinking by applying computer-based tests means developing tests that have an authentic real-world focus to capture students' imagination in an appropriate way. At the core of the test modules "authoring tasks" contain authentic computer software applications (Fraillon, Schulz, & Ainley, 2013). The actual content of the test modules themselves will be published in the report of the study in 2019. Additionally, with the aim of exploring classroom practices regarding student use of computational thinking tasks, ICILS 2018 gathers information from students via student questionnaires. Parts of the questionnaires relate to computational thinking and are only applied in those countries that have chosen to include the computational thinking module. Students should, for example, specify the extent to which they have learned different computational thinking tasks in school. These tasks refer to the study's computational thinking construct described above. Furthermore, with respect to teacher attitudes toward teaching computational thinking, teachers are asked about the value they attach to teaching skills in the field of computational thinking. These skills also refer to the abovementioned computational thinking construct and are part of the teacher questionnaires. It is of note that the experts from the national centers together with the international study center, located at ACER in Melbourne, decided to include teachers from all different school subjects. Going beyond only involving computer science teachers has been a consensual decision of the experts included in the development of the instruments of the study. Based on this decision, the study will allow for comparing teachers practice as well as attitudes between different subject areas.

4.2.2.4 Making Full Use of the International Option: National Extension Toward Computational Thinking

According to the current relevance of computational thinking, education systems in several countries are applying national extensions. In Denmark, for example, the Danish national research center for ICILS 2018 is applying an additional quantitative study addressing school principals (Caeli & Bundsgaard, 2018). This extension aims to examine how schools already support or plan to support competences in the field of computational thinking by realizing the Danish new curriculum, which was implemented in 2017 and gives schools the opportunity to teach "technology understanding" (EMU, 2017). In Germany, several additions are conducted, including adding reading tests and tests on cognitive abilities (Eickelmann, 2017; Heller & Perleth, 2000). A closer examination of the third research question, in particular, the relation between student achievement in computational thinking and their self-reported proficiency in using computers, suggests that their self-reported proficiency of using computational thinking tasks also be taken into account. Therefore, the latter will also be added as a national extension in Germany. Furthermore, items focusing on computer science and its practice in schools are added as well as items aiming to examine the correlation between computational thinking and general problem-solving skills (Labusch & Eickelmann, 2017; see also Chap. 5 by Labusch, Eickelmann, & Vennemann, in this book). With regard to the computer science part, the German national consortium of the study was expanded to include a leading expert in the field of computer science and researching computer science in schools.

4.3 Relevance and Potential Outcomes for Educational Systems Around the World

Considering the increasing relevance of the next generation's competences in the field of computational thinking, several educational systems around the world have already decided to implement computational thinking as an obligatory subject into school curricula. Despite the fact that the approaches taken in implementing computational thinking may differ between countries, it becomes clear that supporting computational thinking processes and competences is considered as a future-oriented part of school education and adds to the traditional subjects and learning areas (Labusch & Eickelmann, 2017). Developing an understanding of computational thinking that leads to facilitating teaching it in schools, however, seems to be challenging. While various perspectives on computational thinking and its relevance for school learning exist, there is very limited availability of empirical knowledge based on a sound database. In addition to many studies currently being conducted around the world, the international comparative large-scale assessment ICILS 2018 for the first time provides empirical findings on student achievement in computational thinking in different education systems. The addition of questionnaire data to the results of the

www.dbooks.org

computer-based student test information about the incorporation of computational thinking in teaching and learning in classrooms as well as about school curricula enables data to be gathered on teachers' attitudes and students' background. In terms of research, one of the main outcomes of ICILS 2018 is the development of a theoretical model explaining student competences. Furthermore, the development of a sound understanding of the construct as well as bringing forward an empirical-based competence model differentiating between different competence levels is to be acknowledged and will be included in the report on the results of the study. For education systems—countries or regions—participating in the computational thinking part of ICILS 2018, which is optional for countries participating in the study, in-depth information on Grade 8 student achievement in computational thinking, the relevance of school and individual factors, such as gender, migration background, or socioeconomic background will be available. Beyond findings at the national and regional levels, the added value of the study lies in the international comparison and the opportunity for education systems to learn from one another.

In summary, it can be stated that the study will close a research and knowledge gap in the field of teaching and learning computational thinking and its implementation in schools and school systems. Apart from reflecting the status quo of student achievement in computational thinking, with publication of reporting on the study by the end of 2019, the future direction for education systems can be drawn from the study's national and cross-national results. Furthermore, the study provides a starting point for developing computational thinking competence in students on the basis of an international comparative approach. Applying and realizing an international approach underpins the importance of the research field related to computational thinking.

References

Aho, A. V. (2011). Computation and computational thinking. *Computer Journal, 55*(7), 832–835.

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.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer education science community? *ACM Inroads, 2*(1), 48–54.

Bescherer, C., & Fest, A. (2018). Computational thinking in primary schools: Theory and casual model. In A. Tatnall & M. Webb (Eds.), *Tomorrow's learning: Involving everyone.* IFIP advances in information and communication technology. Springer.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In A. F. Ball & C. A. Tyson (Eds.), *Proceedings of the 2012 Annual Meeting of the American Educational Research Association* (pp. 1–25). Vancouver: American Educational Research Association.

Caeli, E. N., & Bundsgaard, J. (2018). *Computational thinking initiatives in Danish grade 8 classes. A quantitative study of how students are taught to think computationally*. Paper presented at ECER 2018 (European Conference on Educational Research), Bolzano, Italy.

Dede, C., Mishra, P., & Voogt, J. (2013). *Advancing computational thinking in 21st century learning.* Presented at EDUsummIT. International Summit on ICT in Education, Dallas, TX.

Department of Education (2013). *National curriculum in England: Computing programmes of study*. Retrieved January 08, 2018, from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf.

Eickelmann, B. (2017). Computational Thinking als internationales Zusatzmodul zu ICILS 2018—Konzeptionierung und Perspektiven für die empirische Bildungsforschung. [Computational Thinking as an international option in ICILS 2018—The perspective of educational research] *Tertium Comparationis. Journal für International und Interkulturell Vergleichende Erziehungswissenschaft, 23*(1), 47–61.

Eickelmann, B. (2018). Cross-national policies on information and communication technology in primary and secondary schools—An international perspective. In J. Voogt, G. Knezek, R. Christensen & K.-W. Lai (Eds.), *International handbook of information technology in primary and secondary education*. Singapore: Springer.

EMU (2017). *Teknologiforståelse* [Technology understanding] *valgfag (forsøg) – Fælles Mål og læseplan* [Danish curriculum]. Retrieved January 08, 2018, from https://www.emu.dk/modul/teknologiforst%C3%A5else-valgfag-fors%C3%B8g-%E2%80%93-f%C3%A6lles-m%C3%A5l-og-l%C3%A6seplan.

ETS (Educational Testing Service) (2007). *Digital transformation: A framework for ICT literacy. A report of the international ICT literacy panel. Princeton: Center for Global Assessment.* Retrieved January 09, 2018, from https://www.ets.org/Media/Tests/Information_and_Communication_Technology_Literacy/ictreport.pdf.

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: Springer.

Fraillon, J., Schulz, W., & Ainley, J. (2013). *Assessment framework of ICILS 2013*. Amsterdam: IEA.

Fraillon, J., Schulz, W., Friedman, T., & Duckworth, D. (2019). *Assessment Framework of ICILS 2018.* IEA: Amsterdam.

Heller, K. A., & Perleth, C. (2000). *KFT 5–12 + R. Kognitiver Fähigkeitstest für 5. bis 12. Klassen, Revision* [cognitive abilities test for Grade 5 to 12 students]. Göttingen: Beltz.

IEA (2016). *The IEA's International Computer and Information Literacy Study (ICILS) 2018. What's next for IEA's ICILS in 2018?* Retrieved January 09, 2018, from http://www.iea.nl/sites/default/files/studies/IEA%20ICILS%202018%20Computational%20Thinking%20Leaflet.pdf.

ISTE (2016). *ISTE Standards for Students*. Retrieved April 03, 2018, from https://www.iste.org/standards/for-students.

Kafai, Y. (2016). From computational thinking to computational participation in K–12 education. Seeking to reframe computational thinking as computational participation. *Communications of the ACM, 59*(8), 26–27.

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.

Labusch, A., & Eickelmann, B. (2017). Computational thinking as a key competence—A research concept. In S. C. Kong, J. Sheldon & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education 2017* (pp. 103–106). Hong Kong: The Education University of Hong Kong.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). *Computational thinking in K-9 education.* Presented at Conference on Innovation & technology in computer science education, Uppsala.

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

Rich, P., & Hodges, C. B. (Eds.) (2017). *Emerging research, practice, and policy on computational thinking*. Springer Publishing Company.

Siddiq, F., Hatlevik, O. E., Olsen, R. V., & Throndsen, I. (2016). Taking a future perspective by learning from the past—A systematic review of assessment instruments that aim to measure primary and secondary school students' ICT literacy. *Educational Research Review, 19*(1), 58–84.

www.dbooks.org

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.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 5
Computational Thinking Processes and Their Congruence with Problem-Solving and Information Processing



A. Labusch, B. Eickelmann and M. Vennemann

Abstract Computational thinking is emerging as twenty-first century's key competence, especially for today's students as the new generation of digital learners. The acquisition of this competence requires in-depth knowledge in learning and computational thinking processes (Labusch & Eickelmann, 2017), both of which feature in various frameworks (e.g., Kong et al., 2017). Problem-solving theories can also be taken into account to obtain a more precise analysis of these processes since previous research has suggested a large degree of congruence between problem-solving and computational thinking (e.g., Yadav, Stephenson, & Hong, 2017b; Wing, 2008). In this contribution, to analyze computational thinking processes, computational thinking will be first considered in a general problem-solving context and then in an information processing context. Research of this kind is of immense importance to pave the way for further computational thinking in schools. In this regard, the approach provides the theoretical grounding for a national extension (Labusch & Eickelmann, 2017) to the *International Computer and Information Literacy Study 2018* (ICILS 2018) (Eickelmann, 2017; Fraillon, Schulz, Friedman, & Duckworth, 2019) which enables for the first time the cross-national measurement of student achievement in computational thinking. This contribution, therefore, serves as a theoretical basis for understanding computational thinking processes regarding problem-solving and cognitive information processing.

Keywords Computational thinking · Problem-solving · Information processing · ICILS 2018

A. Labusch (✉) · B. Eickelmann · M. Vennemann
Institute for Educational Science, Paderborn University, Paderborn, Germany
e-mail: amelie.labusch@upb.de

B. Eickelmann
e-mail: birgit.eickelmann@upb.de

M. Vennemann
e-mail: mario.vennemann@upb.de

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_5

www.dbooks.org

5.1 Introduction

Recent studies consider computational thinking to be an important competence of the twenty-first century (e.g., Voogt et al., 2015) that "everyone, not just computer scientists, would be eager to learn" (Wing, 2006, p. 33). Given that computational thinking is an important skill that every student needs to a certain extent for his or her future professional and personal life, the very first question is whether he or she really possesses this skill. In order to clarify this question, it is necessary to carry out an appropriate study. This, in turn, requires a framework that includes the definition and conceptualization of computational thinking. Common sense would suggest that computational thinking can be described as a way of thinking and of solving problems in a modern and technology-filled world (Curzon & McOwan, 2017). However, agreement usually ends when it comes to the question of what constitutes the core elements of computational thinking. The knowledge of the core elements and the associated skills that a student should possess is indispensable in order to be able to measure them at all and for a definition and conceptualization of computational thinking. While researchers have considered several definitions in their endeavors to specify computational thinking, no generally accepted definition has yet been established (Mannila et al., 2014). This also makes assessing it a challenge (Brennan & Resnick, 2012). Voogt et al. (2015) argue that trying to define computational thinking by setting conditions that it has to fulfill makes it "difficult to implement in practice" (p. 718). The role of education and especially of schools (e.g., as a set of transferable, cross-curricular skills) is also a topic for discussion in this context (Fraillon et al., 2019). It raises new tasks and challenges for schools. However, some experts have already developed frameworks (e.g., Brennan & Resnick, 2012; Kong et al., 2017) "to nurture students' ability to see the world through a computational lens" (Kong et al., 2017, p. 84). These frameworks include discipline-specific aspects of programming tasks that can serve as a basis for curricula. While it is true that computational thinking does have a close affinity to computer science (Curzon & McOwan, 2017), it does not necessarily follow that it is solely confined to this area. Indeed, Voogt et al. (2015) argue that the impression that computational thinking can be equated to programming that comes from the fact that many studies or discussions consider programming as its context.

Grover and Pea (2013) emphasize the need to consider whether all students should learn computational thinking, "including those who allege no interest in pursuing CS [computer science] and STEM [science, technology, engineering and mathematics] careers" (p. 40). However, their assertion recognizes the deeper problem, namely that the default way of thinking about computational thinking assigns it solely to computer science or STEM subjects. In fact, everyone should be given the opportunity to gain competences in the field of computational thinking (Wing, 2017) to allow them to successfully participate in a digitalized world. This basically confirms the argument put forward by Wing (2006), who adds weight to the argument that computational thinking is of central importance for all sciences, not only for computer science (Yadav et al., 2017a).

However, this poses major challenges for schools and teachers, because then noncomputer science teachers must also have skills that allow them to teach computational thinking adequately. However, it turns out that schools are often already "struggling with the way in which to use information technology in the classroom" (Faber, Wierdsma, Doornbos, & van der Ven, 2017, p. 13). Faber et al. (2017) suggest "unplugged" lessons in computational thinking. When discussing the question of what unplugged activities of computational thinking might look like, based on existing definitions, problem-solving, and cognitive abilities are important aspects. Taking a large congruence for granted gives researchers the added advantage of being able to resort to problem-solving and cognitive abilities theories. Problem-solving, especially complex problem-solving, has been explored for decades (Fischer, Greiff, & Funke, 2017), thus providing a sound body of well-proven research.

Assuming that the study will show that not all students possess this competence equally and to a sufficient degree, the question also arises as to which influencing factors can explain differences in students' achievement in computational thinking. Personal and social factors can be taken into account, but when it comes to learning and teaching computational thinking, the question of the role of schools and especially teachers arises. Therefore, the school context should not be disregarded as a possible influence variable on students' achievement of computational thinking. It has been a controversial issue in the recent discourse on the implementation of computational thinking into curricula of how to learn and teach it (Grover, 2017). Some experts argue that knowledge of what constitutes computational thinking and how to make full use of it makes it accessible (Barr, Harrison, & Conery, 2011). From this perspective, it is assumed that understanding computational thinking and its core elements "would enable teachers to integrate it in their teaching concepts" (Labusch & Eickelmann, 2017, p. 103).

5.2 Current State of Research

In summary, core elements of computational thinking should be identified in order to obtain a definition and conceptualization of computational thinking in the course of a framework for a study that measures whether students have the necessary competences. This is done in the following sections (5.2 and following) when considering the core elements, which are adopted within the framework and the study based on it. The study has been described in detail in the previous chapter (see Chap. 4 by Eickelmann in this book) and is mentioned again in this chapter in 5.3.1 and extended by an explanation of a national extension. This is followed by considering computational thinking in a general problem-solving context (5.2.1) and then in an information processing context (5.2.2) to show how these, as "unplugged aspects" of computational thinking, are related to its processes. Subsequently, computational thinking processes can be further analyzed (5.2.3). Finally, it is considered which

www.dbooks.org

aspects of the school context are to be taken into account in the context of the in school acquisition of competences in the field of computational thinking (5.2.4) to be able to explain variation in students' achievement in computational thinking.

5.2.1 Computational Thinking and Problem-Solving

In recent years, computational thinking has often been mentioned in the problem-solving context (e.g., Korkmaz, Çakir, & Özden, 2017; Román-González, Pérez-González, & Jiménez-Fernéndez, 2017). Problem-solving is described as a transformation from an undesirable initial state to a desirable final state (Beecher, 2017) by overcoming a barrier. Doing so means operating at high levels of thinking and reasoning (Spector & Park, 2012). Binkley et al. (2012) turn their attention to the challenge of teaching additional competences such as sophisticated thinking and flexible problem-solving to put students in the best possible starting position for participation in work, life, and society.

While the acquisition of problem-solving skills is a current topic of discussion, studies into problem-solving skills and problem-solving processes have been being conducted for decades. A problem-solving process is frequently described as a seven-stage cycle (Pretz, Naples, & Sternberg, 2003): (1) the recognition or identification of a problem, (2) the definition and mental representation of the problem, (3) the development of a strategy to solve the problem, (4) the organization of knowledge concerning the problem, (5) the allocation of mental and physical resources to solving the problem, (6) the monitoring of progress toward the goal, and (7) the evaluation of the solution for accuracy. Although there might be several ways to solve a problem—and the one chosen will depend on the nature of the actual problem in hand—it is assumed that they will all have the same processes in common. The first stage of the problem-solving process is therefore of huge importance: The problem needs to be both recognized and identified in order to select the best possible solution since how a problem is solved depends on the actual problem itself. Since these processes are very similar to computational thinking processes, it is important to investigate their similarities and differences by exploring the congruence between general problem-solving and computational thinking.

5.2.2 Computational Thinking and Information Processing

Many researchers and educators contend that computational thinking changes our way of thinking (Bundy, 2007). Information processing theories and models offer deeper insights into this subject: ACT-R (Adaptive Control of Thought—Revised) (Anderson, 2010) and MEKIV (Model for elementary and complex human information processing) (Hussy, 1998), for instance, are two global models that incorporate a useful cognitive architecture for modeling complex problem-solving processes.

They both have in common that they try to integrate multiple mental functions and thus take a more realistic approach to the complexity of action control and thinking. While ACT-R has a symbolic structure and can be used to create cognitive models that simulate human cognitive processes (Hussy, 1998), MEKIV provides a more theoretical description of elementary and complex human information processing, starting with the sensory organs and ending in the motoric system. All information stored in the sensory register is passed on to the long-term memory, where the incoming pieces of information are compared with the stored patterns that have meaning. Stimuli received via sensory organs are thus encoded and interpreted by translating them into cognitive representations, which are then stored in memory. In doing so, the stimulus is correlated with existing knowledge, whereby it is enriched with information. Prerequisites for this are the accessibility and retrievability of stored knowledge. The process of categorizing the stimulus is of particular importance when it comes to encoding. The encoded perception is itself stored in memory and creates new memory contents in interaction with the knowledge already stored there. Human behavior then represents the output of this information processing (Bless, Fiedler, & Strack, 2004).

5.2.3 Computational Thinking Processes

When computational thinking processes are compared with problem-solving processes and information processing, a closer inspection reveals a high degree of similarity. Accordingly, this contribution focuses on the following definition: "In computational thinking, the students demonstrate the ability to identify a problem, break it down into manageable steps, work out the important details or patterns, shape possible solutions and present these solutions in a way that a computer, a human, or both, can understand. Computational thinking can also involve structuring and manipulating data sets to support the solution process" (IEA, 2016, p. 1). This definition includes problem identification and formulation, a breakdown process (decomposition), the process of pattern recognition and pattern matching, the shaping of a possible solution with several processes such as abstraction, evaluation, and debugging, as well as algorithmic thinking by presenting solutions. It tells a lot about the cognitive structure of computational thinking, containing a structured, conceptual, and strategic way of thinking and solving problems. The computational thinking processes described in the following are all related to the aforementioned definition.

5.2.3.1 Problem Identification and Problem Definition

The chosen definition is based on a computational thinking assessment framework that contains the two strands of conceptualizing problems and operationalizing solutions (Fraillon et al., 2019). The problem-solving process cycle outlined above

www.dbooks.org

includes both the identification and the definition of problems. That means that problems have to be conceptualized in understanding and framed before solutions can be formed (ibid.). Identifying the problem also permits a decision on whether it can be solved using computational thinking. Understanding the problem to be solved is an important prerequisite for the problem-solving process. Pólya (2004) points out that "it is foolish to answer a question that you do not understand" (p. 6). While a scientific investigation process begins with a hypothesis (Riley & Hunt, 2014), a problem-solving process begins with a problem definition. Michaelson (2015) maintains that characterizing the problem is the most challenging part of problem-solving. When a person presents a problem to someone else and asks them to solve it (e.g., in school, when a teacher asks her/his students to solve a problem), it is best if they formulate the problem in their own words (Pólya, 2004). While it is not the rule that computing activities begin with problem identification and definition or are always thought in terms of problems to be solved, according to the above definition, problem identification and problem definition may be included in computational thinking processes.

5.2.3.2 Decomposition and Data Collection, Analysis, and Representation

Miller (1956) has found that the human memory is limited to 7 ± 2 items, which means that some problems are too complex for the human brain to solve unless they are first decomposed into subproblems to be processed in the brain. This process of breaking a problem "down into manageable steps" (IEA, 2016, p. 1) is called decomposition. It is a core element of a computational thinking process and ensures that even complex problems can be understood. While some computational thinking processes belong to only one strand (conceptualizing problems or operationalizing solutions), others—like pattern recognition and pattern matching—are relevant to almost all aspects of a computational thinking process. Decomposing a problem by analyzing it as a whole also requires knowledge of what constitutes manageable steps and how they are related to each other and the whole problem. The decomposition process is well known in language arts, where it is referred to as outlining (Barr & Stephenson, 2011), i.e., organizing work by decomposing it into its main ideas (Riley & Hunt, 2014). Data can support the computational thinking process by providing important information. This means finding a data source, analyzing the data, and using data structures to represent the data (Barr & Stephenson, 2011).

5.2.3.3 Modeling and Revising Solutions

The problem fragments of the decomposition process are shaped into possible solutions by modeling and revising them. A model can serve as an abstract representation of a real-world situation (Frigg, 2002), which is why a modeled solution might also be applied to real-world problems. Modeling a solution involves different processes and

is also part of a problem-solving process. Pattern recognition and pattern matching processes also form part of information processing. They mean finding similarities and differences and accessing something known. Existing knowledge is used to categorize the problem fragments. A pattern indicates whether or not a particular string is part of a category or class defined by the pattern (Riley & Hunt, 2014). Pattern recognition methods also find application in other domains, e.g., medicine, where they are used to recognize patterns in DNA strands (Fink, 2014). In problem-solving processes, pattern matching saves work, since findings can be transferred to a similar problem so that this does not have to be tackled from the very beginning (Curzon & McOwan, 2017).

The categorized problem fragments are then generalized by eliminating unimportant details (Barr & Stephenson, 2011) in an abstraction process based on inductive logic. Inductive logic is based on the principle that general conclusions can be drawn from observations and experiences. Since these conclusions contain uncertainty, the degree of certainty depends on the number of observations/experiences. Inductive reasoning is the process of concluding individual cases into general rules. If fixed rules are consulted and formulated as premises from which conclusions are deduced, deductive reasoning takes place. This requires sufficient knowledge about these fixed rules and a need to organize knowledge regarding the problem. However, the main part of an abstraction process is the elimination of unimportant details and thus the focus on the essential. This allows a particular emphasis to be chosen and the simplification of complex structures. In social studies, abstraction can be a useful means of summarizing facts and using them to draw conclusions (Barr & Stephenson, 2011).

Even if the chosen definition does not actually reflect this in detail, a computational thinking process, like a problem-solving process, contains an evaluation component (Fraillon et al., 2019). During the process, but above all after it has been shaped, the solution has to be tested to make sure that the previously formulated goal can be achieved. Debugging is important here in order to test the shaped solution. Murphy et al. (2008) identify various different debugging strategies, including testing and pattern matching. They recognize pattern recognition and pattern matching processes through the students' intuitive observation and action. Riley and Hunt (2014), in turn, find similarities between a debugging process and the proof of a mathematical theorem. In the education context, debugging also provides a deeper understanding of the problem-solving behavior of students (Liu, Zhi, Hicks, & Barnes, 2017).

5.2.3.4 Algorithmic Thinking

Denning (2009) suggests that the basic idea behind computational thinking is in essence algorithmic thinking. In the field of computer science, an algorithm is defined as "any well-defined sequence of actions that takes a set of values as input and procedures some set of values as output" (Riley & Hunt, 2014, p. 130). An algorithmic view of life is assumed to be very valuable because it involves many essential activities in life by following simple and discrete steps. The thinking process required to formulate an algorithm differs in one aspect from the formulation of a well-defined rule of

www.dbooks.org

action: another form of language is needed. An algorithm run by a computer requires a programming language that ensures that an algorithm can only have one possible interpretation so that there is no misunderstanding (ibid.). While human language does not fulfill this requirement, the formulation of an action rule can be deemed algorithmic thinking when only the process itself is taken into consideration. The formulated solution is usually to be evaluated for correctness, efficiency, elegance, and usability (Beecher, 2017, p. 100). The advantage of algorithmic thinking is that the solution can be delegated to another person or a computer. Barr and Stephenson (2011) advocate the inclusion of algorithmic thinking in other subjects and not just in computer science. A good example here would be writing an instruction. Since it cannot be presupposed that students are familiar with a programming language, writing an instruction helps them to practice the logical reasoning used in writing an algorithm.

The above sections emphasize three things. First, computational thinking processes apply to processes in other domains. Second, there is a strong similarity between problem-solving processes and computational thinking processes, whereby it should be noted that the process of algorithmic thinking represents a clear extension of the pure solution formulation. Third, computational thinking processes can also be practiced and carried out in other subjects in school.

5.2.4 In-School Acquisition of Competences in the Field of Computational Thinking

The above-mentioned computational thinking processes describe the cognitive part of computational thinking. By demonstrating that five factors of personality correlate with computational thinking, Román-González et al. (2016) extend their finding that problem-solving is close to computational thinking. As a result, noncognitive factors that influence computational thinking at student level also need to be considered and investigated. Ramalingam, Labelle, and Wiedenbeck (2004) contend that programming experience influences programming self-efficacy. Since there is a kinship between programming and computational thinking, these results add weight to the assumption that the students' self-efficacy in computational thinking may have an impact on their acquisition of competences in the field of computational thinking. Another student-level variable that might have an impact on student achievement in computational thinking can include background characteristics (e.g., gender). Gender differences are evident in the programming field (Sullivan & Bers, 2016), and it can be assumed that differences are also found in the field of computational thinking.

Sanford and Naidu (2016) emphasize that computational thinking is not self-evident and calls for training and guidance. Indeed, many experts maintain that computational thinking should be integrated into teaching and learning. Focusing on students' individual needs, requirements, and learning environments is important because studying students' achievement in computational thinking sets them in the

context of different effects on the development of computational thinking. These effects can be classified as factual, attitudinal, and behavioral and take place on different levels (Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014).

5.3 Research Concept

An analysis of the above reveals the emergent need for a study which measures the in-school acquisition of computational thinking as a cross-curricular competence, using a research concept that integrates general problem-solving and school-related parameters. The findings indicate that while a congruence between computational thinking and problem-solving is often taken for granted, few scholars have actually investigated this assumption. There is also an absence of a study with a substantial database that explores the congruence between computational thinking and problem-solving as well as individual and school-related factors relating to the acquisition of competences in the field of computational thinking with the aim of creating a holistic picture of computational thinking in schools.

As a result, the following research questions are addressed within this contribution:

1. To what extent are students' achievements in computational thinking related to their general problem-solving skills?
2. To what extent are students' achievements in computational thinking related to their general problem-solving skills under control of individual characteristics?
3. To what extent are students' achievements in computational thinking related to their general problem-solving skills under control of the school context?

It is assumed that, as has already been shown theoretically, there is a strong positive relationship between students' achievement in computational thinking and their problem-solving skills (research question 1). However, it is also assumed that there are other variables at the student and school level influencing the relationship (research question 2 and 3). These assumptions have to be verified or rejected in a suitable study.

5.3.1 Study and Data Basis

The methods and instruments to be used for the analyzes will be aligned with the aforementioned research questions. The data basis is provided by the international comparative study IEA-ICILS 2018 (*International Computer and Information Literacy Study*), in which the authors are participating as members of the national study center in Germany. In ICILS 2013, students' computer and information literacy (CIL) was measured on a representative basis in an international comparison. In the course of the second cycle of ICILS in 2018, the IEA (*International Association for the Evaluation of Educational Achievement*) is for the first time implementing a new,

www.dbooks.org

additional option to assess "computational thinking" by applying computer-based student tests and questionnaires to determine in addition to other variables individual student background characteristics and school-related parameters (Fraillon et al., 2019). Some 20 countries are participating in ICILS 2018; those making use of the additional "computational thinking" option include the USA, France, South Korea, Germany, Denmark, Portugal, and Luxembourg.

The data will be gathered from representative samples, which will be designed—as in 2013—as two-stage cluster samples, first sampling schools and then selecting about 20 students, 15 teachers, 1 principal, and 1 ICT coordinator in each school. The German subsample in 2018 will compromise around 4,500 students. The ICILS 2018 student sample is drawn from Grade 8 students; the teacher population is defined as all teachers teaching Grade 8 at the sampled schools. Both student and teacher data will be used to answer the research questions. While the former will be used to identify students' achievement in different fields (e.g., computational thinking) and background data, the latter will serve to identify the school-related parameter "teachers' attitude to teaching computational thinking." After the main survey and data submission in 2018, the first results of the study will be presented in the national report in 2019.

To explore the congruence between students' achievement in computational thinking and their problem-solving skills (research question 1), both areas need to be tested. This is also the approach taken in the German national extension to ICILS 2018 (Labusch & Eickelmann, 2017), which uses computer-based tests to measure the achievement in computational thinking of Grade 8 students and in addition applies paper-based tests to assess their cognitive, problem-solving, pattern recognition, pattern matching, and debugging skills. The problem-solving processes worked out are used as a theoretical basis for the conceptual design of the problem-solving scale. A four-point Likert scale, ranging from "strongly agree" to "strongly disagree" is used to assess how well students solve problems and contains three subscales: problem-solving confidence (to assess self-perceived confidence in solving problems), approach-avoidance style (to assess whether the students tend to approach or avoid problems), and personal control (to assess elements of self-control). Since the German national extension to this study also collects data about students' computational thinking self-concepts, these can also be used to answer the underpinning research questions.

5.3.2 Methodology and Expected Outcomes

Since contextual factors influence variations in human cognition, affect and behavior, it will be needed to use, in addition to descriptive primary and secondary analyses, a multilevel statistical model to avoid problems of parameter misestimation and model mis-specification (Rowe, 2009). It is now widely accepted that studies that assess the impact of school education on students' learning outcomes must take into account the fact that students' progress is influenced by complex, multilevel,

multidimensional, and interrelated factors (ibid.). To examine the extent to which different factors at the student and school level are associated with variations in the relationship between students' achievement in computational thinking and their problem-solving skills, a multilevel structural equation model will be used. Factors of interest at student level include those related to students' general problem-solving skills, their basic cognitive abilities, their self-reported proficiency in computational thinking as well as their personal and home background. Aggregated at school level, the school-related use of computational thinking and teachers' attitudes to teaching computational thinking are two key factors of interest. The theoretical model has been developed using corresponding research literature to determine which influencing factors on students' achievement in computational thinking are to be taken into account for multivariate analyses. The types of results can be divided into three categories: First, those that illuminate the computational thinking construct and contribute to clarifying the congruence with general problem-solving; second, due to the extensive data basis that will be generated in ICILS 2018, additional student variables such as basic cognitive skills, self-concept, and background characteristics will also be examined; third, the school context can be included.

5.4 Summary and Outlook

For the first time, the congruence between computational thinking and general problem-solving will be examined under the control of other variables at student and school level on the basis of a representative sample in the context of a national extension of the IEA-study ICILS 2018. The data has been collected in 2018, and initial results will be available in 2019. Using a cognitive approach, these results might provide a starting point for implementing computational thinking into curricula as a multidisciplinary and cross-curricular key competence of the twenty-first century and contribute to moving school systems into the digital age.

References

Anderson, J. R. (2010). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Learning with Technology, 38*(6), 20–23.

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.

Beecher, K. (2017). *Computational Thinking: A beginner's guide to problem-solving and programming*. Swindon, UK: BCS Learning & Development Limited.

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. In P. Griffin, B. McGaw & E. Care (Eds.), *Assessment and teaching of 21st century skills* (pp. 17–66). Dordrecht: Springer.

www.dbooks.org

Bless, H., Fiedler, K., & Strack, F. (2004). *Social cognition. How individuals construct social reality* [*Social psychology*: *A modular course*]. Philadelphia, PA: Psychology Press.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Paper Presented at the Annual Meeting of the American Educational Research Association*, Vancouver, Canada.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67–69.

Curzon, P., & McOwan, P. W. (2017). *The power of computational thinking: Games, magic and puzzles to help you become a computational thinker*. London, UK: World Scientific Publishing Europe Ltd.

Denning, P. J. (2009). The profession of IT: Beyond computational thinking. *Communications of the ACM, 52*(6), 28–30.

Eickelmann, B. (2017). Computational Thinking als internationales Zusatzmodul zu ICILS 2018 – Konzeptionierung und Perspektiven für die empirische Bildungsforschung [Computation Thinking in the context of ICILS 2018 – the perspective of educational research]. *Tertium Comparationis. Journal für International und Interkulturell Vergleichende Erziehungswissenschaft, 23*(1), 47–61.

Faber, H. H., Wierdsma, M. D. M., Doornbos, R. P., & van der Ven, J. S. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network, 12,* 13–24.

Fink, G. A. (2014). *Markov models for pattern recognition. from theory to applications*. London: Springer.

Fischer, A., Greiff, S., & Funke, J. (2017). The history of complex problem solving. In B. Csapó & J. Funke (Eds.), *The nature of problem solving: Using research to inspire 21st century learning* (pp. 107–121). Paris: OECD Publishing.

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. In *International Report*. Amsterdam: International Association for the Evaluation of Educational Achievement (IEA).

Fraillon, J., Schulz, W., Friedman, T., & Duckworth, D. (2019). *Assessment framework of ICILS 2018*. Amsterdam: IEA.

Frigg, R. (2002). *Models and representation: Why structures are not enough. Measurement in physics and economics project discussion paper series*. London: London School of Economics.

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In P. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 269–288): Springer Publishing Company.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Hussy, W. (1998). *Denken und Problemlösen [Thinking and problem-solving]* (2nd ed.). Stuttgart: Kohlhammer.

IEA (2016). *The IEA's international computer and information literacy study (ICILS) 2018*. What's next for IEA's ICILS in 2018? Retrieved December 12, 2017, from http://www.iea.nl/fileadmin/user_upload/Studies/ICILS_2018/IEA_ICILS_2018_Computational_Thinking_Leaflet.pdf.

Kong, S. C., Abelson, H., Sheldon, J., Lao, A., Tissenbaum, M., Lai, M., Lang, K., & Lao, N. (2017). Curriculum activities to foster primary school students' computational practices in block-based programming environments. In S. C. Kong, J. Sheldon & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education 2017* (pp. 84–89). Hong Kong: The Education University of Hong Kong.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior, 72,* 558–569.

Labusch, A., & Eickelmann, B. (2017). Computational thinking as a key competence—A research concept. In S. C. Kong, J. Sheldon & K. Y. Li (Eds.), *Conference Proceedings of International*

Conference on Computational Thinking Education 2017 (pp. 103–106). Hong Kong: The Education University of Hong Kong.

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education, 27*(1), 1–29.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 1–29).

Michaelson, G. (2015). Teaching Programming with computational and informational thinking. *Journal of Pedagogic Development, 5*(1), 51–66.

Milller, G. A. (1956). The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological Review, 63*(2), 81–97.

Murphy, L., Lewandowski, G., McCauley, R. Simon, B., Thomas, L., & Zander, C. (2008). Debugging: The good, the bad, and the quirky—A qualitative analysis of novices' strategies. *SIGCSE Bulletin, 40*(1), 163–167.

Pólya, G. (2004). *How to solve it: A new aspect of mathematical method*. Princeton: Princeton University Press.

Pretz, J. E., Naples, A. J., & Sternberg, R. J. (2003). Recognizing, defining and representing problems. In J. E. Davidson & R. J. Sternberg (Eds.), *The psychology of problem solving* (pp. 3–30). Cambridge, UK: Cambridge University Press.

Ramalingam, V., Labelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *SIGCSE Bulletin Inroads, 36*(3), 171–175.

Riley, D. D., & Hunt, K. A. (2014). *Computational thinking for the modern problem solver*. Boca Raton, FL: CRC Press.

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernandez, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior, 72,* 678–691.

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 *Paper presented at the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, Salamanca, Spain.

Rowe, K. J. (2009). Structural equation modeling in educational research. In T. Teo & M. S. Khine (Eds.), *Structural equation modeling in educational research: concepts and applications* (pp. 201–239). Rotterdam, The Netherlands: Sense Publishers.

Sanford, J. F., & Naidu, J. T. (2016). Computational thinking concepts for grade school. *Contemporary Issues in Education Research, 9*(1), 23–32.

Spector, J. M., & Park, S. W. (2012). Argumentation, critical reasoning, and problem solving. In S. B. Fee & B. R. Belland (Eds.), *The role of criticism in understanding problem solving* (pp. 13–33). New York: Springer.

Sullivan, A., & Bers, M. U. (2016). Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. *Journal of Information Technology Education: Innovations in Practice, 15,* 145–165.

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.

Wing, J. M. (2006). Computational thinking. *Communication in 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.

Wing, J. M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology, 25*(2), 7–14.

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017a). Computational thinking in teacher education. In P. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer Publishing Company.

www.dbooks.org

Yadav, A., Stephenson, C., & Hong, H. (2017b). Computational thinking for teacher education. *Communications of the ACM, 60*(4), 55–62.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 6
Combining Assessment Tools for a Comprehensive Evaluation of Computational Thinking Interventions



Marcos Román-González, Jesús Moreno-León and Gregorio Robles

Abstract Given that computational thinking (CT) is still a blurry psychological construct, its assessment remains as a thorny, unresolved issue. Hence, in recent years, several assessment tools have been developed from different approaches and operational definitions of CT. However, very little research has been conducted to study whether these instruments provide convergent measurements, and how to combine them properly in educational settings. In response, we first review a myriad of CT assessment tools and classify them according to their evaluative approach. Second, we report the results of two convergent validity studies that involve three of these CT assessment tools, which come from different perspectives: the Computational Thinking Test, the Bebras Tasks, and Dr. Scratch. Finally, we propose a comprehensive model to evaluate the development of CT within educational scenarios and interventions, which includes the aforementioned and other reviewed assessment tools. Our comprehensive model intends to assess CT along every cognitive level of Bloom's taxonomy and throughout the various stages of typical educational interventions. Furthermore, the model explicitly indicates how to harmoniously combine the different types of CT assessment tools in order to give answer to the most common research questions in the field of CT Education. Thus, this contribution may lead scholars and policy-makers to perform accurate evaluation designs of CT according to their inquiry goals.

M. Román-González (✉)
Universidad Nacional de Educación a Distancia (UNED) – Facultad de Educación,
C/ Juan del Rosal, nº 14, Office 2.18, Madrid, Spain
e-mail: mroman@edu.uned.es

J. Moreno-León
Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado (INTEF),
C/ Torrelaguna, nº 58, Madrid, Spain
e-mail: jesus.morenol@educacion.gob.es

G. Robles
Universidad Rey Juan Carlos (URJC), ETSI Telecomunicación, Camino del Molino s/n,
Fuenlabrada, Madrid, Spain
e-mail: grex@gsyc.urjc.es

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_6

www.dbooks.org

Keywords Computational thinking · Assessment · Evaluation · Comprehensive model · Research designs

6.1 Introduction

In the last decade, computational thinking (CT) (Wing, 2006) has emerged as an umbrella term that refers to a broad set of problem-solving skills, which should be acquired by the new generations to thrive in our computer-based world (Bocconi et al., 2016). Thus, the use of the CT term has evolved and grown up, even without reaching a consensus about its definition (Kalelioglu, Gülbahar, & Kukul, 2016).

Moreover, the relation between CT and computer programming is blurry too. It is assumed that computer programming enables CT to come alive, and it is the main way to demonstrate CT skills (Lye & Koh, 2014). However, CT might be projected onto a wide range of tasks that do not involve programming (Wing, 2008). In other words, it is necessary to activate CT skills in order to program properly, but these skills could be used in other contexts that are disconnected from computer programming. Therefore, CT is a broader term than computer programming.

In a certain sense, the coining of CT as an umbrella term has been extremely useful, and its sudden success can be explained. First, the CT term has helped to place Computer Science Education beyond computer programming. Second, it has helped to lower the entry barriers to computer programming, in parallel of the appearance and rise of visual blocks languages; in the same vein, CT has provided the frame to focus not on the computer programming syntax, but on the underlying mental processes for it. As a result, CT is perceived as a friendly and nonthreatening term that has contributed to bring Computer Science (CS) closer to the teachers and to foster the *CS4all* movement. Third, CT is such a *liquid* term that it can be used more as an approach than as a concept; then, CT has enhanced the metaphor of "programming to learn" instead of "learning to program." In other words, CT has made possible to imagine a computational approach for any of the subjects of the curriculum. Finally, CT term has gathered not only cognitive skills, such as decomposition, pattern recognition, abstraction, and algorithmic design, but also noncognitive variables (Román-González, Pérez-González, Moreno-León, & Robles, 2018) and related soft skills such as persistence, self-confidence, tolerance to ambiguity, creativity, and teamwork, among others. In summary, CT term has served as a response to a global phenomenon in which it has become evident that our lives, increasingly mediated by algorithms, need a new set of skills to relate properly with the ubiquitous machines (Rushkoff, 2010; Sadin, 2015).

Nevertheless, this lack of definition of CT term that has proved useful in the past could be a burden for its future survival and development. Thus, there is not only a lack of consensus on a CT formal definition but also disagreements about how CT should be integrated into educational curricula (Lye & Koh, 2014), and especially on how it should be properly assessed (Grover, 2015; Grover & Pea, 2013). The latter is an extremely relevant and urgent topic to be addressed because without

reliable and valid assessment tools, CT will have difficulties to consolidate in the educational system and it runs a serious risk of disappearing as a construct worthy of consideration for educational psychology.

Expressed in a metaphorical way, CT term has had its naïve childhood, it has gone through an impulsive (and productive) adolescence, and it is now entering adulthood through a process of demystification. As Shute, Sun, and Asbell-Clarke (2017) say, CT is being demystified and, if it wants to survive, it is time to study it scientifically and to define operational CT models that can be empirically validated.

Ironically, although CT assessment seems to be the thorniest issue in the field, we consider that it brings the biggest opportunity to reinforce CT as a serious and well-established psychological construct. Assessment and measurement imply to operationally define the construct, CT in this case, in order to design an assessment tool that must be consequently validated. Hence, advances in assessment can contribute decisively to consolidate CT as a solid concept, a solid variable worthy of being studied and developed. In this sense, this chapter aims to review the current state-of-the-art CT assessment tools and to propose a comprehensive evaluation model, which could combine these tools effectively.

The chapter is structured as follows: in the next section, we review a myriad of CT assessment tools and classify them according to their evaluative approach. In the third section, we report the results of two convergent validity studies that involve three of these CT assessment tools, which come from different perspectives: the Computational Thinking Test, the Bebras Tasks, and Dr. Scratch. In the fourth section, we propose a comprehensive model to evaluate CT interventions within educational scenarios, which includes the aforementioned and other reviewed assessment tools. Finally, in the fifth and last section, we offer our conclusions and we speculate about future lines of research.

6.2 Computational Thinking Assessment Tools

Without being exhaustive, and focusing on K-12 education, we can find the following CT assessment tools, which can be classified depending on their evaluative approach:

- **CT diagnostic tools**: They are aimed at measuring the CT aptitudinal level of the subject. Their major advantage is that they can be administered in pure pretest condition (e.g., subjects without any prior programming experience). Complementarily, the diagnostic tools can be also applied in posttest condition (i.e., after an educational intervention) in order to check if the CT ability has increased. Some of the CT diagnostic tools are the *Computational Thinking Test* (Román-González, 2015; Román-González, Pérez-González, & Jiménez-Fernández, 2017b), the *Test for Measuring Basic Programming Abilities* (Mühling, Ruf, & Hubwieser, 2015), and the *Commutative Assessment Test* (Weintrop & Wilensky, 2015). All the aforementioned tests are aimed at middle-school and/or high-school students; for

www.dbooks.org

elementary-school students, the instrument developed by Chen et al. (2017) in the context of everyday reasoning and robotics programming can be used.

- **CT summative tools**: Their goal is to evaluate if the learner has achieved enough content knowledge (and/or if he is able to perform properly) after receiving some instruction (and/or training) in CT skills. Then, the main use of the CT summative tools is placed in the posttest condition. We can distinguish among several of these tools according to the learning environment used. Thus, we find the summative tools of Meerbaum-Salant, Armoni, and Ben-Ari (2013) in the Scratch context; the tool named *Quizly*[1] (Maiorana, Giordano, & Morelli, 2015) for assessing content knowledge in the context of App Inventor; the *Fairy Assessment* (Werner, Denner, Campe, & Kawamoto, 2012), a performance-based tool that runs in the Alice environment; or the summative tools used for measuring the students' understanding of computational concepts after a new computing curriculum is implemented (e.g., see Zur-Bargury, Pârv, & Lanzberg, 2013).
- **CT formative–iterative tools**: They are aimed at providing feedback to the learner, usually in an automatic way, in order to develop and improve his/her CT skills. Strictly speaking, these tools do not assess the individuals, but their learning products, usually programming projects. Therefore, these tools are mainly used during the learning process, and they are specifically designed for a particular programming environment. Thus, we find *Dr. Scratch* (Moreno-León, Robles, & Román-González, 2015) or *Ninja Code Village* (Ota, Morimoto, & Kato, 2016) for Scratch; *Code Master*[2] for App Inventor; and the *Computational Thinking Patterns CTP-Graph* (Koh, Basawapatna, Bennett, & Repenning, 2010) or *REACT* (Koh, Basawapatna, Nickerson, & Repenning, 2014) for AgentSheets.
- **CT data-mining tools**: These tools, like the previous ones, are focused on the learning process. Nevertheless, while the formative–iterative tools statically analyze the source code of the programming projects, the data-mining tools retrieve and record the learner activity in real time. These latter tools provide valuable data and learning analytics from which cognitive processes of the subject can be inferred, and they are especially useful to detect gaps and misconceptions while acquiring computational concepts. It can be highlighted the research done by the team of Shuchi Grover (Grover et al., 2017; Grover, Bienkowski, Niekrasz, & Hauswirth, 2016) in the Blockly environment, and the one from Eguiluz, Guenaga, Garaizar, and Olivares-Rodriguez (2017) using Kodetu.
- **CT skill transfer tools**: Their objective is to assess to what extent the students are able to transfer their CT skills onto different kinds of problems, contexts, and situations. Hence, we find the *Bebras Tasks* (Dagiene & Futschek, 2008), which are focused on measuring CT skills' transfer to real-life problems. We also find the *CTP-Quiz* (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011), which evaluates how CT skills are transferred to the context of scientific problems and simulations. Finally, the projection of CT skills onto kinesthetic tasks, and vice versa (i.e., embodied learning of CT), can be highlighted (Daily, Leonard, Jörg,

[1] http://appinventor.cs.trincoll.edu/csp/quizly/.

[2] http://apps.computacaonaescola.ufsc.br:8080/.

Babu, & Gundersen, 2014). This type of tools is especially suitable for assessing the degree of retention and transfer of CT, once a time has elapsed since the end of a CT educational intervention.

- **CT perceptions–attitudes scales**: They are aimed at assessing the perceptions (e.g., self-efficacy perceptions) and attitudes of the subjects not only about CT, but also about related issues such as computers, computer science, computer programming, or even digital literacy. Among scales targeted to students, we can name the *Computational Thinking Scales* (CTS) (Korkmaz, Çakir, & Özden, 2017), the *Computational Thinking Skills Scale* (CTSS) (Durak & Saritepeci, 2018), or the *Computer Programming Self-Efficacy Scale* (CPSES) (Kukul, Gökçearslan, & Günbatar, 2017). When we are interested in assessing the perceptions and attitudes of teachers, the research work of Yadav, Mayfield, Zhou, Hambrusch, and Korb (2014) can be highlighted. This kind of tools can be administered both before and after a CT educational intervention.
- **CT vocabulary assessment**: Finally, these tools intend to measure several elements and dimensions of CT, when they are verbally expressed by the subjects. These verbal expressions have been denominated as "computational thinking language" (e.g., see Grover, 2011).

It is worth noting that those different types of instruments have their own intrinsic characteristics, which lead each of them to approach CT assessment in a particular way. For example, while the diagnostic and the summative tools are based on student responses to predefined CT items or questions, the formative–iterative and the data-mining tools rely on the analysis of student programming creations and of student activity when developing CT, respectively. Thus, the information coming from each type of instruments has a different nature and all of them must be harmonized and triangulated to reach a complete CT assessment of the individual, as will be exemplified in the following empirical section.

Consequently, if only one from the aforementioned types of CT assessment tools is utilized, then it is very likely that an incomplete view of the students' CT is obtained. This incomplete and biased view can lead us to misunderstand the CT development of our students, and to take wrong educational decisions. In the same vein, Brennan and Resnick (2012) have stated that assessing students' computational competencies just looking at the programs created by the learners could be clearly insufficient, so they have emphasized the need of multiple means of assessment. Following this line of reasoning, Grover (2015) affirm that different types of complementary assessment tools must be systematically combined to reach a total and comprehensive understanding of the CT of our students. These combinations have been denominated as "systems of assessment," which is the *leitmotiv* on which we want to contribute in the next two sections.

Therefore, in the next section, we investigate these "systems of assessments" from an empirical psychometric approach. It is supposed that a "system of assessment" will be composed of instruments that provide convergent measures. Specifically, in the next section, we study the convergence of the measurements provided by three of the aforementioned CT assessment tools. Later on, in the fourth section, we speculate

www.dbooks.org

about some "systems of assessments" from a pedagogical research point of view, raising some questions like what assessment tools should be used according to the different moments of an educational CT intervention? What assessment tools should be used according to the different levels of cognitive complexity in our educational goals within a CT intervention? How to combine all the above to give answer to the most common research questions in the field of CT Education?

6.3 Convergent Validity Studies

In this section, we report two different convergent validity studies, which were carried out with two independent samples. The first study investigates the convergent validity of the Computational Thinking Test (CTt) with respect to a selection of Bebras Tasks. The second study does the same, but between the CTt and Dr. Scratch. Before reporting the results of both studies, some background and details about these three CT assessment tools are offered:

- **Computational Thinking Test (CTt)**: The CTt[3] is a diagnostic assessment tool that consists of a multiple-choice instrument composed of 28 items, which are administered online in a maximum time of 45 min. Each of the items of the CTt is presented either in a "maze" or in a "canvas" interface, and is designed according to the following three dimensions:
  - **Computational concept(s) addressed**: Each item addresses one or more of the following computational concepts, which appear in increasing difficulty and which are progressively nested along the test: basic directions and sequences; loops (repeat times, repeat until); conditionals (if, if/else); while conditional loops; simple functions.
  - **Style of response options**: In each item, responses are depicted in any of the following two styles: "visual arrows" or "visual blocks".
  - **Required cognitive task**: In order to be solved, each item demands to the subject one of the following cognitive tasks: to sequence an algorithm, to complete an incomplete algorithm, or to debug an incorrect algorithm.

  The CTt has demonstrated to be reliable and valid for assessing CT in subjects between 10 and 16 years old (Román-González, 2015; Román-González et al., 2017b). We show some examples of the CTt items in Figs. 6.1, 6.2, and 6.3, whose specifications are detailed in the respective caption.
- **The Bebras Tasks**: These tasks consist of a set of activities designed within the context of the *Bebras International Contest*,[4] a competition created in Lithuania in 2003, which is aimed at promoting the interest and excellence of K-12 students around the world in the field of CS from a CT perspective (Dagiene & Futschek,

[3] Sample copy available at: https://goo.gl/GqD6Wt.

[4] http://bebras.org/.

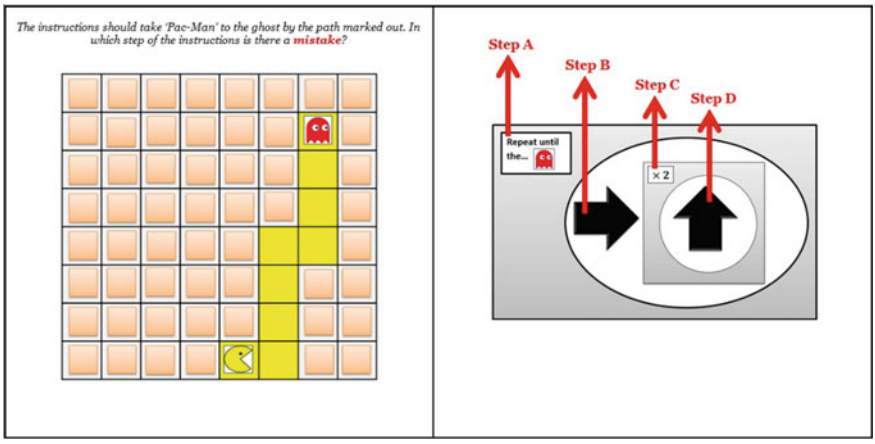


Fig. 6.1 CTt, item #11 ("maze"): loops "repeat until + repeat times" (nested); "visual arrows"; "debugging"

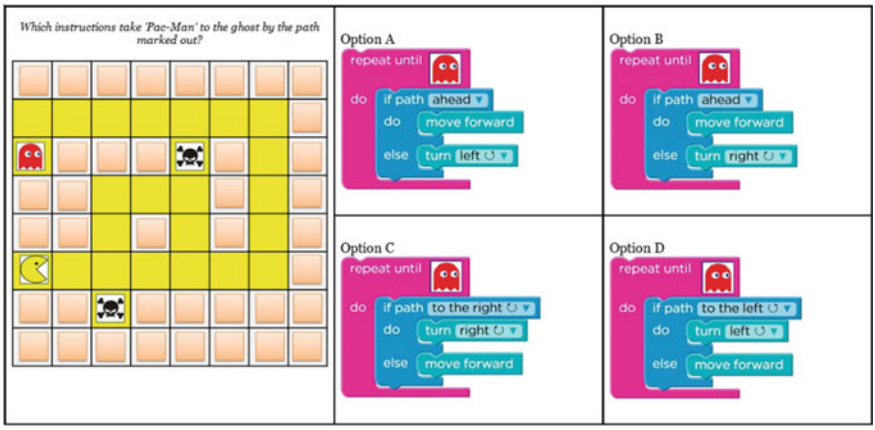


Fig. 6.2 CTt, item #18 ("maze"): loops "repeat until" + if/else conditional (nested); "visual blocks"; "sequencing"

2008). Every year, the contest launches a new set of Bebras Tasks, which require the students to transfer and project their CT skills in order to solve "real-life" problems. For this feature, in the previous sections, we have classified the Bebras Tasks as a CT skill transfer assessment tool. Moreover, another advantage of the Bebras Tasks is that they are independent from any particular software or hardware, and they can even be administered to individuals without any prior programming experience. The three Bebras Tasks used in our convergent validity study are shown in Figs. 6.4, 6.5, and 6.6.

www.dbooks.org

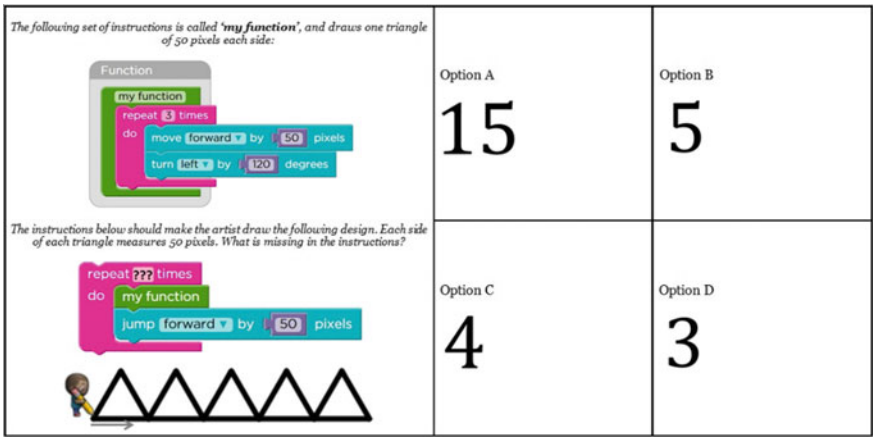


Fig. 6.3 CTt, item #26 ("canvas"): loops "repeat times" + simple functions (nested); "visual blocks"; "completing"

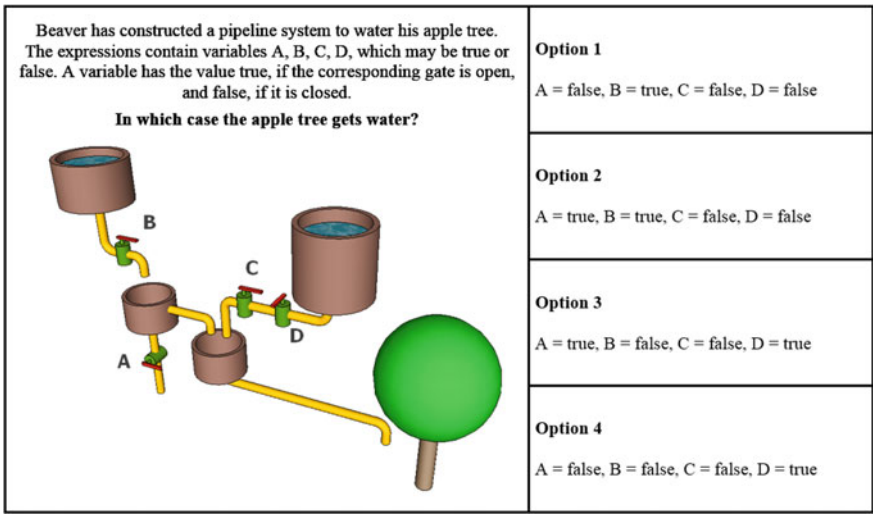


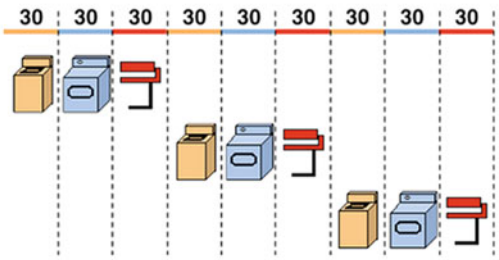
Fig. 6.4 Bebras Task #1: Water Supply (CT dimension involved: logic-binary structures) (reprinted by permission of http://bebras.org/)

- **Dr. Scratch**[5] (Moreno-León et al., 2015) is a free and open-source web application that analyzes, in an automated way, projects programmed with Scratch language. The score that Dr. Scratch assigns to a project is based on the degree of development of seven dimensions of CT competence: abstraction and problem decomposition, logical thinking, synchronization, parallelism, algorithmic notions of flow control,

[5] http://www.drscratch.org/.

Beaver Joe has started a new laundry business. He has got three machines: a washer, a dryer and a pressing iron. Every machine is connected through its own timer which provides for half an hour of electricity.

So, when a client arrives, he needs 90 minutes for all of the three procedures. And three clients using the machinery consequently need 270 minutes.



But now, there are three beavers arriving which are really busy. Each one them has enough clothes for a load of its own. But they agree that they want to finish as quickly as possible.

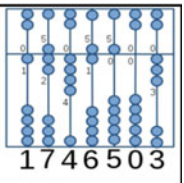
How many minutes does it take for all three of them to finish their laundry?

Option A	Option B	Option C	Option D
90 minutes	**120 minutes**	**150 minutes**	**270 minutes**

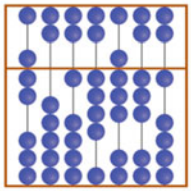
Fig. 6.5 Bebras Task #2: Fast Laundry (CT dimensions involved: parallelism, algorithms) (reprinted by permission of http://bebras.org/)

A number is represented on a Chinese abacus by the position of its beads. The value of a bead on the top part is 5; the value of a bead on the bottom part is 1. The abacus is reset to zero by pushing the beads away from the centre.

To represent the number 1 746 503 the appropriate beads are moved towards the centre of the abacus:



What number does the following abacus represent?



Option A	Option B	Option C	Option D
3014431	**7514831**	**3514431**	**7014831**

Fig. 6.6 Bebras Task #3: Abacus (CT dimension involved: abstraction, decomposition, algorithms) (reprinted by permission of http://bebras.org/)

www.dbooks.org

user interactivity, and data representation. These dimensions are statically evaluated by inspecting the source code of the analyzed project and given punctuation from 0 to 3, resulting in a total evaluation ("mastery score") that ranges from 0 to 21 when all seven dimensions are aggregated. In addition, Dr. Scratch generates a feedback report that is displayed to learners, which includes ideas and proposals to enhance the students' CT skills. The feedback report also encourages learners to try new Scratch blocks and structures, in order to improve the "mastery score" of their next projects (see Fig. 6.7). Because of this feature, in the previous sections, we have classified Dr. Scratch as a CT formative–iterative assessment tool.
The ecological validity of Dr. Scratch, for being implemented with positive results in school settings, has been demonstrated (Moreno-León et al., 2015). Furthermore, the convergent validity of Dr. Scratch with respect to the grades provided by CS educators (Moreno-León, Román-González, Harteveld, & Robles, 2017), and with respect to several software engineering complexity metrics (Moreno-León, Robles, & Román-González, 2016), has been already reported. Finally, Dr. Scratch has also proved discriminant validity to distinguish between different types of Scratch projects, such as animations, art projects, music projects, stories, and games (Moreno-León, Robles, & Román-González, 2017).

Given that the CTt, the Bebras Tasks, and Dr. Scratch are aimed at assessing the same construct (i.e., CT), but they approach this goal 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).

Thus, the convergent validity of these three instruments was investigated through two different correlational studies, with two independent samples. In the first study, the CTt and the aforementioned selection of Bebras Tasks were concurrently administered to a sample of Spanish middle-school students ($n = 179$), in pure pretest condition (i.e., students without prior formal experience in programming or similar). A positive, moderate, and statistically significant correlation was found ($r = 0.52$). As depicted in Fig. 6.8, the higher the CT ability of the subject (as measured by the CTt) is, the more likely it is that the subject correctly transfers CT to real-life problems (as measured by the Bebras Tasks).

Regarding the second study, it was performed in the context of an 8-week programming course in the Scratch platform, following the *Creative Computing* (Brennan, Balch, & Chung, 2014) curriculum and involving Spanish middle-school students ($n = 71$). Before starting with the course, the CTt was administered to the students in pure pretest condition. After the programming course, students took a posttest with the CTt and teachers selected the most advanced project of each student, which was analyzed with Dr. Scratch. These measures offered us the possibility to analyze the convergent validity of the CTt and Dr. Scratch in predictive terms ($CTt_{pre\text{-}test}$ * Dr. Scratch) and in concurrent terms ($CTt_{post\text{-}test}$ * Dr. Scratch). As in the first study, positive, moderate, and statistically significant correlations were found again, both in predictive ($r = 0.44$) and in concurrent terms ($r = 0.53$) (Fig. 6.9).

As we expected, the CTt, the Bebras Tasks, and Dr. Scratch are just *partially convergent* ($0.4 < r < 0.7$). This result is consistent with their different assessment approaches: *diagnostic-aptitudinal* (CTt), *skill transfer* (Bebras Tasks), and *forma-*



Fig. 6.7 Dr. Scratch assessment results and feedback report (bottom), for the Scratch project "Alice in Wonderland—a platformer" (https://scratch.mit.edu/projects/176479438/) (top)

tive–iterative (Dr. Scratch). Precisely, because of that *partial convergence*, we can extract very relevant pedagogical information, especially from the cases in which the expected correlation is not met (i.e., the cases that are especially deviated from the regression line). For example, if we find students with high CT aptitude (as measured by the CTt), but with low or medium scores in their programming projects (as measured by Dr. Scratch), we should probably review how programming is being taught in the classroom, since that teaching could be limiting CT high-ability students by

www.dbooks.org

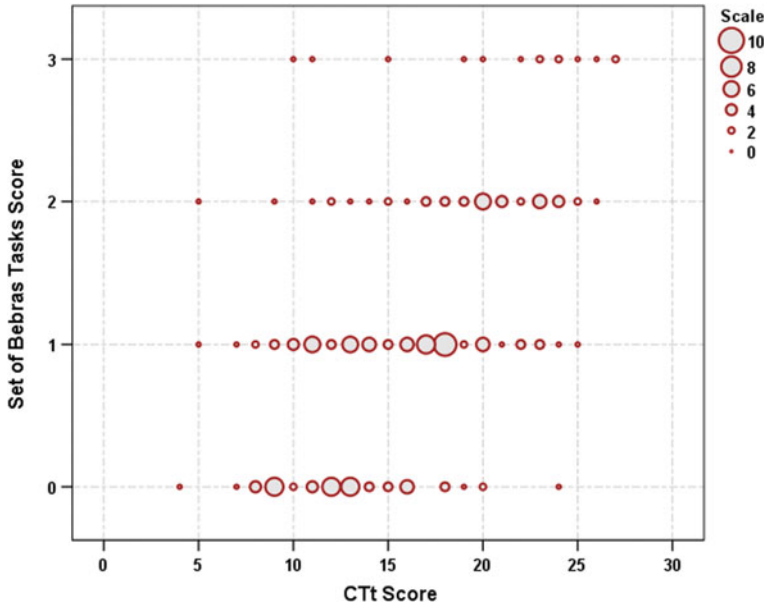


Fig. 6.8 Scatterplot CTt * set of Bebras Tasks

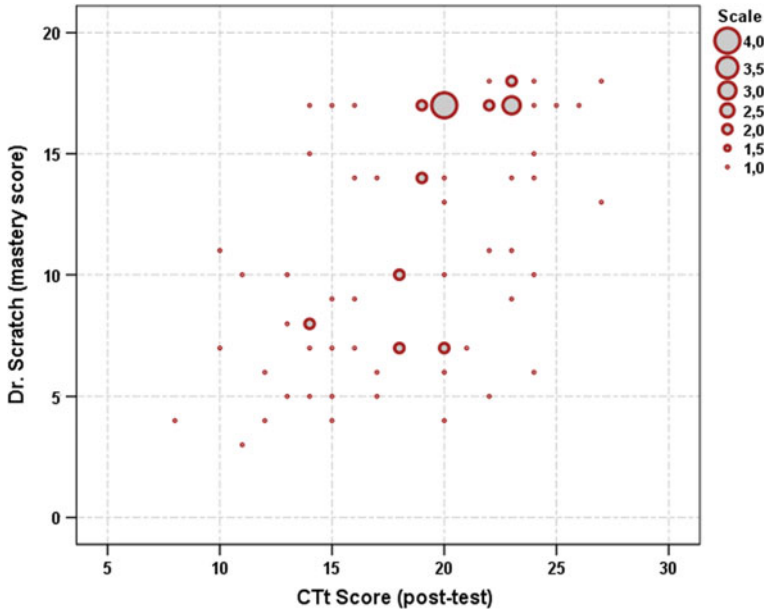


Fig. 6.9 Scatterplot $CTt_{post\text{-}test}$ * Dr. Scratch

means of a "*low ceiling*" methodology. Analogous reasoning can be followed for other partial convergences between any considered couple of CT assessment tools. Moreover, our results are one of the first empirical evidences that can contribute to depict a map with the convergence values between the main CT assessment tools all around the world, which ultimately would lead CT to be seriously considered as a psychological construct with its own entity. Further details of the aforementioned convergent validity studies can be found in Román-González, Moreno-León, and Robles (2017a).

6.4 A Comprehensive Evaluation of Computational Thinking Interventions

The empirical findings presented in the previous section have some implications. On the one hand, the just partial convergence found implies that none of the CT assessment tools considered in our studies should be used instead of any of the others. In other words, since the scores coming from these different instruments are only moderately correlated, none of the measures can replace or be reduced to any of the others; otherwise, the three tools might be combined in school contexts in order to achieve a more sensitive portrait of the students' CT skills. On the other hand, from a pedagogical point of view, the three assessment tools empirically studied seem to be theoretically complementary, as the weaknesses of the ones are the strengths of the others.

Thus, the CTt has some strengths, which are common to other diagnostic tools, such as it can be collectively administered in pure pretest conditions, so it could be used in massive screenings and early detection of computationally talented subjects or individuals with special needs in CT education. Moreover, the diagnostic tools (e.g., the CTt), and also the summative ones, can be utilized for collecting quantitative data in pre–post-evaluations aimed at developing CT. However, the CTt and other tools of the same type have some obvious weaknesses too, since they just provide static and decontextualized assessments, which are usually focused on computational "concepts" (Brennan & Resnick, 2012), ignoring "practices" and "perspectives" (see Table 6.1).

As a counterbalance of the above, the Bebras Tasks and other skill transfer tools provide naturalistic assessments, which are contextualized in significant "real-life" problems. Thus, this kind of tools can be used not only as measuring instruments, but also as tasks for teaching and learning CT in a meaningful way. When used strictly for assessment, the skill transfer tools are especially powerful if they are administered after certain time has elapsed from the end of the CT educational intervention, in order to check the degree of retention and transfer of the acquired CT skills. Nevertheless, the psychometric properties of this kind of tasks are still far of being demonstrated, and some of them are at risk of being too tangential to the core of CT.

www.dbooks.org

Table 6.1 Adequacy of different types of CT assessment tools regarding CT dimensions

	Computational concepts	Computational practices	Computational perspectives
Diagnostic tools	c	a	–
Summative tools	c	a	–
Formative–iterative tools	b	c	a
Data-mining tools	b	c	a
Skill transfer tools	b	a	b
Perceptions–attitudes scales	–	a	c
Vocabulary assessments	–	a	c

[a]Little adequacy, [b]Moderate adequacy, [c]Excellent adequacy, – No adequacy at all

Finally, Dr. Scratch complements the CTt and the Bebras Tasks, given that the former involves "computational practices" (Brennan & Resnick, 2012) that the other two do not, such as iterating, testing, remixing, or modularizing. However, Dr. Scratch, like other formative–iterative or data-mining tools, cannot be used in pure pretest conditions, since it is applied onto programming projects after the student has learnt at least some computer programming for a certain time. In other words, these formative–iterative and data-mining tools are affected by the kind of programming instruction delivered by the teacher, and by the kind of programming projects that are proposed to the students.

In this vein, Table 6.1 shows a tentative proposal on the degree of adequacy between the different types of CT assessment tools with respect to the three CT dimensions stated by Brennan and Resnick (2012) framework: "*computational concepts*" (sequences, loops, events, parallelism, conditionals, operators, and data); "*computational practices*" (experimenting and iterating, testing and debugging, reusing and remixing, abstracting, and modularizing); and "*computational perspectives*" (expressing, connecting, and questioning).

All of the above leads us to affirm the complementarity of the different types of CT assessment tools in educational scenarios, and to raise the clear possibility of building up powerful "systems of assessments" through a proper combination of them. Hence, from a chronological point of view, the degree of adequacy in the use of each type of assessment tools at the different phases of CT educational interventions and evaluations is presented in Table 6.2.

Furthermore, when reflecting about the characteristics of the various and diverse CT assessment tools, it is possible to state what levels of the Bloom's (revised) taxonomy of cognitive processes are being addressed by each type (Fig. 6.10). Thus, the diagnostic tools provide information about how the students "remember" and "understand" some CT concepts; the skill transfer tools inform about the students' competence to "analyze" and "apply" their CT skills to different contexts; and the formative–iterative tools allow the students to "evaluate" their own and others' projects, as well as to "create" better and more complex ones. In addition, the summative tools

Table 6.2 Chronological uses of the different types of CT assessment tools

	Before the intervention (pretest)	Along the intervention (progression)	Just after the intervention (posttest)	Sometime after the end of the intervention (retention and transfer)
Diagnostic tools	[c]	–	[b]	[a]
Summative tools	[a]	–	[c]	[b]
Formative–iterative tools	–	[c]	[b]	–
Data-mining tools	–	[c]	[b]	–
Skill transfer tools	[a]	[b]	[b]	[c]
Perceptions–attitudes scales	[c]	–	[c]	[a]
Vocabulary assessments	[a]	–	[c]	[b]

[a]Little adequacy, [b]Moderate adequacy, [c]Excellent adequacy, – No adequacy at all

serve as a transition between the lower and the intermediate levels of the taxonomy, while the data-mining tools do so between the intermediate and the upper ones. Nonetheless, there might be still some CT issues, such as usability, originality, interface design, etc., which could not be assessed by tools, otherwise by human sensitivity. Finally, since the perceptions–attitudes scales do not assess cognitive processes, but noncognitive ones, these tools are not placed along the pyramidal taxonomy, but they surround and frame it.

As a final contribution, we raise some research designs (RD) that could compose together a comprehensive evaluation model of CT interventions. For each research design, we specify what kind of research question is addressed, and what CT assessment tools should be used:

- **Correlational–predictive RD**: Through this RD, we investigate to what extent the current level of the subject's CT can explain–predict his/her future performance in related tasks (e.g., his/her academic achievement in STEM subjects, his/her computer programming competence, etc.). The right instruments to be utilized in this kind of RD are the diagnostic tools. Once the predictive power of a CT diagnostic tool has been established, then it can be used subsequently to detect subjects who may need preventive CT educational interventions.
- **Quasi-experimental RD**: This type of RD is aimed at verifying if a certain intervention has been effective to develop the CT of the students. A quasi-experimental RD needs pretest and posttest measures, administered on treatment and control groups, in order to monitor the causes of the changes in those measures, if any. The proper instruments to be used in this RD are the summative tools (and some of the diagnostic tools too). Moreover, it is imperative that the intervention under

www.dbooks.org

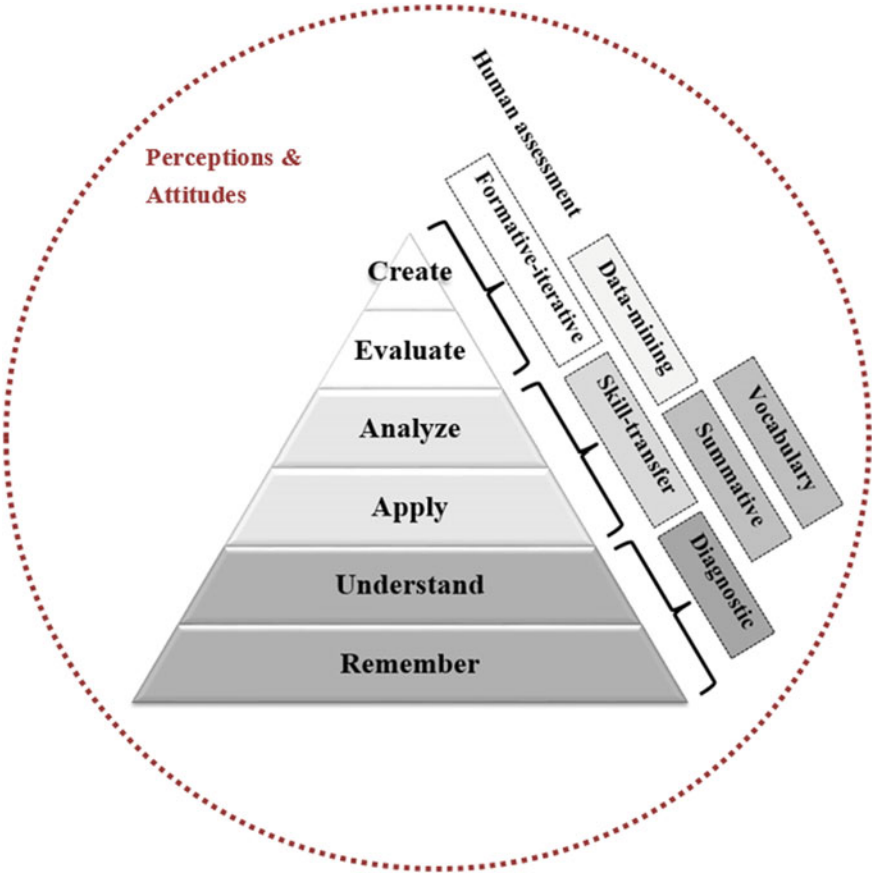


Fig. 6.10 Bloom's taxonomy and CT assessment tools

evaluation is well-documented, so that the quasi-experiment could be replicated in other contexts, and the effect sizes could be later compared.

- **Causal-comparative RD**: If some modulating variables are introduced in the previous RD, then we can also address the question about what factors affect the correlations and/or differences found. Some of the classical modulating variables are gender, age, programming experience, teacher experience, etc. In addition, it would be extremely relevant to include students and teachers' perceptions and attitudes as complementary modulating variables, since it is becoming more evident that noncognitive aspects influence the progression and the results of CT interventions (Román-González et al., 2018). Thus, perceptions and attitudes scales should be incorporated in this kind of RD.
- **Longitudinal RD**: Longitudinal studies involve repeated measures of the same variables, coming from the same subjects, and over a certain period of time. Although this kind of RD is quite difficult to perform, due to its high cost and

sample mortality among other things, it provides answer to extremely relevant questions such as how the CT learning paths are, or what barriers and misconceptions tend to appear along them. If the longitudinal RD is performed along a CT intervention, then the formative–iterative and/or the data-mining tools should be used. Else, when this kind of RD is chronologically extended beyond the end of the CT intervention, then skill transfer tools should be included in the research too.

Furthermore, when combining several CT assessment tools in a single RD, then powerful multivariable analysis techniques can be applied to shed some light on some relevant issues: for example, cluster analysis could be performed to state and describe some profiles of students according to their different CT scores; multiple regression or discriminant analysis could be used to explain what factors better explain different CT levels, etc.

In summary, everything exposed in this section may help scholars and policy-makers to decide and perform accurate evaluation designs of CT according to their needs and inquiry goals.

6.5 Conclusions and Further Research

Computational thinking (CT) is entering adulthood. Along this process, CT is being demystified and it is being demanded to become a more solid psychological variable in order to assure its survival. In this vein, CT assessment might be a key aspect within the whole CT study field. Because of its own nature, CT assessment can decisively contribute to consolidate CT as a mature construct. Pushing in this direction, in this chapter, we have reviewed the state-of-the-art CT assessment tools; we have presented two convergent validity studies between three of those instruments, which could be a starting point for a map with the convergence values between the main CT assessment tools all around the world; and we have raised a comprehensive evaluation model that could combine properly the different types of instruments according to the inquiry and research goals, and which could guide future actions of scholars and policy-makers.

Further research should focus on completing the "convergence map," and on performing and replicating the aforementioned prototypical research designs in order to compare their effectiveness in different countries, populations, and educational contexts.

www.dbooks.org

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). https://doi.org/10.1145/1953163.1953241.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., et al. (2016). *Developing computational thinking in compulsory education-implications for policy and practice*. Seville: Join Research Center (European Commission). Retrieved from http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf.

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, Vancouver, Canada* (pp. 1–25). Retrieved from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf.

Carlson, K. D., & Herdman, A. O. (2012). Understanding the impact of convergent validity on research results. *Organizational Research Methods, 15*(1), 17–32.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education, 109,* 162–175. https://doi.org/10.1016/j.compedu.2017.03.001.

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). Berlin, Germany: Springer. https://doi.org/10.1007/978-3-540-69924-8_2.

Daily, S. B., Leonard, A. E., Jörg, S., Babu, S., & Gundersen, K. (2014). Dancing alice: Exploring embodied pedagogical strategies for learning computational thinking. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 91–96). https://doi.org/10.1145/2538862.2538917.

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education, 116,* 191–202. https://doi.org/10.1016/j.compedu.2017.09.004.

Eguiluz, A., Guenaga, M., Garaizar, P., & Olivares-Rodriguez, C. (2017). Exploring the progression of early programmers in a set of computational thinking challenges via clickstream analysis. *IEEE Transactions on Emerging Topics in Computing* (in press). https://doi.org/10.1109/TETC.2017.2768550.

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*. Retrieved from https://pdfs.semanticscholar.org/69a7/c5909726eed5bd66719aad69565ce46bbdcc.pdf.

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). Retrieved from https://www.sri.com/sites/default/files/publications/aera2015-_systems_of_assessments_for_deeper_learning_of_computational_thinking_in_k-12.pdf.

Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Transactions on Computing Education (TOCE), 17*(3), 14:1–14:25. https://doi.org/10.1145/3105910.

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., & Pea, R. (2013). Computational thinking in K–12 a review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051.

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), 583–596. Retrieved from http://www.bjmc.lu.lv/fileadmin/user_upload/lu_portal/projekti/bjmc/Contents/4_3_15_Kalelioglu.pdf.

Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 59–66). https://doi.org/10.1109/VLHCC.2010.17.

Koh, K. H., Basawapatna, A., Nickerson, H., & Repenning, A. (2014). Real time assessment of computational thinking. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 49–52). https://doi.org/10.1109/VLHCC.2014.6883021.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*. https://doi.org/10.1016/j.chb.2017.01.005.

Kukul, V., Gökçearslan, S., & Günbatar, M. S. (2017). Computer programming self-efficacy scale (CPSES) for secondary school students: Development, validation and reliability. *Eğitim Teknolojisi Kuram ve Uygulama*, *7*(1). Retrieved from http://dergipark.ulakbim.gov.tr/etku/article/view/5000195912.

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. https://doi.org/10.1016/j.chb.2014.09.012.

Maiorana, F., Giordano, D., & Morelli, R. (2015). Quizly: A live coding assessment platform for App Inventor. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (pp. 25–30). https://doi.org/10.1109/BLOCKS.2015.7368995.

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., & 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), 1–23. Retrieved from http://www.um.es/ead/red/46/moreno_robles.pdf.

Moreno-León, J., Robles, G., & Román-González, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. In *2016 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1040–1045). https://doi.org/10.1109/EDUCON.2016.7474681.

Moreno-León, J., Robles, G., & Román-González, M. (2017). Towards data-driven learning paths to develop computational thinking with scratch. *IEEE Transactions on Emerging Topics in Computing*. https://doi.org/10.1109/TETC.2017.2734818.

Moreno-León, J., Román-González, M., Harteveld, C., & Robles, G. (2017). On the automatic assessment of computational thinking skills: A comparison with human experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 2788–2795). New York, NY, USA: ACM. https://doi.org/10.1145/3027063.3053216.

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). https://doi.org/10.1145/2818314.2818320.

Ota, G., Morimoto, Y., & Kato, H. (2016). Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (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). https://doi.org/10.13140/RG.2.1.4203.4329.

Román-González, M., Moreno-León, J., & Robles, G. (2017a). Complementary tools for computational thinking assessment. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), *Proceedings of International Conference on Computational Thinking Education (CTE 2017)* (pp. 154–159).

www.dbooks.org

The Education University of Hong Kong. Retrieved from http://www.eduhk.hk/cte2017/doc/CTE2017Proceedings.pdf.

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017b). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior, 72,* 678–691. https://doi.org/10.1016/j.chb.2016.08.047.

Román-González, M., Pérez-González, J.-C., Moreno-León, J., & Robles, G. (2018). Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior, 80,* 441–459. https://doi.org/10.1016/j.chb.2017.09.030.

Rushkoff, D. (2010). *Program or be programmed: Ten commands for a digital age*. New York: OR Books.

Sadin, É. (2015). *La vie algorithmique: Critique de la raison numérique [The Algorithmic Life: A Critique of Digital Rationality]*. Paris: L'Echappée.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22,* 142–158. https://doi.org/10.1016/j.edurev.2017.09.003.

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).

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). 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).

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

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.

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). https://doi.org/10.1145/2576872.

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).

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 7
Introducing and Assessing Computational Thinking in the Secondary Science Classroom



Hillary Swanson, Gabriella Anton, Connor Bain, Michael Horn and Uri Wilensky

Abstract The importance of computational thinking (CT) as a goal of science education is increasingly acknowledged. The representational affordances of computational tools are changing the way knowledge can be constructed, expressed, and understood across disciplines. Our group has worked to explicitly characterize CT practices used by computational STEM researchers (CT-STEM practices) and to develop computational science curricula that teach both CT-STEM practices and science content. We have previously characterized four strands of CT-STEM practices: *data practices*, *modeling and simulation practices, computational problem-solving practices*, and *systems thinking practices*. In this chapter, we show that a group of 9th grade students developed competencies for *modeling and simulation practices* as a result of their engagement in our computational biology curriculum. As evidence, we present findings from a quantitative analysis of students' written responses to assessments given before and after their participation in three computational biology units. Results suggest that the computational biology curriculum helped students develop a number of important competencies for the strand on *modeling and simulation practices*. Our work contributes to the field's understanding of how science curricula can be designed to foster students' development of CT-STEM practices and how this development can be assessed.

H. Swanson (✉) · G. Anton · C. Bain · M. Horn · U. Wilensky
School of Education and Social Policy, Northwestern University, Evanston, USA
e-mail: hillary.swanson@northwestern.edu

G. Anton
e-mail: gabriellaanton3.2020@u.northwestern.edu

C. Bain
e-mail: connorbain@u.northwestern.edu

M. Horn
e-mail: michael-horn@northwestern.edu

U. Wilensky
e-mail: uri@northwestern.edu

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_7

www.dbooks.org

7.1 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 (Kaczmarczyk & Dopplick, 2014). 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 (CS) classes (Margolis, 2008; Margolis & Fisher, 2003) and thus are not engaging in CT practices through traditional channels. Our goal is to improve access for all students, especially those underrepresented in CS, by embedding CT practices in subjects such as biology, chemistry, and physics, which all secondary students are expected to take.

We believe that developing CT practices in the context of science subjects is a productive endeavor. Our group has worked to explicitly characterize key activities relevant to particular CT-STEM practices as specific learning objectives and used these to guide our development of science curricula and assessments. In this paper, we show that a group of 9th grade students (ages 14–15 years) developed competencies for *modeling and simulation practices* as a result of their engagement in our computational biology curriculum.

7.2 Theoretical Orientation

Our perspective on computational thinking is motivated by Wilensky and Papert's (2010) restructuration theory, which demonstrates that the representational form in which knowledge is encoded significantly influences how it may be understood and learned. Restructuration theory builds on a history of psychological and historical research that has argued that representational forms shape human knowledge and understanding, both at the individual and societal level (e.g., Goody, 1977; Papert, 1980; Olson, 1994; diSessa, 2001). In light of this theory, it is clear that the representational affordances of computational tools are changing the way knowledge can be constructed, expressed, and understood across disciplines.

Building on this perspective, our group has worked to characterize the nature of computational thinking practices in the STEM disciplines. On the basis of interviews with computational STEM researchers, we developed an operational definition of CT-STEM as a set of practices and organized these as a taxonomy (Weintrop et al., 2016). The taxonomy categorizes CT-STEM practices in terms of four major strands: *data practices*, *modeling and simulation practices*, *computational problem-solving*

practices, and *systems thinking practices*. Figure 7.1 depicts the practices within each of these four strands.

Though they are not unique to STEM, these CT practices are common to the STEM disciplines. In this way, they differ from the domain-general CT practices characterized by Wing (2006) (e.g., *using computer science concepts to solve problems and design systems*), the National Research Council (2010) (e.g., *heuristic reasoning, search strategies, and problem abstraction* and *decomposition*), and Brennan and Resnick (2012) (e.g., *being incremental and iterative*, *testing and debugging*, *reusing and remixing*, and *abstracting and modularizing*). We identified key activities relevant to each of the CT-STEM practices in our taxonomy and proposed those as learning objectives. We have used these learning objectives to guide our development of curricula and assessments that foster and evaluate students' development of computational thinking practices in STEM subjects at the secondary level.

In the study described herein, we analyze student gains in the *modeling and simulation practices* strand of the taxonomy. We build on work we have done using agent-based modeling in science classrooms (Blikstein & Wilensky, 2009; Sengupta & Wilensky, 2009; Horn & Wilensky, 2012; Horn, Brady, Hjorth, Wagh, & Wilensky, 2014; Levy & Wilensky, 2009; Wilensky, 2003; Wilensky & Reisman, 2006). In future work, we plan to analyze each of the four strands and gains in summative assessments of CT-STEM practices.

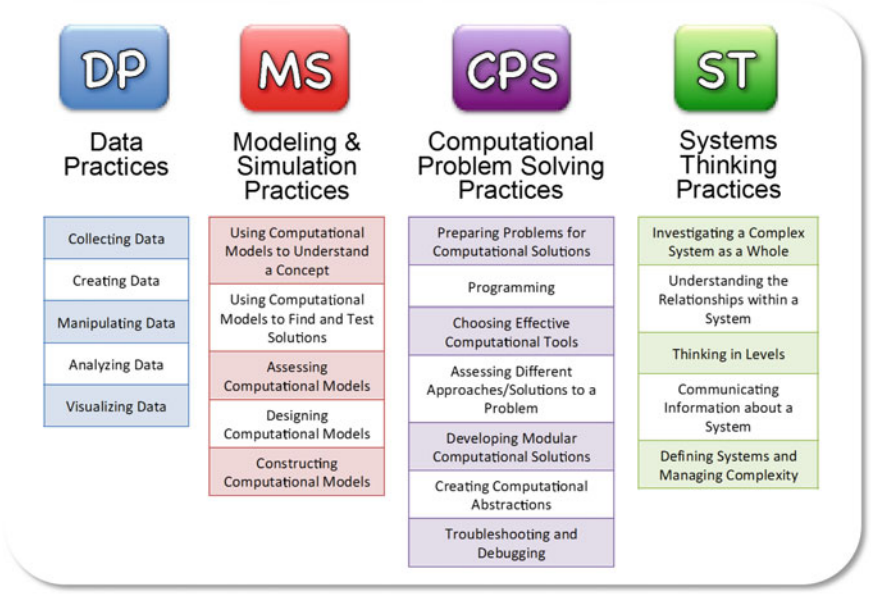


Fig. 7.1 Computational thinking in STEM taxonomy

www.dbooks.org

7.3 Method

We show that a group of 9th grade students developed competencies for *modeling and simulation practices* as a result of their engagement in our computational biology curriculum. As evidence, we present findings from a quantitative analysis of 133 9th grade students' written responses to assessments given before and after their participation in three computational biology units.

7.3.1 Study Design

The data in this study come 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 a partner secondary school in a Midwestern city in the United States. Students were given a CT-STEM practices pre-test (Weintrop et al., 2014) at the beginning of the school year and a CT-STEM practices post-test at the end of the school year. Over the year they participated in three CT-STEM biology units; each unit approximately four days long. We investigated the role of the CT-STEM science units in students' development of competencies for *modeling and simulation practices* by looking for statistically significant gains in student scores for particular items from pre- to post-test.

7.3.2 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 for its diversity and for the willingness of its teachers to participate in our study. The size of the school was typical for an urban public secondary school, with approximately twelve hundred students enrolled (71.1% Black/African American, 24.5% Hispanic/Latino, 1.6% Asian, 0.3% American Indian, 0.2% Pacific Islander, 0.9% Bi-Racial, 1.4% White), with 62% 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 our first learning objective and 24 did not complete the assessment item measuring our second learning objective; these students' responses were therefore removed from the analyzed datasets.

7.3.3 CT-STEM Units

The students participated in three computationally-enriched biology units over the course of the school year. Each unit took approximately four class periods 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 (e.g., 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 HIV. 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 to transfer between partners) on the rate of the 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 the population (Novak & Wilensky, 2011).

All units were meant to help students develop expertise regarding learning objectives related to *modeling and simulation practices* by engaging with 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 units) 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.

7.3.4 Data Collection

The pre- and post-test were each given during one 50-min class period at the beginning and end of the school year. Students took the tests individually on school laptops in their biology classrooms. The pre- and post-tests were not designed to evaluate students' science content knowledge. Rather, they were meant to evaluate their development of competencies relevant to CT-STEM practices. In this chapter, we present results concerned with two particular learning objectives within our *modeling and simulation practices* strand.

The first learning objective focuses on an activity relevant to the CT-STEM practice *using computational models to understand a concept* and states that a student should be able to "explore a model by changing parameters in the interface or code."

www.dbooks.org

This is a very basic activity 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 focuses on an activity relevant to the CT-STEM practice *assessing computational models* and states that a student should be able to "identify the simplifications made by a model." This activity 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 which they were given basic instructions on how to operate. For the pre-test, students interacted with a simulation (shown in Fig. 7.2) that modeled climate change and showed the relationship between temperature and amount of CO_2 in the atmosphere (Tinker & Wilensky, 2007). For the post-test, students explored a simulation (shown in Fig. 7.3) that modeled the relationship between the pressure of a gas, its

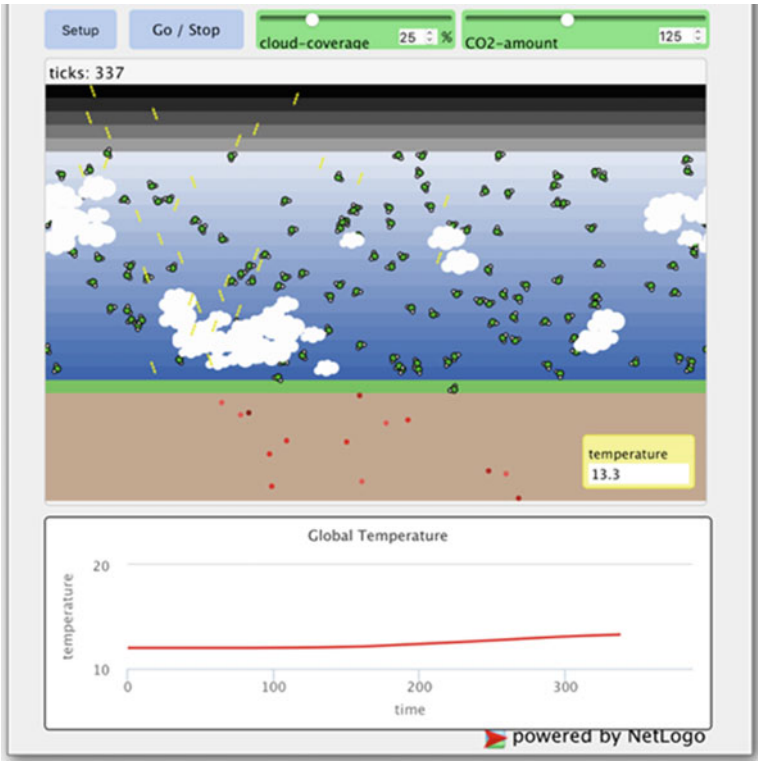


Fig. 7.2 Screenshot of pre-test simulation that models the relationship between temperature and atmospheric CO_2 levels

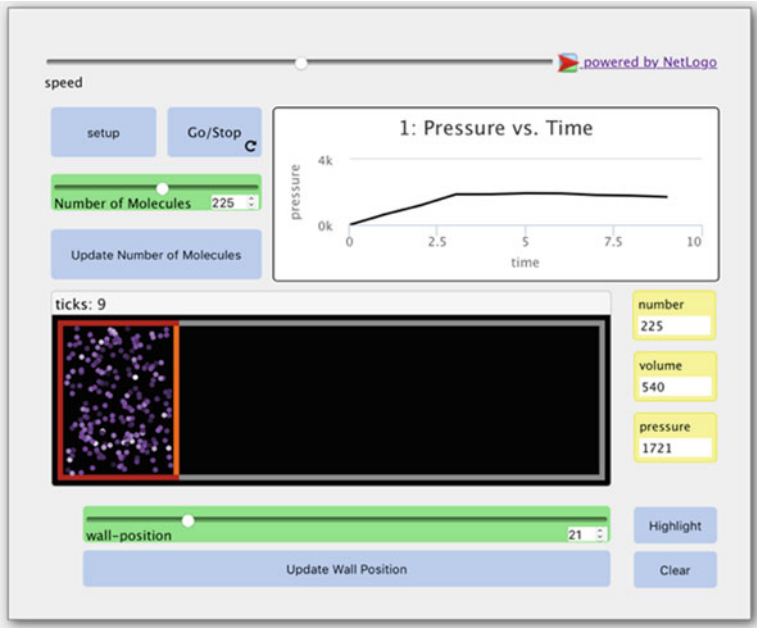


Fig. 7.3 Screenshot of post-test simulation that models the relationship between the pressure of a gas, its volume, and the number of particles

volume, and the number of particles in a sealed environment (Wilensky, 1997a, 2003; Wilensky, Novak, & Levy, 2005).

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. To assess students' abilities to *identify simplifications made by a model*, we analyzed their responses to test items (quoted below) that asked them for the ways in which the simulations differed from the real-world. These assessment items were selected to investigate students' development with respect to the same learning objectives across two very different computationally modeled phenomena.

7.3.5 Analytic Approach

We used a combined top-down (learning objective driven) and bottom-up (data driven) approach to create rubrics for characterizing students' competencies with respect to each learning objective and evaluating their responses to pre- and post-test questions. Two researchers then analyzed students' responses to the two assessment items for both pre-and post-tests. They coded responses (identifying the compe-

www.dbooks.org

tencies outlined in the rubrics) and then scored them. The researchers' inter-rater reliability for the pre-test was 97% for the item measuring the first learning objective and 90% for the item measuring the second learning objective. Inter-rater reliabilities for the post-test items were 95% and 80% respectively.

7.3.5.1 Learning Objective 1: Explore a Model by Changing Parameters

For the pre-test, students were asked to *explore a model* by changing its parameters in the context of the greenhouse gas simulation. In particular, they responded to the prompt: "Set cloud coverage to 0%. Take some time to experiment with different settings for the 'CO_2-amount' slider. What happens to the temperature if you increase the amount of the CO_2 in the model?" For the post-test, students were asked to *explore the model* in the context of the gas-law simulation. 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?" 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, 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*.

We examined students' pre- and post-test responses, sorting responses into categories based on similarities that were relevant to the learning objective. Three categories emerged that were representative of response types across both pre- and post-test. These are *comparing across trials, attending to explanatory factors*, and *attending to parameter-system relationships*. We identified these as three competencies relevant *to exploring a model by changing parameters in the interface or code*. These competencies are outlined, described, and illustrated with examples from the data in Table 7.1.

We scored students' responses by awarding one point for each competence demonstrated in their response and taking the sum of these points. This resulted in scores ranging from 0 to 3. We characterize the distribution of competencies (demonstrated in both pre- and post-test) in our findings section.

7.3.5.2 Learning Objective 2: Identify Simplifications Made by a Model

As part of the pre-test, students were asked to identify the simplifications made by the greenhouse simulation. As part of the post-test, they were asked to identify the simplifications made by the gas-law simulation. For both tests, they responded to the

Table 7.1 Pre- and post-test rubric for analyzing students' responses and characterizing the competencies they drew upon when *exploring a model by changing parameters in the interface or code*

Comparing across trials Response compares data across multiple simulation trials. When exploring a model to learn more about the dynamics of, or test a hypothesis regarding, a complex system, it is important to observe more than one simulation run. This is because complex systems are inherently random and the results of changing a parameter vary over different simulation trials. A pattern of cause-effect relationships will hover around an average tendency, but this average tendency may not be exactly embodied in one (or several) simulation trials. So, if a student only runs one trial, they may have a misguided impression of a pattern in system behavior. It is also a good idea to run multiple trials in order to systematically compare the effects of different parameter values on system behavior.	
Pre-test	"When I increase the amount of CO_2 the earth heats up much faster than it would if the setting was lower."
Post-test	"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."
Attending to explanatory factors Response provides some explanation for the relationship between system parameters and macro-level patterns. Explanations such as this convey the students' reasoning and suggest that they are not only attending to cause and effect, but that they are going one step further and trying to make sense of the relationship between cause and effect—a fundamental activity of science.	
Pre-test	"The carbon dioxide blocks the IR from reaching the sky but doesn't stop the sunlight from reaching the ground the higher you increase the Carbon Dioxide."
Post-test	"A bigger area and less particles shouldn't produce a large amount of pressure since it's a lot of space for the particles."
Attending to parameter-system relationships Response describes relationship between system parameters and macro-level patterns. It is important to attend to outcomes of the simulation when tinkering with or testing parameters, in order to notice relationships between cause and effect. Simple qualitative characterizations of the relationships within a system are a foundation for constructing more detailed or mathematical relationships. A simple qualitative understanding of a cause-effect relationship can be a powerful tool for reasoning about system dynamics and for conveying the big ideas about the relationships within a system to others. In the scientific world, these "others" might be collaborators or members of the scientific community at-large.	
Pre-test	"The temperature increases."
Post-test	"I slid the wall-position to its maximum and the number of particles to its minimum."

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. Five categories emerged that were representative of response types across both pre- and post-test. These are *attending to general issues, attending to rep-*

www.dbooks.org

resentational issues, attending to issues of controllability, attending to issues of completeness, and attending to procedural limitations. We identified these as five competencies relevant to *identifying simplifications made by a model*. These competencies are arranged in order of increasing sophistication, described and illustrated with examples from the data in Table 7.2.

General comments about accuracy and representational limitations seemed to be the easiest to make with attention to mere surface-features. Responses that identified 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. Responses were therefore awarded a slightly higher score (two points) for identi-

Table 7.2 Pre- and post-test rubric for analyzing students' responses and characterizing the competencies they drew upon when *identifying simplifications made by a model*

Attending to general issues—score: 1 Response refers to general, rather than specific, inaccuracies or missing factors. This suggests that students understand that the model is not an accurate depiction of reality, however they have not done the cognitive work of identifying a particular limitation.	
Pre-test	"In reality, other factors could come into play rather than just CO_2 and clouds."
Post-test	"Inaccuracy in particles and wall position can make it different from the real world."
Attending to representational issues—score: 1 Response refers to representational limitations of the model. This suggests that students understand that the model is not an accurate depiction of reality. This is not a "meaningful" limitation compared to other limitations that students mentioned, as the simplification does not influence the interactions between the elements of the model and therefore does not influence the outcome of any given simulation trial.	
Pre-test	"Obviously, sunlight is not a bunch of little sticks raining down."
Post-test	"It's not actually life size."
Attending to issues of 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. This suggests that students understand the model is different from reality because it allows certain conditions to be tested by being varied, which is impossible to do in reality.	
Pre-test	"Because you can control how much CO_2 and cloud coverage there is."
Post-test	"In real life, you cannot add or subtract molecules nor can you adjust the wall positioning."
Attending to issues of completeness—score: 2 Response refers to specific elements or factors that are missing from, or extraneous to, the model. These students recognize that a model is an approximation of reality. They have compared it with the real world and identified factors that are found in the real world but missing from the model. It is probable they believe these factors are somehow important to the model and would change the outcome of a simulation trial. Limitations such as these are important for scientists to identify, because they help them interpret their results and recognize their limitations.	

(continued)

Table 7.2 (continued)

Pre-test	"There are humans on earth and humans also can add to the amount of heat."
Post-test	"The real world, does not have this many boundaries and an infinite number of particles."
Attending to procedural limitations—score: 3 Response refers to interactions, behaviors, or relationships within the model that differ from real life. Limitations such as this are extremely important for scientists to recognize, as they are related to how successful the model is at approximating reality. Procedural limitations of the model influence the outcome of a simulation run in an important way: if the simulation does not reproduce patterns found in real-world data, something about the encoded theoretical model is *wrong* and needs to be revised.	
Pre-test	"CO_2 might not speed up that much when it absorbs IR light."
Post-test	"Particles don't travel in and out of the room in this simulation, when in real life they do."

fying these simplifications. 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 compare this with scientific laws or theories. Responses were therefore awarded the highest score (three points) for these simplifications. Responses that were not coded for any of the three competencies were given a score of zero. For our statistical analysis, we counted the point-value of the highest competence demonstrated in a student's response. Scores ranged from 0 to 3. We characterize the distribution of competencies (demonstrated in both pre- and post-test) in our findings section.

7.4 Findings

To test whether the computational biology units played a role in developing competencies for *modeling and simulation practices*, pre- and post-test scores for the two items were compared using a Wilcoxon signed-rank test and competence frequencies were compared using McNemar's tests. We report the results of our analysis below.

7.4.1 Learning Objective 1: Explore a Model by Changing Parameters

The class average for the pre-test item measuring students' ability to *explore a model by changing parameters in the interface or code* was a score of 1.24. The average for the post-test item was a score of 1.46. The p-value obtained using a paired Wilcoxon signed-rank test (with continuity correction) was 0.01389 ($V = 1175.5$). The difference in student scores is therefore statistically significant at the 5% level,

www.dbooks.org

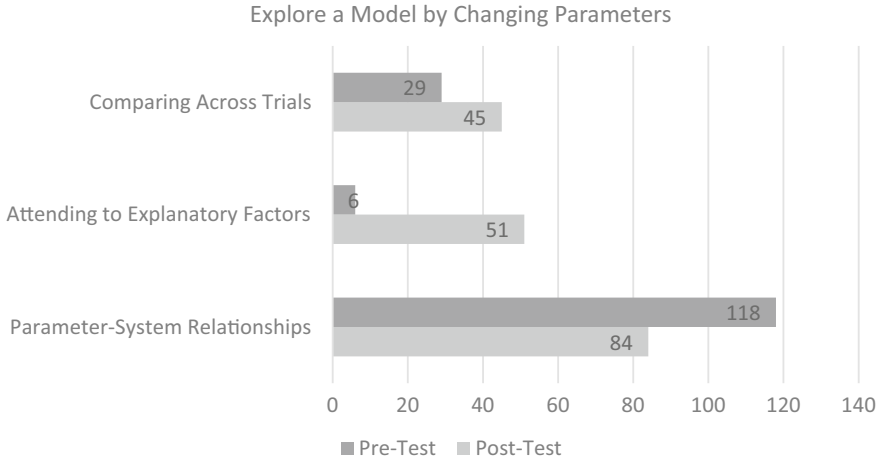


Fig. 7.4 Frequencies of competencies demonstrated in students' responses to the pre- and post-test items assessing their mastery of learning objective 1

which supports the claim that engagement in our curriculum helped students improve with regards to this learning objective. To gain a more nuanced understanding of how students developed their abilities to *explore a model*, we compared the frequencies of competencies they demonstrated in pre- and post-test responses. The bar chart (Fig. 7.4) illustrates the number of students *comparing across trials*, *attending to explanatory factors*, and *attending to parameter-system relationships*, on both the pre- and post-test.

Notably, the frequencies increased from pre- to post-test for *comparing across trials* and *attending to explanatory factors*. Frequencies decreased for *attending to parameter-system relationships*. Below, we present results of statistical analyses that show whether these changes in frequency may have been the result of students' participation in our computational biology units.

7.4.1.1 Comparing Across Trials

An increase in *comparing simulation results across multiple trials* suggests students have adopted a more systematic approach or learned the value of tinkering in exploration. An example of a student response that suggests a student is comparing simulation results across multiple trials is "When I increase the amount of CO_2 the earth heats up much faster than it would if the setting was lower." A McNemar's test on our counts (without continuity correction) results in a test statistic (χ^2) of 4.2667 and a p-value of 0.03887, which is a significant effect. This is evidence that engagement in our computational biology curriculum improved students' abilities to *explore a model* by encouraging more students to compare results across multiple trials.

7.4.1.2 Attending to Explanatory Factors

An increase in *attending to explanatory factors* suggests more students are drawing inferences from the model visualization to understand the mechanisms that produce system behavior. An example of a student response that suggests a student is attending to explanatory factors is "The carbon dioxide blocks the IR from reaching the sky but doesn't stop the sunlight from reaching the ground the higher you increase the Carbon Dioxide." A McNemar's test (with continuity correction) results in a test statistic (χ^2) of 37.961 and a p-value of less than 0.001, which is a significant effect. This is evidence that engagement in our computational biology curriculum improved students' abilities to *explore a model* by encouraging them to attend to explanatory factors.

7.4.1.3 Attending to Parameter-System Relationships

A decrease in *attending to parameter-system relationships* suggests fewer students are articulating inferences from the model visualization about the macro-level relationships between parameter settings and system behavior. An example of a student response that suggests a student is attending to parameter-system relationships is "The temperature increases," when the student is asked to describe what happens to the environment as atmospheric CO_2 increases. A McNemar's test (without continuity correction) results in a test statistic (χ^2) of 27.524 and a p-value of less than 0.001, which is a significant effect. This decrease may be the result of a difference in the form of pre- and post-test items. While the pre-test item asked students *only* what would happen to a macroscopic characteristic of the system as a parameter was varied, the post-test item asked students for *both* the parameter value that resulted in a particular system characteristic and the steps they took to find that value. It is possible that the additional question caused students to shift their attention away from answering the question focused on parameter-system relationships.

Our analysis of the pre/post changes in competence frequencies suggests that the students are improving with regard to purposeful exploration of the model by comparing more than one simulation trial and attending to explanatory factors. This suggests they began to look more closely at the model and to understand the interactions at the micro-level that explained the macro-level phenomenon.

7.4.2 Learning Objective 2: Identify Simplifications Made by a Model

The class average for the pre-test item measuring students' ability to *identify simplifications made by a model* was a score of 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

www.dbooks.org

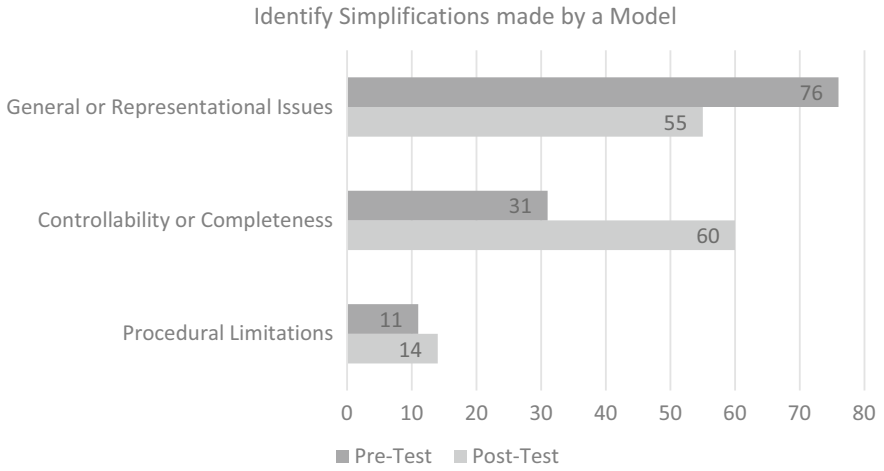


Fig. 7.5 Frequencies of competencies demonstrated in students' responses to the pre- and post-test items assessing their mastery of learning objective 2

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 with regards to this learning objective. To gain a more nuanced understanding of how students developed their abilities to *identify the simplifications made by a model*, we compared the frequencies of competencies they demonstrated in pre- and post-test responses. For ease of coding, we combined competencies of the same score. This is reflected in the bar chart (Fig. 7.5), which illustrates the number of students noting *general or representational limitations*, *controllability or completeness limitations*, and *procedural limitations*, on both pre and post-test.

The frequency decreased from pre- to post-test for *attending to general or representational limitations* and increased from pre- to post-test for both *attending to limitations of controllability or completeness* and *attending to procedural limitations*. Below, we present results of statistical analyses that show whether these changes in frequency may have been the result of students' participation in our computational biology units.

7.4.2.1 Attending to General or Representational Issues

A decrease in *attending to general or representational limitations* of the model suggests fewer students are distracted by surface-level limitations that are inconsequential to the model's ability to simulate reality. An example of a student response that attends to general or representational limitations is "Obviously, sunlight is not a bunch of little sticks raining down." A McNemar's test (without continuity correction) results in a test statistic (χ^2) of 9 and a p-value of 0.0027, which is a significant

effect. This is evidence that engagement in our computational biology curriculum improved students' abilities to *assess a model* by encouraging them to pay less attention to limitations with the model's surface features.

7.4.2.2 Attending to Limitations of Controllability or Completeness

An increase in *attending to limitations of controllability or completeness* suggests students have begun to pay attention to limitations with elements of the model's deeper structure. An example of a student response that attends to limitations of controllability or completeness in a model is "In real life, you cannot add or subtract molecules nor can you adjust the wall positioning." A McNemar's test on our counts (without continuity correction) results in a test statistic (χ^2) of 15.868 and a p-value of less than 0.001, which is a significant effect. This is evidence that engagement in our computational biology curriculum improved students' abilities to *assess a model* by encouraging them to attend to deeper model limitations, such as controllability and completeness limitations.

7.4.2.3 Attending to Procedural Limitations

An increase in *attending to procedural limitations* of the model suggests more students have begun to pay attention to elements of the model's deeper structure. An example of a student response that attends to procedural limitations is "CO_2 might not speed up that much when it absorbs IR light." A McNemar's test (without continuity correction) results in a test statistic (χ^2) of 0.42857 and a p-value of 0.5127, which is not a significant effect. Though we do see an increase in the number of students enacting this practice, there is not enough evidence to say that engagement in our computational biology curriculum improved students' abilities to assess the procedural limitations of a computational model.

Our analysis of the pre/post changes in competence frequencies suggests that students developed expertise in *assessing a model* by shifting their attention from limitations with the surface features of the model to limitations with elements of its deeper structure. More specifically, students shifted from identifying general limitations or limitations with the visual representation to limitations with a model's controllability and completeness.

7.5 Discussion

We have presented findings from a quantitative analysis of 133 9th grade students' written responses to assessments given before and after their participation in three computational biology units. Our results suggest that our curriculum helped students develop a number of important competencies for *exploring a model by changing*

www.dbooks.org

parameters in the interface or code, such as *comparing simulation results across multiple trials* and moving beyond merely describing relationships between a parameter and system behavior, to *attending to explanatory factors* in the model. Our results also suggest that students developed important competencies for *identifying simplifications made by a model*, such as shifting attention from *general and representational limitations* with the model to deeper limitations such as *model completeness and controllability*. While our results are encouraging, we can't rule out the possibility that limitations of our experimental design (such as asymmetries between pre- and post-test items discussed earlier) may have influenced our findings.

Our work is concerned with characterizing students' engagement in computational thinking practices in their secondary science classrooms. It is therefore in conversation with scholarship on the nature of computational thinking and the nature of computational thinking in STEM. Previously, we created a taxonomy of computational thinking practices used by experts in computational STEM disciplines. The findings presented here provide insight into how students can develop expertise with respect to *modeling and simulation practices* by characterizing, at a fine grain-size, the competencies students draw upon when *exploring a model by changing its parameters in the interface or code* and *identifying simplifications made by a model*. Our research program continues to uncover the space of competencies relevant to CT-STEM practices representing all strands of our taxonomy and investigate how these competencies can be developed through engagement with our computationally-enriched science curriculum. In future work, we aim to connect our quantitative treatment with qualitative analysis of student utterances, NetLogo log files, and work.

While the units investigated by this study featured NetLogo, other CT-STEM units (which have been created as part of a larger curricular design effort) feature modeling environments such as Molecular Workbench (Concord Consortium, 2010) and PhET (Perkins et al., 2006). Other units introduce students to computational tools for data analysis and problem solving, such as CoDAP (Finzer, 2016). Exposing students to a diverse range of computational tools is meant to help them develop a flexible set of CT-STEM practices.

In addition to understanding how our curriculum can support students' development of CT-STEM practices, our research aims to understand how engagement in these practices can support students' science content learning. Research already points to the productivity of computational tools for science learning (Guzdial, 1994; National Research Council, 2011; Redish & Wilson, 1993; Repenning, Webb, & Ioannidou, 2010; Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013; Sherin, 2001; Taub, Armoni, Bagno, & Ben-Ari, 2015; Wilensky & Reisman, 2006). As described by restructuration theory, the representational form of knowledge influences how it can be understood. The advance of computational tools has afforded representations that have had profound influence on the way scientists understand phenomena. We argue that these same tools can also be employed in science learning to make complex content more accessible to students, while at the same time broadening engagement with computational thinking.

Acknowledgements We acknowledge, with thanks, the permission of the International Conference on Computational Thinking Education 2017 (CTE 2017) to allow us to extend our previous work, "Swanson, H., Anton, G., Bain, C., Horn, M., & Wilensky, U. (2017). Computational Thinking in the Science Classroom. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education 2017*. Hong Kong: The Education University of Hong Kong", in preparing this book chapter. This work was supported by the Spencer Foundation and the National Science Foundation (CNS-1138461, CNS 1441041, DRL-1640201).

References

Blikstein, P., & Wilensky, U. (2009). An atom is known by the company it keeps: A constructionist learning environment for materials science using agent-based modeling. *International Journal of Computers for Mathematical Learning, 14*(2), 81–119.

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).

Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *Journal of the learning sciences, 13*(1), 15–42.

Concord Consortium. (2010). Molecular workbench. *Java simulations and modeling tools (2004–2013).*

diSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. MIT Press.

Finzer, W. (2016). Common online data analysis platform (CODAP). Emeryville, CA: The Concord Consortium. [Online: concord. org/codap].

Goody, J. (1977). *The domestication of the savage mind*. New York: Cambridge University Press.

Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments, 4*(1), 001–044.

Horn, M. S., & Wilensky, U. (2012). NetTango: A mash-up of NetLogo and Tern. In *AERA 2012*.

Horn, M. S., Brady, C., Hjorth, A., Wagh, A., & Wilensky, U. (2014, June). Frog pond: A code-first learning environment on evolution and natural selection. In *Proceedings of the 2014 Conference on Interaction Design and Children* (pp. 357–360). ACM.

Kaczmarczyk, L., & Dopplick, R. (2014). *Rebooting the pathway to success: Preparing students for computing workforce needs in the United States*. Education Policy Committee, Association for Computing Machinery.

Levy, F., & Murnane, R. (2004). *The new division of labor: How computers are creating the new job market*. Princeton, NJ: Princeton University Press.

Levy, S. T., & Wilensky, U. (2009). Students' learning with the connected chemistry (CC1) curriculum: Navigating the complexities of the particulate world. *Journal of Science Education and Technology, 18*(3), 243–254.

Margolis, J. (2008). *Stuck in the shallow end: Education, race, and computing*. Cambridge: The MIT Press.

Margolis, J., & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. Cambridge: The MIT Press.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: The National Academies Press.

National Research Council. (2011). *Learning science through computer games and simulations*. Washington, DC: The National Academies Press.

Novak, M., & Wilensky, U. (2011). *NetLogo fish tank genetic drift model*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling.

Olson, D. R. (1994). *The world on paper*. New York: Cambridge University Press.

www.dbooks.org

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

Perkins, K., Adams, W., Dubson, M., Finkelstein, N., Reid, S., Wieman, C., et al. (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.

Redish, E. F., & Wilson, J. M. (1993). Student programming in the introductory physics course: MUPPET. *American Journal of Physics, 61,* 222–232.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 265–269).

Sengupta, P., & Wilensky, U. (2009). Learning electricity with NIELS: Thinking with electrons and thinking in levels. *International Journal of Computers for Mathematical Learning, 14*(1), 21–50.

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.

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning, 6*(1), 1–61.

Taub, R., Armoni, M., Bagno, E., & Ben-Ari, M. (2015). The effect of computer science on physics learning in a computational science environment. *Computing Education, 87,* 10–23.

Tinker, R., & Wilensky, U. (2007). *NetLogo Climate Change model*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling.

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

Weintrop, D., Beheshti, E., Horn, M. S., Orton, K., Trouille, L., Jona, K., & Wilensky, U. (2014). Interactive assessment tools for computational thinking in high school STEM classrooms. In D. Reidsma, I. Choi, & R. Bargar (Eds.), *Proceedings of Intelligent Technologies for Interactive Entertainment: 6th International Conference, INTETAIN 2014, Chicago, IL, USA* (pp. 22–25). Springer International Publishing.

Wilensky, U. (1997a). *NetLogo GasLab gas in a box model*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo/models/GasLabGasinaBox.

Wilensky, U. (1997b). *NetLogo wolf sheep predation model*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation.

Wilensky, U. (1997c). *NetLogo AIDS model*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo/models/AIDS.

Wilensky, U. (1999). *NetLogo*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. 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. (2003). Statistical mechanics for secondary school: The GasLab multi-agent modeling toolkit. *International Journal of Computers for Mathematical Learning, 8*(1), 1–41.

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*. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling.

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulations of knowledge disciplines through new representational forms. In J. Clayson & I. Kalas (Eds.), *Proceedings of the Constructionism 2010 Conference*. Paris, France, 10–14 Aug 2010 (p. 97).

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—An embodied modeling approach. *Cognition and instruction, 24*(2), 171–209.

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

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 8
Components and Methods of Evaluating Computational Thinking for Fostering Creative Problem-Solvers in Senior Primary School Education



Siu-Cheung Kong

Abstract The challenge of introducing Computation Thinking (CT) education to K-12 is how to evaluate learners' CT development. This study used information from previous studies to identify essential components and methods for evaluation. A review of literature from 2010 onwards aimed to identify all studies related to CT evaluation to contribute to the development of appropriate evaluation components and methods for senior primary school education. To facilitate the design of a comprehensive evaluation approach, the CT framework of concepts, practices and perspectives developed by Brennan and Resnick (2012) was adopted in this study. This study has two highlights: (1) problem formulation is proposed to be included as a component of CT practices and (2) CT perspectives are refined to include computational identity and programming empowerment as components. CT concepts could be evaluated by test designs with multiple choice questions to identify students' learning outcomes. CT practices could be evaluated by designing rubrics to rate programming projects and using test designs with task-based questions to identify learners' learning outcomes. Finally, CT perspectives could be measured by designing survey instruments and administrating surveys to learners.

Keywords Computational thinking · Creative problem-solver · Evaluation · Senior primary school education · Visual programming

8.1 Introduction

To keep pace with the Fourth Industrial Revolution, which has integrated physical, digital and biological spheres in all aspects of life (World Economic Forum, 2016), it is necessary to nurture the next generation to become creative problem-solvers in the digital era. Computational Thinking (CT) is widely accepted as a fundamental practice for equipping young people to formulate and solve problems in the dig-

S.-C. Kong (✉)
Centre for Learning, Teaching and Technology, The Education University of Hong Kong, 10 Lo Ping Road, Tai Po, Hong Kong
e-mail: sckong@eduhk.hk

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_8

ital world (Computer Science Teachers Association, 2011), and it is unsurprising that many scholars have called for the integration of CT into K-12 education (e.g. Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). To successfully implement CT education in schools, designing appropriate assessing components and methods for evaluating learners is of paramount importance. The National Research Council (2011) pointed out that the purposes of assessing learners' CT development are to evaluate the curriculum, teaching materials and pedagogy, to judge individuals' progress and to manage teacher development and support. Educators are responsible for determining what should be measured and what methods should be used to measure learners' CT development (Duncan & Bell, 2015). Although researchers have explored evaluation components and methods for young learners in recent years, there is no consensus on which are most effective (e.g. Brennan & Resnick, 2012; Grover, Pea, & Cooper, 2015; Zhong, Wang, Chen, & Li, 2016). There is a pressing need to clarify the evaluation components and to propose appropriate approaches and methods to measure learners' CT development.

This study used information from previous studies to identify and propose essential components and methods for evaluating senior primary school learners' CT development. In terms of CT concepts, proposed components to be evaluated include loops, conditionals, sequences, parallelism, data structures, mathematics operators, functions and Boolean logic, event handling, procedures and initialisation. These can be measured by test designs with multiple choice questions to identify students' learning outcomes. In terms of CT practices, proposed components include problem formulating, planning and designing, abstracting and modularising, algorithmic thinking, reusing and remixing, being iterative and incremental, and testing and debugging; these could be evaluated by designing rubrics to rate programming projects and using test designs with task-based questions to identify learners' learning outcomes. In terms of CT perspectives, proposed components include computational identity, programming empowerment and the perspectives of expressing, connecting and questioning, which could be measured by designing survey instruments and administering surveys to learners.

8.2 Background

8.2.1 Computational Thinking

CT is not a new idea; Papert (1980) advocated that children can learn to think expressively about thinking and can foster procedural thinking through programming. He argued that 'computational technology and computational ideas can provide children with new possibilities for learning, thinking, and growing emotionally as well as cognitively' (Papert, 1980, pp. 17–18). The idea of CT has gained global awareness and discussion after Wing's introduction in 2006. She suggested that everyone—not only computer scientists—should learn and use CT (Wing, 2006). According to Wing, CT

www.dbooks.org

includes problem-solving, systems design and understanding human behaviour, and it draws on the concepts that are fundamental to computer science. CT is further defined 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, p. 1). Aho (2012) echoed the viewpoint that CT refers to problem formulation and that solutions can be represented as computational steps and algorithms. In sum, CT is a process of formulating and solving computational problems.

Because CT helps equip the next generation with problem-solving skills, a number of scholars have called for the integration of CT into K-12 education (e.g. Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). Researchers have varying ideas with respect to K-12 curriculum because of their diverse interpretations of CT. One viewpoint is that CT should be promoted by means of programming courses because it can be incited by programming knowledge (Araujo, Andrade, & Guerrero, 2016) and learners can demonstrate their CT practices through programming tools. For example, Grover and Pea (2013) introduced nine essential components of CT practices: abstraction and pattern generalisations; systematic processing of information; symbol systems and representations; algorithmic notions of flow of control; structured problem decomposition; iterative, recursive and parallel thinking; conditional logic; efficiency and performance constraints and debugging and systematic error detection. Another viewpoint is that CT practices should be elicited by other subjects and disciplines. For instance, the International Society for Technology in Education (ISTE) emphasises data representations and working with automated solutions and simulations (Barr, Harrison, & Conery, 2011). ISTE's framework does not emphasise the building of artefacts through programming environments, as learners can display CT practices through data representations and can work with automated solutions and simulations (Lye & Koh, 2014). For a CT curriculum that adopts the constructionism approach (Papert, 1980), the acquisition of programming concepts and practices through programming is considered as the most effective way to learn CT. This study adopts this approach to propose measures to evaluate learners' understanding of CT development instead of using ISTE's framework.

8.2.2 The Adopted Framework for Computational Thinking Evaluation

This study adopted a CT framework in the context of programming proposed by Brennan and Resnick (2012) as the evaluation framework. The framework consists of three dimensions: CT concepts, CT practices and CT perspectives. The dimension of CT concepts refers to the computational concepts that learners develop in programming, the dimension of CT practices refers to the problem-solving practices that learners demonstrate repeatedly in the programming process and the dimension of CT perspectives refers to learners' understanding of themselves and their relationships

to others and the technological world that they develop by expressing, connecting and questioning during programming. According to the National Research Council (2012), in the twenty-first century, it is important to assess a person's competence in the cognitive, intrapersonal and interpersonal domains. CT concepts and practices represent the cognitive domain, such as learners' programming knowledge and practices, while CT perspectives represent the intrapersonal and interpersonal domains, such as learners' expression using programming tools and connection to the digitalised world. Lye and Koh (2014) pointed out that most studies focused on CT concepts, and fewer on practices and perspectives. This framework helps to provide a comprehensive picture for evaluating CT development in schools.

This framework is also appropriate for young learners to begin learning CT. In Piaget's model of the four stages of development, learners in the 7–11 age groups are developing operational thinking (Piaget, 1972). Thus, senior primary school learners in the 9–11 age groups are at the suitable starting point to learn basic programming knowledge. As the context of this framework is in visual programming environments such as Scratch and App Inventor, it enables learners to 'learn programming concepts and practice CT abilities while avoiding the syntax hurdle associated with text-based programming languages' (Araujo et al., 2016, p. 6). Within this context, Brennan and Resnick's framework facilitates the design of evaluation components and methods to capture the progress and achievements of learners studying CT in senior primary education.

8.3 Methodology

To facilitate the promotion of CT education in K-12 schools, it is necessary to conduct a comprehensive review of literature to identify the evaluation components and methods of CT for young learners. To achieve the aim of this study, two research questions are raised: (1) What evaluation components should be included in the CT concepts, practices and perspectives dimensions according to the CT framework proposed by Brennan and Resnick (2012) in K-12? (2) What methods are appropriate for evaluating the CT components in each dimension in this context? As each study has put forth its own components and methods for evaluating learners' CT development, this study used information from previous studies to identify the essential components and methods for evaluation. This review of the literature from 2010 to the present aimed to identify every study related to programming and CT evaluation to contribute to the proposal of appropriate evaluation components and methods for senior primary school education. Table 8.1 summarises the 24 studies reviewed in this study.

www.dbooks.org

Table 8.1 List of 24 studies reviewed in this study

	Study	Dim[a]		Study	Dim[a]		Study	Dim[a]
1.	Brennan and Resnick (2012)	1, 2, 3	9.	Grover, Cooper, and Pea (2014)	1, 2	17.	Ruf, Mühling, and Hubwieser (2014)	1, 3
2.	Burke (2012)	1, 2	10.	Grover et al. (2015)	1, 2, 3	18.	Seiter and Foreman (2013)	1, 2
3.	Denner, Werner, Campe, and Ortiz (2014)	2, 3	11.	Kukul and Gökçearslan (2017)	3	19.	Sherman and Martin (2015)	1, 2
4.	Duncan and Bell (2015)	2, 3	12.	Maguire, Maguire, Hyland, and Marshall (2014)	3	20.	Werner, Denner, Campe, and Kawamoto (2012)	2, 3
5.	Ericson and McKlin (2012)	1, 3	13.	Meerbaum-Salant, Armoni, and Ben-Ari (2013)	1	21.	Wilson, Hainey, and Connolly (2012)	1
6.	Fessakis, Gouli, and Mavroudi (2013)	2	14.	Mueller, Beckett, Hennessey, and Shodiev (2017)	2	22.	Wolz, Stone, Pearson, Pulimood, and Switzer (2011)	3
7.	Giordano and Maiorana (2014)	1, 3	15.	Rodriguez, Kennicutt, Rader, and Camp (2017)	2	23.	Zhong et al. (2016)	1, 2
8.	Gouws, Bradshaw, and Wentworth (2013)	2	16.	Román-González, Pérez-González, and Jiménez-Fernández (2016)	1, 2	24.	Zur-Bargury, Pârv, and Lanzb erg (2013)	1

[a]The numbers refer to the dimensions discussed in the reviewed literature, according to the CT framework proposed by Brennan and Resnick (2012). '1' refers to CT concepts, '2' refers to CT practices and '3' refers to CT perspectives

8.4 Results and Discussion Based on Literature Review

Grover et al. (2015, p. 199) wrote that 'a system of assessments is beneficial to get a comprehensive picture of learners' CT learning.' In other words, there is no single evaluation component or method that can entirely measure learners' CT achievements. Researchers, therefore, strive to develop both quantitative and qualitative approaches to evaluate learners' learning outcomes. This section sheds light on the essential evaluation components and assessment methods of each of the three dimensions proposed by Brennan and Resnick (2012): CT concepts, CT practices and CT perspectives.

8.4.1 CT Concepts

8.4.1.1 Literature Review Results

The 14 studies on evaluating programming concepts in block-based environments provided 9 CT concepts to be included: (1) repetition/loops/iteration, (2) conditionals, (3) data structures, (4) sequences, (5) parallelism/concurrency, (6) mathematical operators, functions and Boolean logic, (7) event handling, (8) procedures and (9) initialisation. Table 8.2 summarises components of CT concepts that were evaluated by past studies and the tabulated results were sorted according to the frequency with which they were discussed.

Brennan and Resnick (2012) pointed out that the first seven concepts in Table 8.2 were useful in the Scratch programming environment and stated that learners should be able to transfer these CT concepts to other programming and non-programming contexts. The reviewed studies generally concurred on the importance of these seven components, in particular, all 14 studies agreed that repetition (or loops or iteration) is an essential component in evaluating CT concepts. Conditionals were identified by 11 studies as a core component of CT concepts. Ten studies put forth data as a key programming concept, which includes important knowledge such as variables and lists. The concept of sequences and parallelism (or concurrency), including sending or passing messages and handling coordination, was mentioned in eight studies. Eight studies also suggested that learners should understand the concepts of mathematical operators such as 'addition, subtraction, multiplication, division, as well as functions like sine and exponents' (Brennan & Resnick, 2012, pp. 5–6) and Boolean logic operations such as 'AND, OR, NOT'. Six studies investigated learners' ability to handle events in programming. In addition, the ability to apply procedures in programming tasks was identified as a component of CT concepts in two studies. It helps to avoid the repetition of codes and duplication of commands (Marji, 2014). One study attempted to evaluate learners' ability to deal with initialisation.

The 14 evaluation studies used both quantitative and qualitative methods to measure learners' understanding of CT concepts. Table 8.3 summarises the methods

www.dbooks.org

Table 8.2 Components of CT concepts in past studies

Component	Study	Frequency
1. Repetition/loops/iteration	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Román-González et al. (2016), Ruf et al. (2014), Seiter and Foreman (2013), Sherman & Martin (2015), Wilson et al. (2012), Zhong et al. (2016), Zur-Bargury et al. (2013)	14
2. Conditionals	Brennan and Resnick (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Ruf et al. (2014), Seiter and Foreman (2013), Sherman and Martin (2015), Wilson et al. (2012), Zur-Bargury et al. (2013)	11
3. Data structures (e.g. variables, lists)	Brennan and Resnick (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Seiter and Foreman (2013), Sherman & Martin (2015), Wilson et al. (2012), Zhong et al. (2016)	10
4. Sequences	Brennan and Resnick (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Zhong et al. (2016), Seiter and Foreman (2013), Wilson et al. (2012)	8
5. Parallelism/concurrency (e.g. messages, coordination)	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Meerbaum-Salant et al. (2013), Seiter and Foreman (2013), Wilson et al. (2012), Zhong et al. (2016)	8
6. Mathematical operators, functions and Boolean logic	Brennan and Resnick (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Román-González et al. (2016), Seiter and Foreman (2013), Wilson et al. (2012), Zhong et al. (2016)	8
7. Event handling	Brennan and Resnick (2012), Burke (2012), Ericson and McKlin (2012), Giordano and Maiorana (2014), Seiter and Foreman (2013), Wilson et al. (2012)	6
8. Procedures	Giordano and Maiorana (2014), Seiter and Foreman (2013)	2
9. Initialisation	Meerbaum-Salant et al. (2013)	1

Table 8.3 Methods used by studies to evaluate CT concepts

Method	Study	Frequency
1. Test designs with multiple choice type questions in programming context	Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013), Román-González et al. (2016), Ruf et al. (2014), Zur-Bargury et al. (2013)	8
2. Task/project rubrics	Seiter and Foreman (2013), Sherman and Martin (2015), Wilson et al. (2012), Zhong et al. (2016)	4
3. Interviews	Brennan and Resnick (2012), Grover et al. (2014, 2015), Meerbaum-Salant et al. (2013)	4
4. Project analysis	Brennan and Resnick (2012), Burke (2012)	2
5. Observations	Burke (2012), Meerbaum-Salant et al. (2013)	2
6. Reflection reports	Zhong et al. (2016)	1

by studies used to evaluate CT concepts. The tabulated results are sorted according to the frequencies with which the methods were used. In terms of quantitative approaches, most of the studies measured learners' understanding of CT concepts using test items, most of which were designed with multiple choice questions in the programming context (Ericson & McKlin, 2012; Ruf, Mühling, & Hubwieser, 2014; Zur-Bargury et al., 2013). Task and project rubrics were also commonly used so that teachers could assess learners' project work with scoring guidelines.

Among the studies using qualitative approaches, interviews were the most common method of evaluating CT concepts. Interviews are conducted to understand the CT concepts that learners use to complete programming tasks or projects. Project analysis was another method used to evaluate learners' development in CT concepts. Evaluators used the Scrape tool to analyse the programming blocks of projects. Scrape can provide a record of the CT concepts that the learner utilised in his/her projects (Brennan & Resnick, 2012; Burke, 2012). Studies also observed programming lessons to investigate whether learners could correctly handle the CT concepts in programming activities. One study asked learners to write reflective reports on what kinds of CT concepts they used to complete their programming tasks and projects (Zhong et al., 2016).

8.4.1.2 Proposed Evaluation Components and Methods

Duncan and Bell (2015) analysed the computing curricula for primary school in England and recommended that key stage 2 learners in the 7–11 year age group acquire

www.dbooks.org

Table 8.4 Proposed evaluation components of CT concepts at the primary school level

Component of CT concepts
1. Loops
2. Conditionals
3. Sequences
4. Parallelism
5. Data structures such as variables and lists
6. Mathematics operators, functions and Boolean operators
7. Event handling
8. Procedures
9. Initialisation

concepts such as loops, conditionals, sequences and variables. One study found that loops, conditionals, variables and operators were the fundamental programming concepts used by children in the Scratch programming environment (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). Wilson et al. (2012) found that over half of children's game-making projects used concepts such as loops, conditionals, sequences and parallelism. Because event handling is commonly used in block-based programming environments such as Scratch and App Inventor, the concept of event handling should be included (Brennan & Resnick, 2012). Although some may believe that procedure and initialisation are difficult for senior primary learners, they are essential concepts in programming. Procedure can avoid the repetition of codes and duplication of commands (Marji, 2014), and initialisation helps a programme reach its initial state. Therefore, if teachers introduce procedure and initialisation in class, they can consider assessing learners' abilities regarding these two concepts. In line with the above discussion and the results tabulated in Table 8.2, the following CT concepts should be evaluated in the K-12 CT curriculum: (1) loops, (2) conditionals, (3) sequences, (4) parallelism, (5) data structures such as variables and lists, (6) mathematical operators, functions and Boolean operators, (7) event handling, (8) procedures and (9) initialisation. Table 8.4 summarises the proposed CT concepts at the primary school level.

Because CT concepts are made up of different kinds of components, they can be evaluated by an aggregation of learners' achievement across components. Therefore, CT concepts can be evaluated by designing quantitative test instruments with itemised pieces that each evaluates a particular component. Evaluators can use these test instruments to measure the learning outcomes of learners and to administer pre- and post-test designs to measure learners' progression. Evaluators can also assess whether learners apply CT concepts appropriately by looking into their tasks/projects with the rubrics. Quantitative data can thus be obtained by accumulating the scores of learners generated from the rubrics. It is proposed that test designs with multiple choice questions be used to measure learners' progression for large numbers of learners. Evaluators can show programming segments in the test items and ask questions related to the segments, such as finding the output from the programming segment,

Please read the blocks below. If the answer is 'No', how will the programme respond?



A. Say "Great!"
B. Say "Cheer up!"
C. I don't know
D. No response
E. An error will occur

Fig. 8.1 An example of testing the CT concept of conditionals in the form of a multiple choice question

which can demonstrate learners' understanding of CT concepts. Figure 8.1 shows an example of testing conditionals in the form of a multiple choice question.

CT concepts can also be evaluated by qualitative methods such as interviews, project analyses, observations and reflection reports. Researchers found that when learners were asked about how parts of their code work, their explanations showed large conceptual gaps (Grover et al., 2014). Therefore, interviews with learners can show how they explain their work. It is also suggested that evaluators can evaluate learners' CT concepts by using automated tools to analyse 'the number, the range, and the frequency' of different programming blocks being used in their projects (Burke, 2012). These blocks could provide information on learners' development in CT concepts. In addition, classroom observations can enable evaluators to determine how learners use CT concepts to tackle computational tasks. Asking primary school learners to write simple reflective reports on what kinds of CT concepts they used in completing their programming tasks and projects is another alternative. Because qualitative approaches require time and effort to explore learners' learning outcomes, they should be regarded as supplementary for evaluators seeking to comprehend the CT concepts of learners. Table 8.4 summarises the proposed evaluation components of the CT concepts at the primary school level.

www.dbooks.org

Table 8.5 Components of CT practices proposed by research studies

Component	Study	Frequency
1. Abstraction/abstracting, modelling/abstracting and modularising	Brennan and Resnick (2012), Denner et al. (2014), Gouws et al. (2013), Grover et al. (2015), Mueller et al. (2017), Rodriguez et al. (2017), Seiter and Foreman (2013), Sherman and Martin (2015), Werner et al. (2012), Zhong et al. (2016)	10
2. Algorithmic thinking	Denner et al. (2014), Duncan and Bell (2015), Gouws et al. (2013), Mueller et al. (2017), Rodriguez et al. (2017), Seiter and Foreman (2013), Werner et al. (2012)	7
3. Testing and debugging	Brennan and Resnick (2012), Burke (2012), Fessakis et al. (2013), Grover et al. (2015), Mueller et al. (2017), Román-González et al. (2016), Zhong et al. (2016)	7
4. Being incremental and iterative	Brennan and Resnick (2012), Grover et al. (2015), Mueller et al. (2017), Zhong et al. (2016)	4
5. Problem decomposition	Grover et al. (2015, 2014), Mueller et al. (2017), Seiter & Foreman (2013)	4
6. Planning and designing	Burke (2012), Zhong et al. (2016)	2
7. Reusing and remixing	Brennan and Resnick (2012), Mueller et al. (2017)	2

8.4.2 CT Practices

8.4.2.1 Literature Review Results

Fifteen studies in the literature provide seven CT practices: (1) abstraction/abstracting and modelling/abstracting and modularising, (2) algorithmic thinking, (3) testing and debugging, (4) being incremental and iterative, (5) problem decomposition, (6) planning and designing and (7) reusing and remixing. Table 8.5 summarises the components of CT practices identified by past studies, and the tabulated results were sorted according to the frequency with which they were discussed.

Abstraction is one of the most fundamental practices in CT (Wing, 2006). Ten studies are related to learners' ability to think abstractly. Brennan and Resnick (2012, p. 9) suggested that modularisation ability should be merged with abstraction so that learners can avoid complexity and 'build something large by putting together collections of smaller parts'. Modelling, another skill that is often linked with abstracting, allows learners to 'organize data, structure their thoughts, describe relationships, and analyse proposed designs' (Voland, 1999, p. 215). Seven of the studies regarded algo-

rithmic thinking as the foundation of CT practices. Learners are required to define the steps and develop instructions to solve a problem. Seven studies also suggested that testing and debugging are the core of CT practices because it is 'rare for a program to work correctly the first time it is tried, and often several attempts must be made before all errors are eliminated' (Peterson, 2002, p. 93). Four past studies suggested that problem decomposition and the practices of being incremental and iterative are indispensable in the programming process. Learners should learn to repeatedly 'develop a little bit, then try it out, then develop more' until the programme is complete (Brennan & Resnick, 2012, p. 7). Problem decomposition involves breaking down problems into smaller, more manageable tasks (Waller, 2016). Several researchers found that planning and designing were part of CT practices. They were used to investigate whether learners plan their solutions before they write the code or use trial and error during programming. Additionally, researchers noted that the reuse and remix of the works of other programmers are crucial in the online communities of Scratch and Alice (Brennan & Resnick, 2012), and they encouraged novices to produce more complicated creations by building on existing projects or ideas (Brennan & Resnick, 2012).

For evaluation methods, because of the difficulties in finding out how learners tackle problems during programming, evaluators use both quantitative and qualitative methods to measure learners' CT practices. Table 8.6 summarises the methods adopted by the studies to evaluate CT practices. The tabulated results are sorted by the frequency with which the methods were used. Seven studies used task/project rubrics to evaluate learners' CT practices, as teachers marked the programming outcomes of tasks or projects based on rubrics to evaluate the CT practices of learners. Four projects used tests with task-based questions in coding and non-coding contexts to assess learners' proficiency in CT practices. The qualitative methods proposed are favourable for demonstrating their learning process. Four studies proposed the use of interviews to understand learners' CT practices, as interviews enable teachers to understand learners' thoughts behind the code (Grover & Pea, 2013). Two studies proposed the use of observations: researchers observe in classrooms, take notes and videorecord the entire programming process to understand learners' CT practices (Grover & Pea, 2013). One study used learners' reflection reports to understand their programming practices. Reflection reports are self-assessments that are distributed to learners after they finish a task or a project (Zhong et al., 2016), in which they write out how they accomplished their problem-solving tasks.

8.4.2.2 Proposed Evaluation Components and Methods

Programming is not a simple process, and it might not have a clear order. However, the components of CT practices can be divided into two categories: design practices and programming practices. In the design practices category, learners design programmes. This category includes three practices: problem decomposition, abstracting and modularising, and algorithmic thinking. Because people often perceive planning as a 'highly specialized skill for solving problems' that requires scarce resources

www.dbooks.org

Table 8.6 Methods adopted by studies to evaluate CT practices

Method	Study	Frequency
1. Task/project rubrics	Denner et al. (2014), Rodriguez et al. (2017), Román-González et al. (2016), Seiter and Foreman (2013), Sherman and Martin (2015), Werner et al. (2012), Zhong et al. (2016)	7
2. Tests designed with task-based questions	Duncan and Bell (2015), Gouws et al. (2013), Grover et al. (2014, 2015)	4
3. Interviews	Brennan and Resnick (2012), Grover et al. (2014, 2015), Mueller et al. (2017)	4
4. Observations	Burke (2012), Fessakis et al. (2013)	2
5. Reflection reports	Zhong et al. (2016)	1

(Lawler, 1997), planning and designing are not recommended for inclusion in the programming process. Learners are supposed to decompose problems into sub-tasks first (Waller, 2016). The practice of modularising is recommended to be merged with abstracting. Learners will be able to connect the parts of the whole so that they can test and debug different parts of the programme incrementally (Brennan & Resnick, 2012). The Center for Computational Thinking, Carnegie Mellon (2010) asserted that algorithmic thinking is essential, as it helps learners to produce efficient, fair and secure solutions. In sum, in line with the results in Table 8.5, problem decomposition, abstracting and modularising, and algorithmic thinking are essential CT practices for evaluation at this stage.

In the programming practices category, learners implement their designs to produce a concrete programme artefact. This stage includes three practices: reusing and remixing, being iterative and incremental, and testing and debugging. The first practice helps novices create their own programmes by building on others' works and developing their own works incrementally (Brennan & Resnick, 2012). The practice of being iterative and incremental is tied to testing and debugging: programmers have to develop part of the programme and test it to ensure that it works; they then repeat these steps and continue to develop the programme. In sum, in line with the results in Table 8.5, reusing and remixing, being iterative and incremental, and testing and debugging are essential CT practices for evaluation at this stage.

Problem formulating is proposed to be included in the CT practices component in the study of this paper. Although Cuny et al. (2010, p. 1) defined CT as the 'thought processes involved in formulating problems and their solutions', none of the reviewed studies proposed assessing learners' abilities in problem formulation. In the early twentieth century, Einstein argued that raising questions was more important than solving problems. In his classic study, *The Evolution of Physics*, he wrote, 'the formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill. To raise new questions, new possibilities, to regard old problems from a new angle require creative imagina-

Table 8.7 Proposed evaluation components of CT practices at the primary school level

Component of CT practices
1. Problem formulating
2. Problem decomposition
3. Abstracting and modularising
4. Algorithmic thinking
5. Reusing and remixing
6. Being iterative and incremental
7. Testing and debugging

tion and marks real advances in science' (Einstein & Infeld, 1938, p. 92). Because learners' creativity can be demonstrated by formulating creative problems, there is a need to add this component to CT practices to produce creative problem-solvers in the digitalised world. The aims of CT practices are thus to develop both creative thinkers and problem-solvers (Brugman, 1991). Therefore, this study proposes to include 'problem formulating' as one of the components of CT practices. Table 8.7 summarises the proposed evaluation components of the CT practices of this study: (1) problem formulating, (2) problem decomposition, (3) abstracting and modularising, (4) algorithmic thinking, (5) reusing and remixing, (6) being iterative and incremental and (7) testing and debugging.

Like CT concepts, CT practices involve several components; these can be measured by an aggregation of learners' achievement across components. One quantitative method to evaluate CT practices is designing rubrics to assess the final programming projects of learners. The criteria of the rubrics are the components of CT practices, and there is a need to design descriptors of performance levels for each CT practice. These rubrics help evaluators to review learners' abilities to formulate and solve problems. In addition to using rubrics to assess the CT practices of learners, SRI International proposed a task-based approach to assess CT practices. SRI International also published a report titled 'Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science' (Bienkowski, Snow, Rutstein, & Grover, 2015), which presented an overview of four CT practice design patterns and two supporting design patterns. The CT practice design patterns include analysing the effects of developments in computing, designing and implementing creative solutions and artefacts, designing and applying abstractions and models, and supporting design patterns analysing learners' computational work and the work of others. SRI International developed focal knowledge, skills and other attributes for each CT practice, which illustrate the core abilities that learners need to acquire and what should be assessed. Its report provided direction for evaluating CT practices using the task-based approach. In this approach, evaluators' first step is to define the fundamental abilities of each CT practice. The next step is to design tasks that include a scenario and to ask questions about the scenarios to test learners' abilities related to CT practice. Learners need to understand the scenarios and to answer questions about selecting options, organising sequences of activities or performing

www.dbooks.org

1. Scenario

An app developer developed a mobile location tracking app that can show the location of a place on a map using latitude and longitude. However, a user reported that the app does not show the location correctly on the map. Please identify the reason(s) why the error occurred.

- ☐ (a) The mobile app developer was unable to find the correct latitude and longitude of the location
- ☐ (b) The mobile app developer was unable to mark the location correctly on the map
- ☐ (c) The mobile app developer committed both mistakes (a) and (b)

Fig. 8.2 An example of testing the CT practice of testing and debugging using a task with a scenario

categorisations to demonstrate their abilities. For example, one of the focal skills of the CT practice of testing and debugging practice is the ability to explain the cause of an error. Figure 8.2 demonstrates an example of testing learners' understanding of testing and debugging using a task with a scenario.

With this approach, tests with sets of task-based questions can be designed to evaluate learners' abilities with the various components of CT practices. Pre- and post-test designs with task-based questions can enable evaluators to identify learners' progression in CT practices.

Measuring learners' CT practices by directly analysing their processes of designing and building programmes requires great effort. Qualitative approaches such as observing learners in designing and building programmes are one such method. Another is to interview them after they formulate, design and build their programmes and to ask them semi-structured questions. It is difficult to ask primary school learners to write detailed reflection reports on how they formulate, design and build their programmes; however, evaluators can ask learners to write simple reflection reports on how they can improve their programmes after solving their computational tasks. These qualitative methods can serve as supplements for evaluators seeking to understand what practices learners used in their programming tasks. Table 8.7 summarises the proposed evaluation components of CT practices at the primary school level.

8.4.3 CT Perspectives

8.4.3.1 Literature Review Results

The literature review indicated not only that CT concepts and practices are of paramount importance in measuring learners' CT development, but learners' perspectives on learning programming are also vital. Brennan and Resnick (2012) proposed three kinds of perspectives in the programming process—expressing, connecting and questioning—to investigate learners' understanding of themselves and their relationships to others and the technological world. Based on the literature review, CT perspectives include not only Brennan and Resnick's (2012) suggestions

Table 8.8 Components of CT perspectives proposed to be evaluated by research studies

Component	Study	Frequency
1. Attitudes such as interest in programming and computing	Denner et al. (2014), Duncan and Bell (2015), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2015), Maguire et al. (2014), Ruf et al. (2014), Werner et al. (2012), Wolz et al. (2011)	9
2. Confidence in programming and computing, programming self-efficacy and programmers	Denner et al. (2014), Giordano and Maiorana (2014), Kukul and Gökçearslan (2017), Maguire et al. (2014), Werner et al. (2012), Wolz et al. (2011)	6
3. Expressing, connecting and questioning	Brennan and Resnick (2012)	1

(i.e. expressing, connecting and questioning) but also learners' motivation beliefs, namely, value and expectancy (Wigfield & Eccles, 2000; Chiu & Zeng, 2008), in their understanding of themselves. Value refers to learners' intrinsic motivations, such as their attitudes towards and interest in learning programming. Expectancy refers to learners' programming confidence, which includes their programming self-efficacy and self-concept. Programming self-efficacy is 'people's judgments of their capabilities to organize and execute courses of action required to attain designated types of performance' in the context of programming (Bandura, 1986, p. 391; Kong, 2017), while programming self-concept is the belief in one's programming competence (Chiu & Zeng, 2008). Previous studies indicated that researchers investigated both learners' motivation beliefs regarding learning programming and their perspectives on the technological world.

The 10 related studies in the literature identify three components of CT perspectives: (1) attitudes such as interest in programming and computing, (2) confidence in programming and computing, programming self-efficacy and programming competence and (3) expressing, connecting and questioning. Table 8.8 summarises the components of CT perspectives proposed to be evaluated by previous studies. The tabulated results are sorted according to the frequency with which they were discussed.

Most researchers agree that learners' attitudes, such as their interest in programming, should be included in this evaluation dimension. Researchers have focused on learners' programming confidence, self-efficacy and competence, and most have used quantitative instruments to measure learners' CT perspectives, although some have used qualitative data to analyse their attitudes. Table 8.9 summarises the methods adopted by studies to evaluate CT perspectives. Surveys are commonly used, with five-point Likert scales used most frequently (e.g. Ericson & McKlin, 2012; Ruf et al., 2014). Conducting surveys before and after a learning programming might facilitate the investigation of learners' CT perspectives. Qualitative methods such

www.dbooks.org

Table 8.9 Methods adopted by studies to evaluate CT perspectives

Method	Study	Frequency
1. Survey	Denner et al. (2014), Duncan and Bell (2015), Ericson and McKlin (2012), Giordano and Maiorana (2014), Grover et al. (2015), Kukul and Gökçearslan (2017), Maguire et al. (2014), Ruf et al. (2014), Werner et al. (2012), Wolz et al. (2011)	10
2. Interview	Brennan and Resnick (2012)	1

as interviews can provide a more in-depth understanding of learners' perspectives towards programming because programming courses and related activities can be reviewed.

8.4.3.2 Proposed Evaluation Components and Methods

To fully understand learners' intrinsic values regarding programming, asking whether they like or dislike programming is far from sufficient. Deci and Ryan (1985, p. 34) argued that 'when people are intrinsically motivated, they experience interest and enjoyment, they feel competent and self-determined'. Learners' interest in programming can be assessed by their engagement in programming activities, sense of affiliation and career orientation. In addition to their attitudes, self-actualisation indicates whether learners feel competent after learning programming. These four factors can be used to explore learners' computational identity, which they form through networking and sharing knowledge and experience in programming activities.

Wenger (1998, pp. 4–5) argued that engagement is 'the most immediate relation to a practice-engaging in activities, doing things, working alone or together, talking, using and producing artifacts', and that it creates 'an identity of participation or non-participation'. When learners are more interested in programming tasks, their engagement in activities is deeper. Learners' self-formation through engagement is thus a critical factor that can directly reflect their interest in and enjoyment of learning programming. Measuring learners' sense of affiliation can also indicate whether they have a feeling of belongingness regarding their interest in programming. They can then develop a computational identity by the 'participation of sharing' (Gee, 2000).

It is believed that learners' interests affect their career choices. The construct of career orientation explores learners' motivation to work with people who share the same interests. Learners' self-actualisation is another crucial subcomponent. According to Maslow (1943, p. 382), self-actualisation is 'the desire for self-fulfilment, namely, to the tendency for one to become actualized in what he is potentially'. In other words, it is the development of competencies in knowledge, attitudes, skills and character (Huitt, 2007). In sum, these factors can be used to extensively investigate learners' intrinsic motivation, especially in their programming interest and competence. Thus, computational identity is proposed to be measured as the first

Table 8.10 Proposed evaluation components of CT perspectives at the primary school level

Component of CT perspectives
1. Computational identity
2. Programming empowerment
3. Perspectives of expressing, connecting and questioning

component of CT perspectives in this study. If learners have positive perceptions regarding these four factors, then they have a strong computational identity.

Concerning expectancy, programming empowerment, the second component of the CT perspectives proposed in this study, can help evaluate learners' programming confidence more comprehensively. Borrowed from the concept of digital empowerment (Makinen, 2006), programming empowerment can be defined as a person's experiences creating and designing programmes that enable them to tackle real-life problems and confidently participate in the digital world. Learners' belief in their competence to acquire the necessary concepts and practices to handle programming tasks is the primary factor that evaluators must measure. Programming empowerment also emphasises the meaningfulness, impact and creativity of programming, where meaningfulness refers to a person's perceived value—that is, the importance of their programming—and impact is a person's perception of the influence of the programming. Because this study proposes evaluation components to foster creative problem-solvers, creativity can be used to measure learners' beliefs in their ability to produce new ideas in the digitalised world through programming and related digital technologies. In sum, learners will contribute confidently in the digital world if they feel empowered concerning their programming experiences.

The third component is a broader horizon that stresses learners' relationships with others and the technological world. It explores how learners use their innovative thinking after learning programming to contribute to society through expressing, connecting and questioning the digitalised world (Brennen & Resnick, 2012). Expressing means learners' abilities to express themselves by creating with programming. Connecting means learners' understanding of the value of creating with and for others by programming, and it highlights the linkage between programming and real life. Questioning means that learners are empowered to ask questions about the technological world based on their programming experiences. It also means that learners are empowered to formulate problems in the computational context. The idea of questioning could also be expanded to formulate problems before programming. Learners are encouraged to think of questions that could be solved computationally.

As mentioned, CT perspectives represent a person's competence in intrapersonal and interpersonal domains. Therefore, this study proposed the evaluation of learners' (1) computational identity, (2) programming empowerment and (3) perspectives of expressing, connecting and questioning. Table 8.10 shows the proposed evaluation components of the CT perspectives of this study.

It is suggested that CT perspectives can be evaluated by well-designed survey instruments. This study proposed the design of survey instruments to measure

www.dbooks.org

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Computational Identity					
1. I feel associated with my classmates when participating in computer programming activities with them.	○	○	○	○	○
2. Learning computer programming with my classmates gives me a strong sense of belonging.	○	○	○	○	○
Programming Empowerment					
1. I have confidence in my ability to use digital technologies.	○	○	○	○	○
2. I can solve problems with digital technologies.	○	○	○	○	○
Perspectives of expressing, connecting, and questioning					
1. I feel excited to express new ideas through programming.	○	○	○	○	○
2. I think carefully about potential problems in the process of programming.	○	○	○	○	○

Fig. 8.3 Sample items of the computational identity, programming empowerment and perspectives of expressing, connecting and questioning survey instruments

(1) computational identity, (2) programming empowerment and (3) perspectives of expressing, connecting and questioning. These surveys should be conducted at different time points to capture the development of learners' CT perspectives. Learners are required to rate their agreement or disagreement with the statements in these instruments, and it is convenient to analyse a large number of learners' replies using quantitative methods. Figure 8.3 shows sample items for the computational identity, programming empowerment and perspectives of expressing, connecting and questioning survey instruments. Qualitative methods such as interviews can be used to obtain more details on learners' perspectives of programming, but these require time and effort.

8.5 Conclusion

Every country/region requires creative problem-solvers who can contribute to every aspect of life in the digitalised world. CT is a way to cultivate creativity and problem-solving skills in young people. Evaluation is important to facilitate the implementation of CT education in schools because it enables teachers to identify the progress and outcomes of learners and help them attain their learning goals. Based on the literature review, this study proposed nine components of CT concepts, seven components of CT practices and three components of CT perspectives for evaluation.

As the goal of the CT curriculum is to nurture creative problem-solvers, it is essential to evaluate learners' abilities to formulate and solve problems in the computational context. This study has two highlights. First, this study proposed 'problem formulating' as an additional component of evaluating CT practices to encourage learners to raise questions before they programme. Problem formulation is also

closely linked with questioning in CT perspectives, which means that learners are empowered to formulate problems in the computational context. In addition, this study discussed the ideas of computational identity and programming empowerment and proposed them as components for evaluating the perspectives of expressing, connecting and questioning proposed by Brennan and Resnick (2012) to capture learners' motivation beliefs in a more comprehensive manner.

Past studies indicated that no single method could effectively measure learners' CT development in the three dimensions; multiple methods are needed to evaluate learning outcomes. To assess a large number of learners' CT development, quantitative methods are more feasible in terms of resources. This study proposed the design of tests with multiple choice questions to evaluate the CT concepts of learners. Evaluators could also assess learners' CT concepts by analysing their programming tasks and projects with rubrics. Qualitative methods such as interviews, project analyses, observations and simple reflection reports can be conducted as supplements to quantitative approaches. CT practices can be evaluated by measuring programming project outcomes with rubrics and the design of tests with task-based questions. Qualitative methods such as interviews, observations and simple reflection reports can be conducted given the in-depth understanding of a number of learners. CT perspectives can be measured by well-designed survey instruments, and qualitative methods such as interviews can be conducted if in-depth understanding is needed.

The future work of evaluating learners' CT development is to design instruments for measuring primary school learners in these three dimensions. Regarding CT concepts and practices, researchers should design programming project rubrics and test instruments to capture learners' learning outcomes in these areas. Regarding CT perspectives, the constructs of computational identity, programming empowerment and perspectives of expressing, connecting and questioning should be further explored and established. Researchers are recommended to develop and validate instruments for measuring learners' computational identity and programming empowerment, and how they express, connect and question in the digitalised world. The proposed evaluation components and methods will be implemented in senior primary schools when these instruments are developed. With appropriate evaluation components and methods, it is believed that schools will be in a better position to motivate and nurture young learners to become creative problem formulators and problem-solvers in the digitalised world.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Araujo, A. L., Andrade, W. L., & Guerrero, D. D. (2016, October 12–15). A systematic mapping study on assessing computational thinking abilities. In *Proceedings of the IEEE Frontiers in Education Conference* (pp. 1–9). Erie, PA.

Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory*. Englewood, Cliffs, NJ: Prentice-Hall.

www.dbooks.org

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.

Barr, V., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *ISTE Learning & Leading, 38*(6), 20–22.

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved November 21, 2016, from http://pact.sri.com/resources.html.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In A. F. Ball & C. A. Tyson (Eds.), *Proceedings of the 2012 Annual Meeting of the American Educational Research Association* (25 pp.). Vancouver, Canada: American Educational Research Association.

Brugman, G. M. (1991). Problem finding: Discovering and formulating problems. *European Journal of High Ability, 2*(2), 212–227.

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.

Center for Computational Thinking, Carnegie Mellon. (2010). Retrieved November 15, 2017, from https://www.cs.cmu.edu/~CompThink/.

Chiu, M. M., & Zeng, X. (2008). Family and motivation effects on mathematics achievement: Analyses of students in 41 countries. *Learning and Instruction, 18*(4), 321–336.

Computer Science Teachers Association. (2011). *K-12 computer science standards*. Retrieved November 21, 2016, from http://csta.acm.org/Curriculum/sub/K12Standards.html.

Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress*. Retrieved April 16, 2017, from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Deci, E. L., & Ryan, R. M. (1985). *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum.

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). London, England.

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). Raleigh, NC.

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.

Gee, J. P. (2000). Identity as an analytic lens for research in education. *Review of Research in Education, 25,* 99–125.

Giordano, D., & Maiorana, F. (2014, April 3–5). Use of cutting edge educational tools for an initial programming course. In *Proceedings of IEEE Global Engineering Education Conference* (pp. 556-563). Istanbul, Turkey.

Gouws, L., Bradshaw, K., & Wentworth, P. (2013, October 7–9). First year student performance in a test for computational thinking. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference* (pp. 271–277). East London, South Africa.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Grover, S., Cooper, S., & Pea, R. (2014, June 21–25). Assessing computational learning in K-12. In *Proceedings of the Conference on Innovation & Technology in Computer Science Education* (pp. 57–62). Uppsala, Sweden.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school learners. *Computer Science Education, 25*(2), 199–237.

Huitt, W. (2007). *Maslow's hierarchy of needs. Educational psychology interactive*. Valdosta, GA: Valdosta State University. Retrieved November 8, 2017, from http://www.edpsycinteractive.org/topics/regsys/maslow.html.

Kong, S. C. (2017). Development and validation of a programming self-efficacy scale for senior primary school learners. In S. C. Kong, J. Sheldon & K. Y. Li (Eds.), *Proceedings of the International Conference on Computational Thinking Education 2017* (pp. 97–102). Hong Kong: The Education University of Hong Kong.

Kukul, V., & Gökçearslan, Ş. (2017). Computer programming self-efficacy scale (cpses) for secondary school students: Development, validation and reliability. *Eğitim Teknolojisi Kuram ve Uygulama, 7*(1), 158–158.

Lawler, R. W. (1997). *Learning with computers*. Exeter: Intellect Books.

Lye, S. Y., & Koh, J. H. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41,* 51–61.

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using pair programming: Who benefits? *All Ireland Journal of Teaching and Learning in Higher Education, 6*(2), 1411–1425.

Makinen, M. (2006). Digital empowerment as a process for enhancing citizens' participation. *E-learning, 3*(3), 381–395.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008, March 12–15). Programming by choice: Urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 367–371). Portland, OR.

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.

Maslow, A. H. (1943). A theory of human motivation. *Psychological Review, 50*(4), 370.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239–264.

Mueller, J., Beckett, D., Hennessey, E., & Shodiev, H. (2017). Assessing computational thinking across the curriculum. In P. J. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 251–267). Cham, Switzerland: Springer International Publishing.

National Research Council. (2011). *Report of a workshop of pedagogical aspects of computational thinking*. Retrieved November 21, 2016, from https://www.nap.edu/catalog/13170/report-of-a-workshop-on-the-pedagogical-aspects-of-computational-thinking.

National Research Council. (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century. Committee on defining deeper learning and 21st century skills*. Retrieved December 6, 2016, from http://www.p21.org/storage/documents/Presentations/NRC_Report_Executive_Summary.pdf.

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

Peterson, I. (2002). *Mathematical treks: From surreal numbers to magic circles*. Washington, DC: Mathematical Association of America.

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human Development, 15*(1), 1–12.

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017, March 8–11). Assessing computational thinking in CS unplugged activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501–506). Seattle, Washington.

Román-González, M., Pérez-González, J., & Jiménez-Fernández, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test.

www.dbooks.org

Computers in Human Behavior. Retrieved 22 March, 2017, from http://www.sciencedirect.com/science/article/pii/S0747563216306185.

Ruf, A., Mühling, A., & Hubwieser, P. (2014, November 5–7). Scratch vs. Karel: Impact on learning outcomes and motivation. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 50–59). Berlin, Germany.

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). San Diego, CA.

Sherman, M., & Martin, F. (2015). The assessment of mobile computational thinking. *Journal of Computing Sciences in Colleges, 30*(6), 53–59.

Voland, G. (1999). *Engineering by design*. Reading, MA: Addison-Wesley.

Waller, D. (2016). *Gcse computer science for OCR student book*. Cambridge, England: Cambridge University Press.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge, England: Cambridge University Press.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February 29–March 3). 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). Raleigh, NC.

Wigfield, A., & Eccles, J. S. (2000). Expectancy-value theory of achievement motivation. *Contemporary Educational Psychology, 25,* 68–81.

Wilson, A., Hainey, T., & Connolly, T. M. (2012, October). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *Proceedings of the 6th European Conference on Games-based Learning* (10 pp.). Cork, Ireland.

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

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational thinking and expository writing in the middle school. *ACM Transactions on Computing Education, 11*(2), 1–22.

World Economic Forum. (2016). *The Fourth Industrial Revolution: What it means, how to respond*. Retrieved April 15, 2017, from https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research, 53,* 562–590.

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). Canterbury, England.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part III
Computational Thinking and Programming Education in K-12

www.dbooks.org

Chapter 9
Learning Composite and Prime Numbers Through Developing an App: An Example of Computational Thinking Development Through Primary Mathematics Learning



Siu-Cheung Kong

Abstract Computational Thinking (CT) is a universal competence that young learners should acquire. This study illustrates the development of CT through app development for the learning of composite and prime numbers in primary school mathematics. Learners begin with inquiry activities regarding the meaning of composite numbers and prime numbers, and they are guided to find the factors of a given number by exploring an algorithm that involves the concept of modulo and can be implemented in a programming environment. Learners use a simple app to learn how the operator MOD works and then work in pairs to design an algorithm and devise a programming solution to find the factors of a given number by reusing the code of the simple app. The app will tell the users whether an inputted number is a composite or prime. Learners will make an initial connection with cryptography when large numbers such as 563,879 are explored, and they will obtain a more in-depth understanding of composite and prime numbers when 1 and 0 are tested in the app. In the process of building the app, learners will learn CT concepts such as sequence, loops, conditionals, events, operators and procedures. Learners will experience CT practices like reusing and remixing, being iterative and incremental, testing and debugging, abstracting and modularising and algorithmic thinking in developing the app. Learners will develop CT perspectives in the process of expressing, questioning and connecting with the digital world in developing the app.

S.-C. Kong (✉)
Centre for Learning, Teaching and Technology, The Education University of Hong Kong, 10 Lo Ping Road, Tai Po, Hong Kong
e-mail: sckong@eduhk.hk

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_9

Keywords App development · Composite numbers · Computational thinking · Primary school mathematics · Prime numbers

9.1 Introduction

Researchers and practitioners in education advocate the introduction of Computational Thinking (CT) in P-12 education to nurture learners' problem-solving and creativity (Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). The goal of CT development is to enable learners to draw on the fundamental concepts of computer science to formulate problems and find computational solutions to real-life problems (Wing, 2011). The usefulness of app development is not limited to the field of computer science, and it can also help students to learn in a wide range of subjects. Computational tools and skill sets are helpful in mathematics and science learning (Eisenberg, 2002; Repenning, Webb, & Ioannidou, 2010; Wilensky 1995; Wilensky, Brady, & Horn, 2014), while mathematics and science offer a meaningful context in which problems can be formulated and CT can be developed (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009; Jona et al., 2014; Wilensky et al., 2014). This reciprocity is the motivation to combine CT and mathematics and science learning. Many P-12 learners have started to learn mathematical concepts through app development (Kahn, Sendova, Sacristán, & Noss, 2011; Wolber, Abelson, & Friedman, 2014; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). To teach novice programmers, block-based programming environments, such as Scratch and App Inventor, are considered appropriate (Meerbaum-Salant, Armoni, & Ben-Ari, 2010; Price & Barnes, 2015). This study, therefore, demonstrates how mathematical concepts can be developed through the creation of an app in App Inventor using an example of learning composite and prime numbers in primary school mathematics. The experience of creating an app helps learners turn abstract pattern recognition into an algorithmic and concrete operational experience.

9.2 Background

Wing's (2006) article popularised the term 'computational thinking' and defined it 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' (p. 1). Computational thinking relates to design, problem-solving and the understanding of human behaviour and draws on the basic concepts of computer science. Although it originates from computer science, CT is not exclusive to that field, and it goes well beyond computing. It is 'reasoning about the problems in computational terms' (Syslo & Kwiatkowska, 2014, p. 2), which means that learners are able to formulate problems in ways that help to develop computer solutions.

www.dbooks.org

9.2.1 CT Framework

Brennan and Resnick developed a CT framework (2012) that was structured in three dimensions: CT concepts that learners engage with in programming, CT practices that learners develop when solving computational problems and CT perspectives that learners gain when interacting with technologies and relate to their experience with the digital world. Regarding CT concepts, learners are expected to learn the basic ingredients of programming such as sequences, conditionals, loops, events, operators and data handling. When tackling computational problems, learners experience CT practices such as reusing and remixing, being incremental and iterative, abstracting and modularising, testing and debugging, and algorithmic thinking. In the programming process, learners have opportunities to develop CT perspectives such as expressing, questioning and connecting with the digital world. Expressing refers to learners' opportunities to use programming as a medium for self-expression. Questioning is the ability to ask questions about and with technology. Connecting is the process by which learners create with and for others in programming. It helps learners to experience working with other people and can lead to more accomplishments resulting from discussions and collaborations. Creating for others enables learners to experience positive and negative feedbacks in authentic contexts when their creations are appreciated (or not appreciated) by other people. Connecting is the most important component of CT perspectives, as it enables learners to connect CT practices with the digital world so that they feel empowered and have an identity in the digital world. All three dimensions of CT are essential for implementing CT in school education.

9.2.2 Block-Based Programming Environments

Inspired by LogoBlocks (Begel, 1996), block-based environments allow learners to create apps by dragging and dropping blocks into a scripting pane with minimal effort spent dealing with programming syntax. It minimises syntax errors because the blocks can only fit together when the code makes sense. In addition, feedback is immediate, as the results of the code blocks that the learners build are shown right in front of them when the program is executed. Such a user-friendly programming environment enables novice programmers to be more engaged in thinking about the programming process in a more systematic manner as they build the code block by block. Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010) and App Inventor (Wolber, Abelson, Spertus, & Looney, 2011) are the most common block-based programming environments used by novice programmers (Price & Barnes, 2015). This study illustrates how App Inventor was used to design an app for learning primary mathematics and CT development.

9.2.3 Mathematics Learning and CT Development

Learners can develop CT in learning mathematics when programming is introduced as pedagogy. Learners can develop CT in learning a variety of subjects, including mathematics, science, social studies and language studies (Barr & Stephenson, 2011). CT and mathematics are particularly closely related. This study connects CT and mathematics learning explicitly to show that learners can develop both CT and mathematical concepts holistically. The National Research Council (NRC) suggested that mathematics and CT should be essential practices for the scientific and engineering aspect of the Framework for K-12 Science Education (NRC, 2012). Programming is pedagogically helpful in mathematics learning (Eisenberg, 2002; Repenning, Webb, & Ioannidou, 2010; Wilensky, 1995; Wilensky, Brady, & Horn, 2014). Syslo and Kwiatkowska (2014) advocated that traditional topics in school mathematics should be extended and enriched through the application of programming to build solutions. The literature review conducted by Lye and Koh (2014) revealed that learners learn mathematics well with the help of programming (e.g. Kahn, Sendova, Sacristán, & Noss, 2011). The study conducted by Yadav et al. (2014) found that teachers favoured the implementation of CT into as many subjects as possible. Mathematics is most preferred as CT always involves problem-solving with mathematics, such as design algorithms and heuristics in the solving process. Research studies have documented that the immersion of CT in mathematics learning can be successfully achieved by building apps. In the study conducted by Morelli et al. (2010), participating teachers and learners developed a learning app to promote the immersion of CT in traditional subjects, such as mathematics. In Ke's study (2014), a group of secondary school learners used Scratch to design mathematics games that successfully enhanced their mathematics learning outcomes. The study by Foerster (2016) documented similar outcomes in a high school that integrated programming into its mathematics curriculum by combining Scratch and geometry learning in Grades 6 and 7. Long-term assessments indicated that these education initiatives resulted in significant improvements in their learners' performance. Hsu and Hu (2017) conducted a study in which Grade 6 learners were guided by teachers to apply CT to write a block program in Scratch to solve daily-life mathematic problems. It was proved successful in enhancing learners' mathematics learning effectiveness. Evidence indicates that CT can be integrated into meaningful mathematics learning through app development. This study demonstrates how programming can serve as pedagogy in the learning and teaching of primary school mathematics for better mathematics learning and CT development.

www.dbooks.org

9.2.4 *Learning Composite and Prime Numbers in Primary School Mathematics*

One of the traditional methods of learning composite and prime numbers is through the sieve of Eratosthenes. While it is generally taught at the tertiary level, its simplified version is introduced in primary school: 'sifting out the composite numbers, leaving only the primes' (Maletsky et al., 2004). Ploger and Hecht (2009) suggested using the software Chartworld to introduce the sieve of Eratosthenes to primary learners in a visualised and interactive way. Chartworld colours all multiples of a number when that number is selected. For instance, after selecting the number 3, all multiples of 3 except 3 will be highlighted. After selecting the numbers from 2 to 10, all of their multiples will be highlighted. The remaining un-highlighted numbers are prime numbers, except 1. Although this method enables young learners to learn about composite and prime numbers, it is limited in finding prime numbers in a given range. In this regard, there is a need to explore an alternative way to teach primary school learners so that they can explore larger composite and prime numbers. This study proposes an innovative pedagogy to guide primary school learners to make an app that can find the factors of an inputted number.

9.3 Developing an App as Pedagogy for Supporting the Conceptual Understanding of Composite and Prime Numbers

In traditional mathematics lessons, senior primary school learners are taught that prime numbers are only be divisible by 1 and themselves, while composite numbers have factors other than 1 and themselves. Yet mere rote memorisation of these definitions is not encouraged, as learners will not truly understand the concepts of composite and prime numbers. This study introduces an inquiry pedagogical approach to guide learners to develop these mathematics concepts together with the development of an app. Learners will be initially introduced to examples of and ideas underlying composite and prime numbers. Learners will then develop an app that finds the factors of an inputted number and tests whether they can distinguish between composite and prime numbers. The programming process helps learners to think more deeply about the meaning of composite and prime numbers.

9.3.1 *Inquiry Activities About Composite and Prime Numbers*

This study introduces the concepts of composite and prime numbers using an inquiry approach, which is a learning process whereby 'students are involved in their learning, formulate questions, investigate widely and then build new understandings, meanings

Example: 8 = 1×8 = 2×4	4×2 is not needed because it is the same as 2×4. 8×1 is not needed because it is the same as 1×8.
The factors of 8 are 1, 2, 4 and 8.	

Fig. 9.1 Find all factors of 8 by listing out all factorisations of 8

and knowledge' (Alberta Learning, 2004, p. 1). In this activity, learners will be asked what a composite number and a prime number are. Teachers will then explain the concepts with examples. Numbers with more than two factors are composite numbers, while numbers with only two factors—1 and themselves—are prime numbers. When finding the factors of a number, learners are reminded of the commutative properties of multiplication. For instance, 8 can be expressed as '$1 \times 8 = 8$', '$2 \times 4 = 8$', '$4 \times 2 = 8$' and '$8 \times 1 = 8$'. Because the products are the same irrespective of the order of the factors, learners know that '$4 \times 2 = 8$' and '$8 \times 1 = 8$' are equivalent to '$2 \times 4 = 8$' and '$1 \times 8 = 8$', respectively. When finding all of the factors of 8, learners only need to explore '$1 \times 8 = 8$', '$2 \times 4 = 8$' and do not need to consider '$4 \times 2 = 8$' and '$8 \times 1 = 8$'. The order of operation is not important in finding the factors of a number, and this is the commutative property of multiplication. This study encourages learners to generalise composite and prime numbers from examples. The concepts of composite and prime numbers can also be better understood by asking young learners to explore the factors of a number through visualisation activities, such as exploring with the online activity 'Find the factorisations of the number' in the Illumination Project (National Council of Teachers of Mathematics [NCTM], n.d.). By choosing a particular number, such as 8, learners are invited to represent all of the factorisations of 8 by selecting the appropriate rectangular planes.

Learners are guided to conclude that composite numbers can be represented by more than one rectangular plane, while prime numbers can only be represented by a single rectangular plane. This indicates that composite numbers have more than one pair of factorisations, while prime numbers have only a single pair of factors: 1 and themselves. Figure 9.1 shows a way to find all of the factors of 8 by listing all of its factorisations. Using this form of representation, young primary school learners can be easily guided to categorise numbers as composite or prime by counting the factors of a selected number. Figure 9.2 lists examples of numbers that have only two factors and numbers that have more than two factors in a range from 2 to 12.

9.3.2 Developing an App as Pedagogy

After introducing composite and prime numbers by counting the number of factors of a number, teachers can guide learners to develop an algorithm to generalise a method

www.dbooks.org

The numbers that only have two factors from 2 to 12	The numbers that have more than two factors from 2 to 12
2, 3, 5, 7, 11	4, 6, 8, 9, 10, 12

Fig. 9.2 Categorising composite and prime numbers by counting the number of factors of a number in the range from 2 to 12

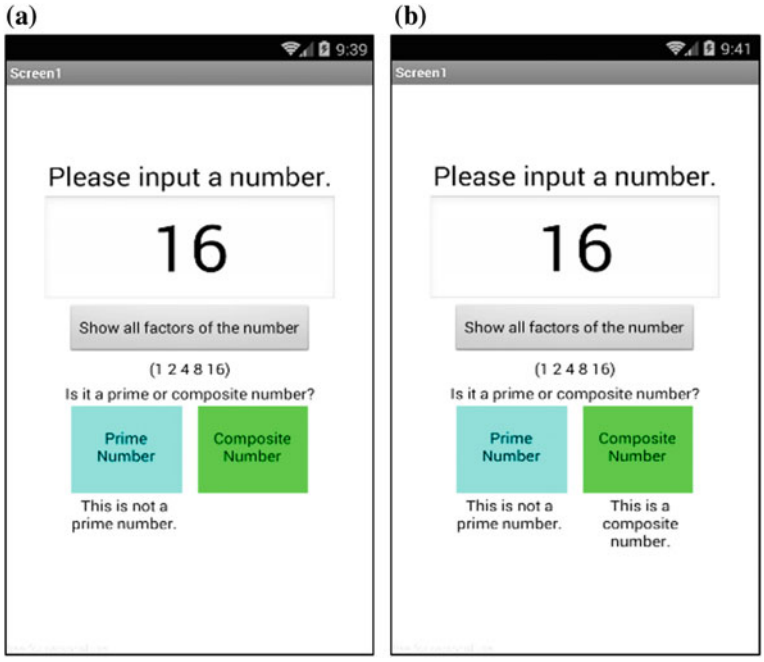


Fig. 9.3 The sample screens display the results of testing the inputted number 16 when **a** the 'Prime Number' button is pressed and **b** the 'Composite Number' button is pressed

to differentiate whether a number is composite or prime. Learners can input any integral numbers greater than or equal to one and ask the program to identify and list all factors of the given number. Learners are asked to decide whether it is composite or prime based on the number of factors found and listed by the app. They can check the answer by pressing the 'Composite Number' button or the 'Prime Number' button. They are asked to explore with the app by inputting different numbers before starting the programming task. Figure 9.3 shows sample screens displaying the results of testing an inputted number 16 when (a) the 'Prime Number' button is pressed and (b) the 'Composite Number' button is pressed.

9.3.3 Problem Decomposition and Algorithmic Thinking

There is a need to guide learners to figure out the sub-tasks in building the app. Decomposing the problems into sub-tasks is a high-level abstraction process. The four sub-tasks are finding the factors of the inputted number, showing the factors on the screen, deciding whether the number is a prime number and deciding whether the number is a composite number. Figure 9.4 shows the high-level abstraction involved in decomposing the problem into sub-tasks.

The most challenging task is to facilitate learners to find ways to check whether a number is a factor of an inputted number. The use of examples is the best way to guide learners. For example, learners can be asked to fill in the columns 'Modulo' and 'Factor of 8?' to check whether 1, 2, 3, 4, 5, 6, 7 and 8 are factors of 8 when it is divided by 1, 2, 3, 4, 5, 6, 7 and 8, respectively, as listed in Table 9.1. Table 9.1 shows all of the results in columns 'Modulo' and 'Factor of eight?' when eight is divided by 1, 2, 3, 4, 5, 6, 7 and 8, respectively. Learners are encouraged to discuss in pairs to identify the pattern of zero modulo in identifying the factors of the given number.

The operator of modulo in programming needs to be introduced to learners to prepare them to find an algorithm to check whether a number is a factor of a given

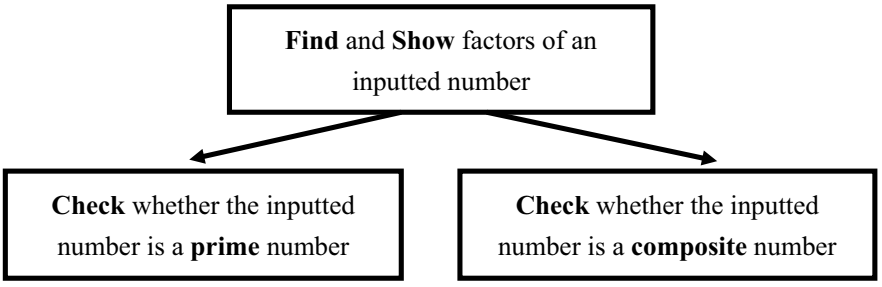


Fig. 9.4 A high-level abstraction of decomposing the problem into four sub-tasks

Table 9.1 A table with columns 'Modulo' and 'Factor of 8?' to be filled in by learners identifying patterns to find the factors of a given number

Division	Divisor	Modulo	Factor of 8?
$8 \div 1$	1	0	✓
$8 \div 2$	2	0	✓
$8 \div 3$	3	2	
$8 \div 4$	4	0	✓
$8 \div 5$	5	3	
$8 \div 6$	6	2	
$8 \div 7$	7	1	
$8 \div 8$	8	0	✓

www.dbooks.org

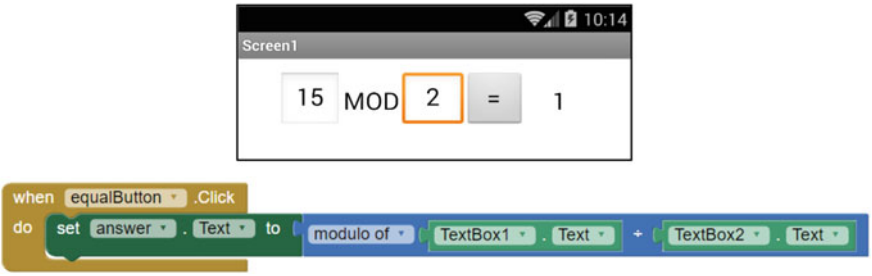


Fig. 9.5 The interface and code for learners to understand the operator of modulo in the programming environment

Input an InputNumber
Repeat number from 1 to InputNumber:
    If the modulo of InputNumber ÷ number = 0 then
    The number is the factor of InputNumber and put them in a Factorlist
End Repeat
Print Factorlist

Fig. 9.6 An algorithm for finding all of the factors of an inputted number

number. Figure 9.5 shows the interface and the code for learners to understand the operator of modulo in the programming environment.

Learners are then asked to work in pairs to design an algorithm to find all factors of an inputted number. Figure 9.6 shows an algorithm for finding all factors of an inputted number.

9.3.4 Reusing Code from a Simple App to Build an App to Find Factors

Following the algorithmic design, learners can start to build an app to find all of the factors of a given number by using the operator of modulo. Figure 9.7 shows the code of a program for finding all of the factors of a positive integral number inputted by a learner.

Figure 9.8 shows the code executed when the 'Prime Number' button and the 'Composite Number' button are clicked, respectively. The code in 'primeButton' and 'compButton' counts the number of factors of the input number to check whether it is a prime or composite number.

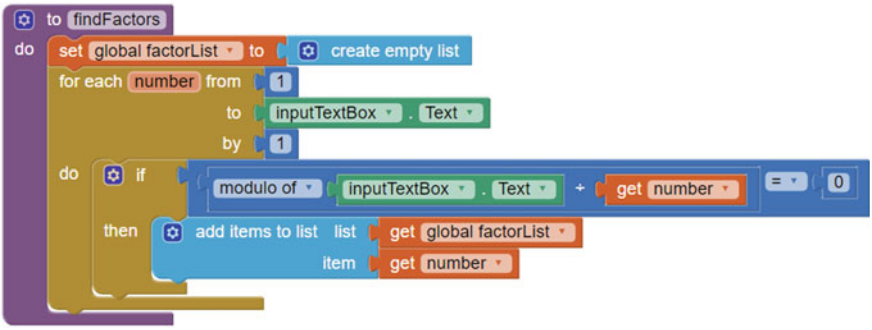


Fig. 9.7 The code of a program for finding all of the factors of a positive integral number

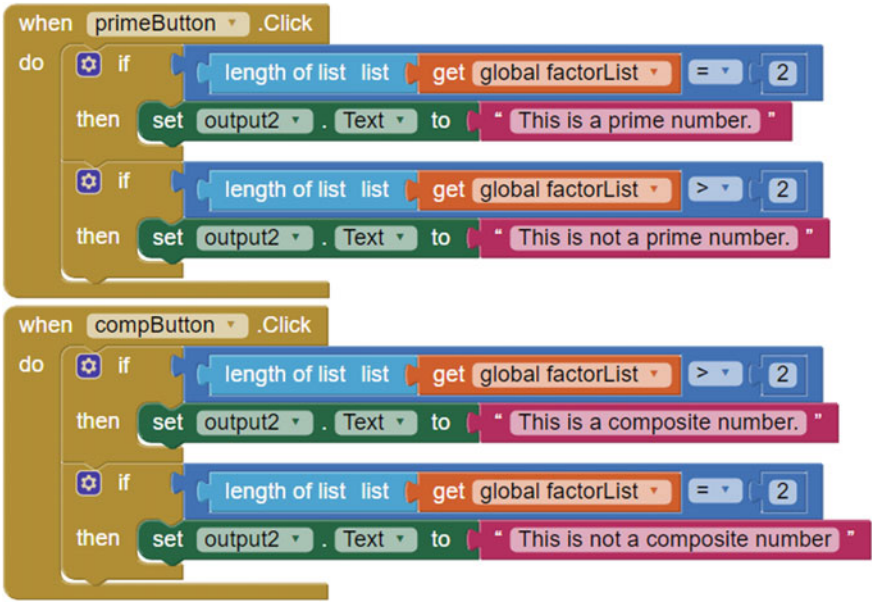


Fig. 9.8 The code executed when the 'Prime Number' button and the 'Composite Number' button are clicked, respectively

9.3.5 Testing the App and Connecting the Tasks with the Digital World

Learners can then test the program by checking whether the input number is a prime number or a composite number. The possible answers are 'This is a prime number' or 'This is not a prime number' when the 'Prime Number' button is clicked and 'This is a composite number' or 'This is not a composite number' when the 'Composite Number' button is pressed. Figure 9.9 shows the sample screens displaying the results

www.dbooks.org

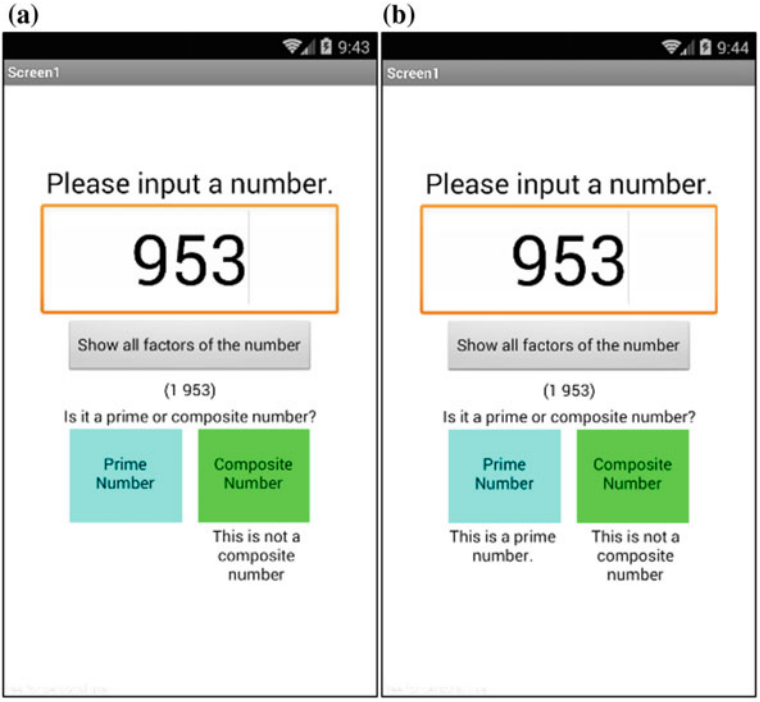


Fig. 9.9 Sample screens displaying the results of testing the inputted number 953 when **a** the 'Composite Number' button and **b** the 'Prime Number' button are pressed

of testing the inputted number 953 when (a) the 'Composite Number' button and (b) the 'Prime Number' button are pressed, respectively.

This app enables learners to test whether large positive numbers are prime or composite numbers, and they will thus come to understand the importance of the application of such testing in cryptography. This mathematical topic is selected as the theme of this computational task because of the significant role of prime numbers in cryptography. Email communication, instant messaging and satellite television, which are ubiquitous in everyday life, often require data encryption for the safe delivery of information. Encryption methodologies such as RSA use trapdoor one-way functions to encrypt messages. These functions are easy to compute in the forward direction, but difficult to compute in reverse without special information. RSA utilises some properties of prime number multiplication to design a function that satisfies the definition of the trapdoor one-way function (Salomaa, 1996). To enable learners to experience a one-way function, multiplication and factorisation are used as examples in this study to demonstrate the forward and reverse operations, respectively. The app developed in this study enables learners to test whether a large positive number is a composite or prime number. For example, the app can provide the factors of a relatively large number, such as 563,879, from the perspective of

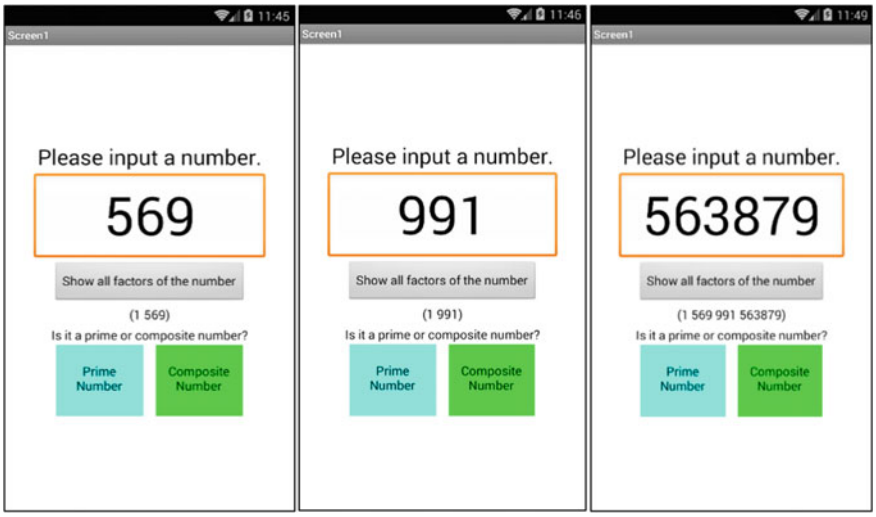


Fig. 9.10 The factors of a relatively large number are displayed after a time gap

young primary learners. As shown in Fig. 9.10, the factors of 563879 are 569 and 991, which are prime. Learners have to wait several minutes before they can see the outcomes, during which time they can discuss and connect the function with the work of encryption. The larger the number, the more time will be needed to identify its factors. It is important for us to find sufficiently large numbers for the encryption work so that our messages can be transmitted safely in all digital communications.

9.3.6 Using '1' and '0' to Trigger In-depth Discussion of Composite and Prime Numbers

Learners are then guided to find the number of factors of the number 1, which is one. The outcomes will be 'This is not a prime number' and 'This is not a composite number'. At this stage, teachers can emphasise the definition of a prime number and explain why 1 is not considered prime, which is rarely discussed in teaching. This app can thus help teachers to trigger discussion among learners in classrooms. Figure 9.11 shows the sample screens displaying the results of testing an inputted number 1 when (a) the 'Composite Number' button and (b) the 'Prime Number' button are pressed.

Learners are then asked to find number of factors of the number 0. The app will return an empty list of factors because it is not designed to handle this case. There are infinitely many factors for 0 when it is checked by the rule 'the inputted number can be divided by a positive integer and it is a factor when the modulo is zero'. When 0 is divided by 1, the modulo is 0. Similarly, when 0 is divided by 2 or any

www.dbooks.org

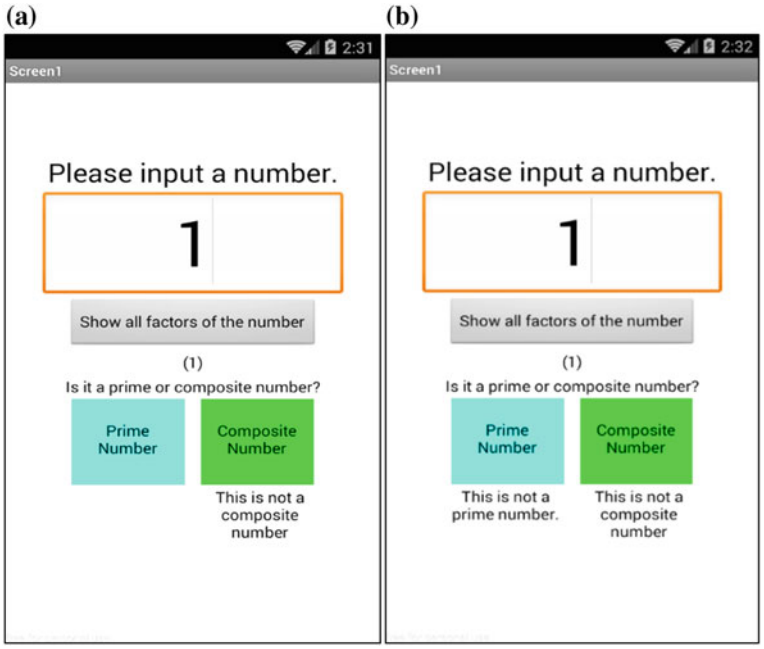


Fig. 9.11 The sample screens displaying the results of testing an inputted number 1 when **a** the 'Composite Number' button and **b** the 'Prime Number' button are pressed

other positive number, the modulo is 0. Therefore, 0 has infinitely many factors, and the app will report 'There are infinitely many factors' when 0 is inputted for testing. Thus, learners can understand the limitations of the computer program regarding this case, which is worthy of discussion among learning peers and teachers. Figure 9.12 shows the message after clicking the 'Show All Factors of the Number' button when 0 is inputted.

9.3.7 Adding a Conditional Statement to the App to Handle the Case of Inputting 0

Learners are then asked to write a conditional statement to respond to the input of 0. Figure 9.13 shows the conditional statement for handling the special case when 0 is inputted. This is an opportunity for learners to understand program debugging. Learners will know that programming an app is an iterative process and that the program needs to be completed incrementally. Learners can reuse the code for the 'findFactors' function shown in Fig. 9.13 to search for factors.

With the categorisation of composite and prime numbers taking place in the above manner, this study proposed four categories of numbers: composite numbers, prime

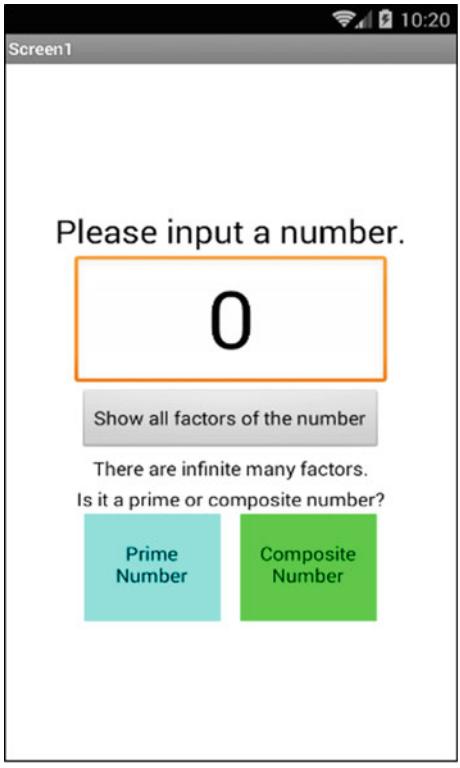


Fig. 9.12 The message 'There are infinitely many factors' after clicking the 'Show All Factors of the Number' button when 0 is inputted

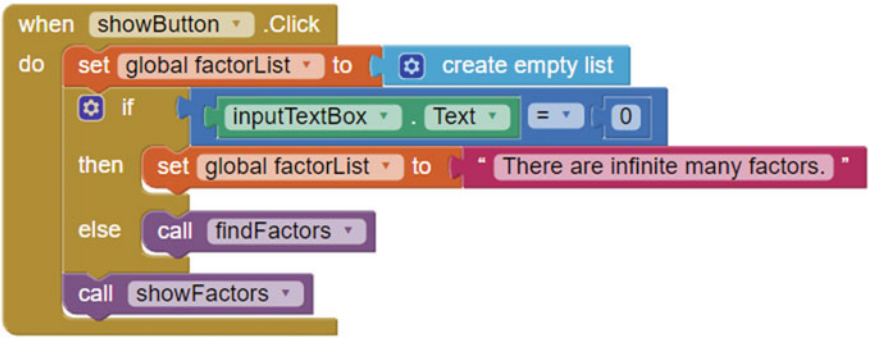


Fig. 9.13 The conditional statement for handling the case in which 0 is inputted in this app

www.dbooks.org

numbers, 1 and 0. In this regard, the definition of a composite number should be revised from the original definition, ‘a number with more than two factors’, to ‘a number with more than two factors but a finite number of factors’.

9.4 Computational Thinking Development

9.4.1 CT Concepts Development

In the process of building the app, learners learn CT concepts such as sequence, loops, conditionals, events, operators, data handling and procedures (see Fig. 9.14). The concept of sequences can be developed in building the ‘when showButton.Click’ program. Learners need to think carefully about the order of finding the factors after considering the special case of 0. When building the ‘when showButton.Click’ program, learners will first consider whether the input number is 0. If so, the app should tell users that it has an infinite number of factors instead of listing all of its factors. If the input number is not 0, the program should call the ‘findFactors’ procedure, as all numbers except 0 have a finite number of factors. Learners will thus understand the concept of sequence in greater depth in this case if they understand that the order of instructions for finding factors or assigning the message ‘There are infinitely many factors’ to the factorList should always come before showing the factors or the message. The concept of repetition is also developed when building the ‘findFactors’ codes. Using the ‘for each from to’ block, the program repeatedly runs the blocks in the do section, testing the numbers ranging from 1 to the input number in finding all of the factors. Learners can understand the concept of events when using the ‘when showButton.Click’ block. They will observe when an event occurs (i.e. clicking the ‘show’ button), and the app will trigger a series of tasks for execution until completion. They can also learn the concept of naming/variables when they create a factor list with the ‘initialise global name to’ block. The concept of conditionals is embedded in the use of modulo to find the factors of the input number. Using the ‘if-then’ block, the program will add the number to the factor list if the modulo is 0 after the input number is divided by the number. Learners can also develop the concept of operators when using the modulo to check the remainders of the divisions and using ‘greater than’ and ‘equals’ blocks in the ‘when primeButton.Click’ and ‘when compButton.Click’ programs. The concept of data is introduced in the use of lists. After the program finds the factors by the modulo, learners need to store all of the factors in the list before showing them. Learners can also understand the use of procedures in a complex computational task like this one. After the ‘findFactors’ program is built, the app will show the list of factors on the screen. Therefore, a procedure that shows factors should be created. To show the factor list, the ‘showButton’ is made. When the ‘showButton’ is clicked, the program calls the ‘findFactors’ and ‘showFactors’ procedures that were created previously. Learners thus know that using procedures can avoid bulky programs.

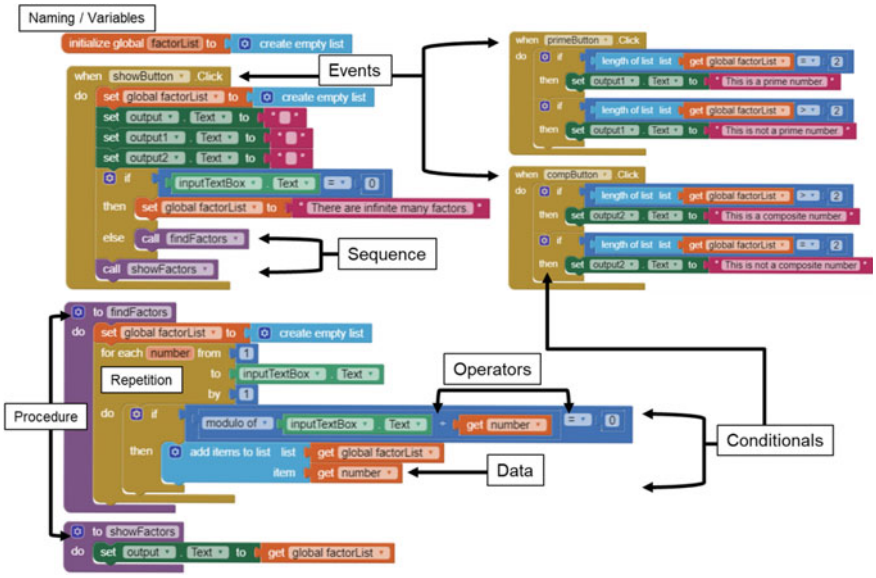


Fig. 9.14 The CT concept learned from the factor app

9.4.2 CT Practices Development

Learners can also experience CT practices, such as reusing and remixing, being iterative and incremental, testing and debugging, abstracting and modularising, and algorithmic thinking, in developing the app. They experience the practice of reusing and remixing when they use the code block of the modulo from the simple app (see Fig. 9.5) to create the 'findFactor' program (see Fig. 9.15). To reuse the code block of the modulo from the simple app in making the Factor app, learners can save the code in the backpack, and it can then be extracted when building the 'findFactors' program. The backpack is a copy-and-paste feature in App Inventor that allows users to copy blocks from a previous workspace and paste them into the current workspace.

Learners can also experience the practice of being incremental and iterative in the process of app development (see Fig. 9.16). Learners can be guided to initially build a program that finds and shows the factors of an input number and then put the program into testing and debugging. Learners can develop the program further to check whether the inputted number is a prime or a composite number. Thus, learners can experience the process of developing the app incrementally and iteratively.

Learners learn also about testing and debugging throughout the app development process in this example. One particular point concerns the testing of the case of inputting 1 and 0 in this example. When the user presses either of the buttons to test whether 1 is a prime or composite number, the app will give no response (see Fig. 9.17), as the original app only functions when the inputted number has two or more factors. When the inadequacy is identified, learners have to modify the program

www.dbooks.org

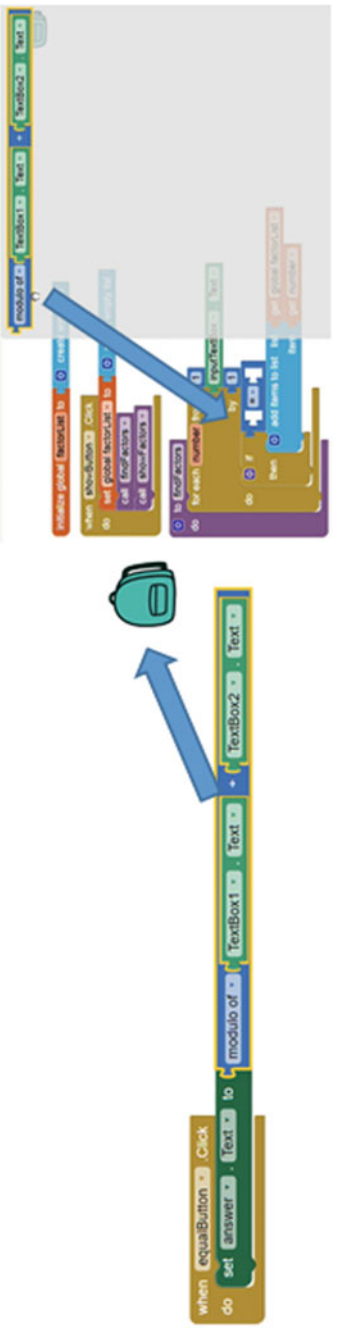


Fig. 9.15 Reuse and remix the modulo code from the simple app using the backpack feature in App Inventor

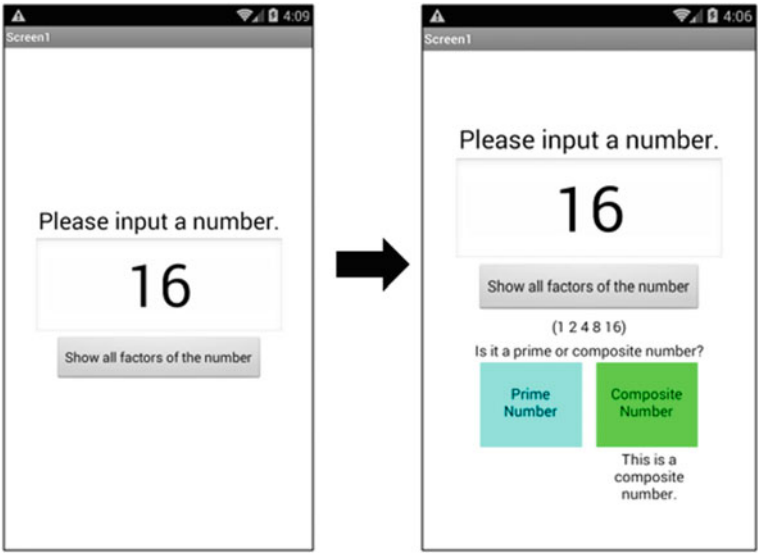


Fig. 9.16 Learners develop the app incrementally and iteratively

to handle the special case of inputting 1. Moreover, an error will be found if 0 is input. Because 0 has an infinite number of factors, the app cannot list them; thus, the app needs to be enhanced to inform the user that 0 has an unlimited number of factors, as shown in Fig. 9.17. From these examples, learners will understand the importance of testing and debugging in app development.

It is not realistic to expect primary school learners are able to conduct high-level abstraction in programming design; however, this app development exposes learners to the high-level abstraction process. The four main sub-tasks highlighted at the beginning of programming design (i.e. finding factors and showing factors of the input number, checking whether the input number is a prime number and checking whether the input number is a composite number) can be shown to learners if they are unable to generate the idea after discussion. Teachers should not be frustrated if learners are unable to decompose problems into sub-tasks at this stage, but they should expose them to this process so that they appreciate that it is an important part of the programming process. Before building the code, learners should be motivated to conduct an algorithmic design that delineates the detailed instructions required to solve the problem. This process facilitates the development of algorithmic thinking, which is an essential cognitive skill in CT development (Angeli et al., 2016; Barr & Stephenson, 2011; Wing, 2006). Again, we should not expect learners to be able to work this out on their own; however, they should be guided and exposed to this process so that the seeds are sown for later development and growth. Individuals with algorithmic thinking skills can help them to precisely state a problem, divide the problem into well-defined sub-tasks and work out a step-by-step solution to tackle

www.dbooks.org

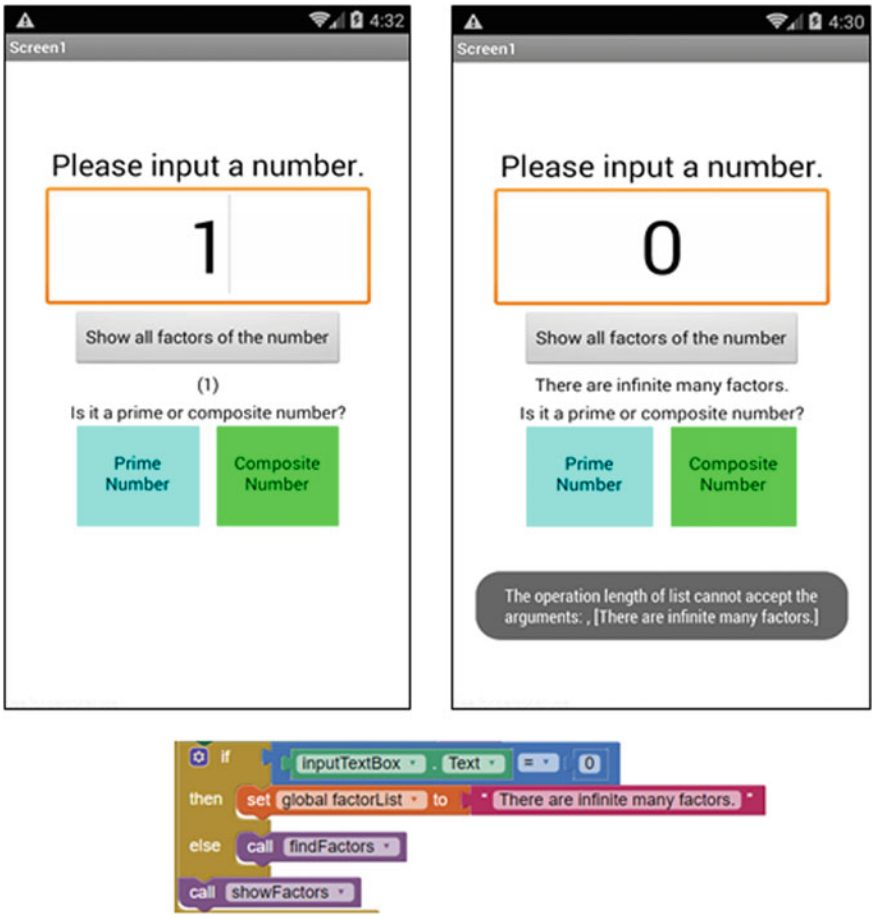


Fig. 9.17 Learners experience testing and debugging after 1 and 0 are inputted

each sub-task (Cooper, Dann, & Pausch, 2000). Figure 9.6 shows the algorithm for finding the factors with modulo.

As mentioned in Chap. 8, problem formulation is an essential component of CT practices. With the experience gained from developing the factor app, learners should be encouraged to formulate problems using their pattern recognition experience. Learners can be guided to reflect again on the pattern-identifying factors of a number by considering the outcome of the modulo when the number is divided by a divisor (Table 9.1). Teachers can ask learners to look for other patterns for formulating problems, for instance, learners may be able to discover patterns such as those shown in Table 9.2, which presents a pattern of identifying odd numbers using a similar strategy to that in Table 9.1.

Table 9.2 A table with the columns 'Modulo' and 'Odd number?' to be filled in by learners to identify patterns in finding an odd or even number

Division	Divisor	Modulo	Odd number?
1 ÷ 2	2	1	✓
2 ÷ 2	2	0	
3 ÷ 2	2	1	✓
4 ÷ 2	2	0	
5÷ 2	2	1	✓

9.4.3 CT Perspectives Development

In the process of developing the app, learners have opportunities to develop their CT perspectives, expressing themselves and connecting with and questioning the digital world. An important issue in the digital world is the security of data transmission. To encode and decode messages in real-life protocols, two sufficiently large prime numbers are used to compose a composite number. This app enables learners to obtain an initial idea of the speed of computing when they use the app to test whether a given relatively large number is a prime number or not. Through the app development, learners cannot only express what they understand about prime and composite numbers in the coding process, but also connect these mathematical concepts with encryption technology in the digital world and to think of different ways to encrypt messages. This encourages them to learn more mathematical concepts for encryption. Learners will be better able to raise questions about the digital world and privacy issues. Are the messages that they send in the digital world safe? Are there any other methods for data encryption? Should they learn more about programming in the digital world? They will also gain a sense of programming empowerment and will feel autonomous and competent in contributing to society by developing apps that can help them learn. We hope that in the long term, with the accumulation of more programming experience, learners will develop their computational identity and regard themselves as members of the computational community.

9.5 Conclusion

This study demonstrates how primary school learners develop an in-depth understanding of composite and prime numbers by building an app using App Inventor that uses the number of factors to formulate the concepts of composite and prime numbers. Learners can develop their CT concepts and experience CT practices in a series of programming tasks, including high-level planning to find and show factors and ground-level implementation of the modularised blocks to find and show the factors and to check whether the input number is a prime or composite number. They can also experience logical thinking and gain an in-depth understanding of composite and prime numbers in the process of building and testing the app. This example shows that building an app cannot only deepen their mathematics understanding but

www.dbooks.org

also develop their CT in terms of concepts, practices and perspectives. Future work is to evaluate the effectiveness of this pedagogy in classroom teaching. CT should not be limited to programming lessons, but should be developed in young learners in other subjects. It is also necessary to extend this pedagogy to other subjects such as science, language learning and visual arts.

References

Alberta Learning. (2004). *Focus on inquiry: A teacher's guide to implementing inquiry-based learning*. Edmonton, AB: Alberta Learning, Learning and Teaching Resources Branch.

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, MA: Massachusetts Institute of Technology.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the computational thinking. In *Annual American educational research association meeting.* Vancouver, BC, Canada.

Cooper, S., Dann, W., & Pausch, R. (2000). Developing algorithmic thinking with Alice. In *Proceedings of the information systems educators conference* (pp. 506–539). Philadelphia, PA: AITP.

Eisenberg, M. (2002). Output devices, computation, and the future of mathematical crafts. *International Journal of Computers for Mathematical Learning, 7*(1), 1–44.

Foerster, K. (2016). Integrating programming into the mathematics curriculum: Combining scratch and geometry in grades 6 and 7. In *Proceedings of the 17th annual conference on information technology education* (pp. 91–96). 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*(1), 38–43.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin, 41*(1), 183–187.

Hsu, T., & Hu, H. (2017). Application of the four phases of computational thinking and integration of blocky programming in a sixth-grade mathematics course. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), *Proceedings of international conference on computational thinking education 2017* (pp. 73–76). Hong Kong: The Education University of Hong Kong.

Jona, K., Wilensky, U., Trouille, L., Horn, M. S., Orton, K., Weintrop, D., & Beheshti, E. (2014, January). *Embedding computational thinking in science, technology, engineering, and math (CT-STEM)*. Paper presented at the future directions in computer science education summit meeting, Orlando, FL.

Kahn, K., Sendova, E., Sacristán, A. I., & Noss, R. (2011). Young students exploring cardinality by constructing infinite processes. *Technology, Knowledge and Learning, 16*(1), 3–34.

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*(1), 26–39.

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*(1), 51–61.

Maletsky, E., Roby, T., Andrews, A., Bennett, J., Burton, G., Luckie, L., McLeod, J., Newman, V., …, Scheer, J. (2004). *Harcourt math* (Grade 4, Florida Edition). Orlando, FL: Harcourt.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The programming language and environment. *ACM Transactions on Computing Education, 10*(4), 16.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. M. (2010). Learning computer science concepts with Scratch. In *Proceedings of the sixth international workshop on computing education research* (pp. 69–76). New York, NY: ACM.

Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2010). Can android app inventor bring computational thinking to K-12. *Proceeding of 42nd ACM technical symposium on computer science education* (pp. 1–6). New York, NY: ACM.

National Council of Teachers of Mathematics. (n.d.). *Illuminations: Factorize*. https://www.nctm.org/Classroom-Resources/Illuminations/Interactives/Factorize.

National Research Council. (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. Washington, DC: The National Academies Press.

Ploger, D., & Hecht, S. (2009). Enhancing children's conceptual understanding of mathematics through Chartworld Software. *Journal of Research in Childhood Education, 23*(3), 267–277.

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.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on computer science education* (pp. 265–269). New York, NY: Association for Computing Machinery.

Salomaa, A. (1996). *Public-key cryptography*. Berlin, Germany: Springer.

Syslo, M. M., & Kwiatkowska, A. B. (2014). Learning mathematics supported by computational thinking. Constructionism and creativity. In G. Futschek, & C. Kynigos (Eds.), *Constructionism and creativity: Proceedings of the 3rd international constructionism conference 2014* (pp. 367–377). Vienna, Austria: Austrian Computer Society.

Wilensky, U. (1995). Paradox, programming, and learning probability: A case study in a connected mathematics framework. *Journal of Mathematical Behavior, 14*(2), 253–280.

Wilensky, U., Brady, C., & Horn, M. (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.

Wing, J. M. (2011). Research notebook: Computational thinking—What and why? *The link magazine*, Carnegie Mellon University, Pittsburgh. http://link.cs.cmu.edu/article.php?a=600.

Wolber, D., Abelson, H., & Friedman, M. (2014). Democratizing computing with App Inventor. *GetMobile: Mobile Computing and Communications*, *18*(4), 53–58.

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor: Create your own android apps*. Sebastopol, CA: O'Reilly Media.

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 (TOCE), 14*(1), 1–16.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 10
Teaching Computational Thinking Using Mathematics Gamification in Computer Science Game Tournaments



Chee Wei Tan, Pei-Duo Yu and Ling Lin

Abstract One of the key foundations in computer science is abstract algebra. Ideas of abstract algebra can be introduced to students at middle or pre-college schools to cultivate their capacity for logical thinking and problem-solving skills as well as gaining mathematical competency required in computer science. In this book chapter, we introduce ideas of mathematics gamification and a mobile app game, Algebra Game, richly rooted in abstract algebra and first proposed by the mathematician Tao (Gamifying algebra, 2012a, Software mock-up of algebra game 2012b). The Algebra Game teaches elementary algebra seemingly on the surface, and yet the game-play design possesses interesting abstract algebra ideas and mathematics gamification potential. We define mathematics gamification as the process of embedding mathematical concepts and their logical manipulations in a puzzle game-like setting aided by computers. We describe several mathematics gamification instances to enrich the Algebra Game play. Finally, we evaluate the learning efficacy of the Algebra Game mobile app software in computer science game tournaments modeled after eSports-like computer games in order to scale up the number of students who can learn the Algebra Game mathematics.

Keywords Mathematics education · Mathematics gamification · Mobile app games · Pedagogy · Personalized learning · K-12 mathematics · Computer science

C. W. Tan (✉)
The Institute for Pure and Applied Mathematics, Los Angeles, USA
e-mail: algebrachallenge@gmail.com

C. W. Tan · P.-D. Yu · L. Lin
Nautilus Software Technologies Limited, Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong
e-mail: pdyu@princeton.edu

L. Lin
e-mail: hkalexling@gmail.com

P.-D. Yu
Department of Electrical Engineering, Princeton University, Princeton, USA

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_10

10.1 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.*" (Minsky 1970). 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 described a preliminary pedagogical study on learning K-12 mathematics through mathematics gamification ideas and tested it 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. There are several ways to implement ideas of mathematics gamification in software that can be delivered to the students. We have chosen to deliver them by mobile app software that presents potentially strong opportunities for students to learn advanced mathematics in a systematic manner and at their own pace.

Any form of personalized learning technologies should provide a way to teach for mastery, where students are allowed to progressively move on to other challenging topics only after they have fully grasped the topic at hand. This approach requires a personalized approach that caters for individual student learning pace that can also complement with traditional classroom teaching. Our mathematics gamification technology leverages the ubiquitous availability of personal computing devices such as mobile devices like smartphones and tablets, offering students the opportunity to personalize their mathematics learning experience, allowing students to learn new knowledge or perform self-assessments at their own pace, and therefore maximizing the efficiency of learning in order to "teach for mastery". Another unique advantage of our mathematics gamification technology is its ability to offer students instant feedback as they play, which is a crucial part of "debugging" their thinking process. In traditional classroom learning, students' answers often are graded and then returned weeks later. On the contrary, personalized learning technologies such as our mathematics games (namely, Algebra Game and Algebra Maze, which we will present in details in the following) enable students to get instant visual feedback as they play, so that they can reconsider the situation and correct their moves, which is an example of "thinking process debugging" on the go and in real time.

www.dbooks.org

10.2 Algebra Gamification

Elementary algebra—the first cornerstone of K-12 mathematics—has been highlighted by the National Academy of Engineering (Lavernia & VanderGheynst, 2013) 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 (Devlin, 2011; Pope & Mangram, 2015; Shapiro, 2013; Mackay, 2013; Novotney, 2015).

Algebra gamification is a pedagogical approach to learning elementary algebra. This approach is particularly useful when used at an early stage of K-12 education to give students a heads up on 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 the 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* (Wing, 2006). 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 (Tao, 2012a, b) 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, Tao (2012a, b) developed a software mock-up of the game as shown in Fig. 10.1.

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 left-hand and right-hand sides, respectively, through a selection of a finite number of given clues. In the following,

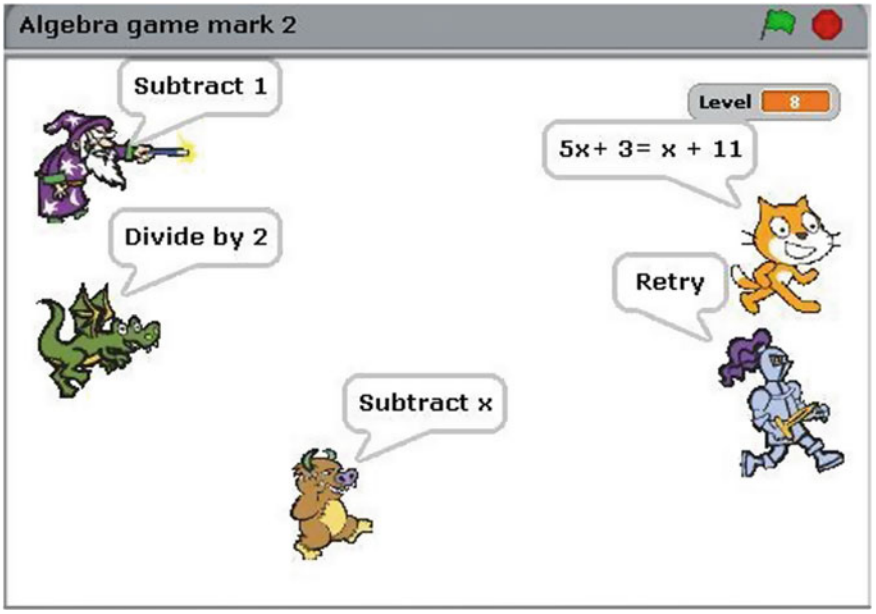


Fig. 10.1 Terence Tao's software mock-up of his algebra game in 2012

we give an example using a screenshot of the game as shown in Fig. 10.1. 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 Fig. 10.1 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. With regard to the gameplay design, 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

www.dbooks.org

author ("The Algebra Game Project", n.d.), and detailed answers to these questions along with the software development will be published in other venues.

10.3 Mathematics Gamification of Algebra Maze

In this and the next section, we describe the mathematics gamification building on Tao (2012a, b) algebra game and the software implementation of our mobile apps called the Algebra Game and the Algebra Maze, and the mobile app software are freely available to download at the Apple iOS Store or Google Play Store ("The Algebra Game Project", n.d.).

In Algebra Maze, we combine maze-solving elements and single-variable algebra equation solving together as shown in Fig. 10.2, which is the game-play screenshot 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 one unit upward, and the button "**+2**" corresponds to the avatar moving rightward 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 an algebraic meaning, i.e., each position in the maze represents a different equation having different coefficients or constant values.

In the initial levels of Algebra Maze, the treasure is made visible, and then at subsequent levels, the treasure is rendered invisible, i.e., hidden from the player as shown in the right-hand side of Fig. 10.2. Hence, the player needs to make use of the "information" in the given equation to deduce the location of the treasure. In

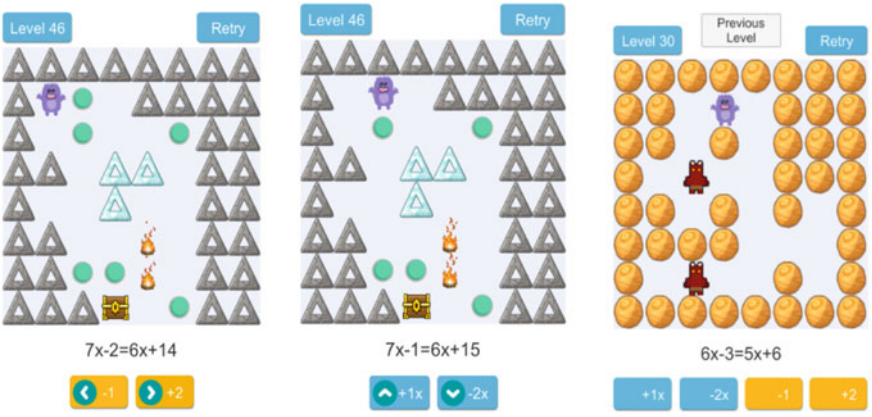


Fig. 10.2 Algebra maze mobile app game with maze-like gamification design and freely available for download at iTunes app store and Google play store

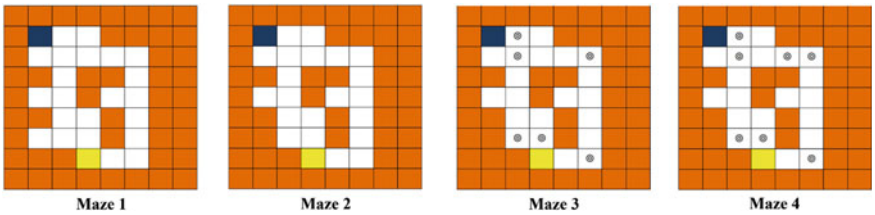


Fig. 10.3 Four different game designs of algebra maze

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 = numerical_solution", i.e., the puzzle is solved.

In order to make Algebra Game more challenging, such that players can learn thoroughly, we add more new features in subsequent higher levels. One of the features is "traps", once the player steps on the trap features, the buttons for movement control will toggle, either from left–right to up–down or from up–down to left–right. For example, the player can only perform "+s" or "−t", to the given equation initially, and after the player has stepped on the "traps", the operation will change from or "−t" to "+ux" or "−vx" which is the operation related to the x-term where s, t, u, and v are four constants. In Fig. 10.2, there are only left–right buttons, "+2" and "−1", in the left most screenshot. After the character steps on the trap, which are the green buttons, the left–right buttons will then be changed into up–down buttons, "+1x" and "−2x", which is shown in the middle screenshot of Fig. 10.2.

Another way to increase the difficulty of Algebra Maze is to create "fake paths" in the maze (Figs. 10.3 and 10.4). We define a "fake path" as a path that seems to be a possible path to the treasure box, but it will, in fact, lead the player to a dead end. In the following, we give illustrative examples on how to combine the fake path and the "trap" together and to design four mazes in which their topology structures look similar but actually possess different difficulty levels. All of these four mazes are with clues {+ 2, −1, +1x, −2x}. The starting point marked as dark blue, and the treasure box marked as yellow, and the orange squares represent the walls and obstacles, respectively.

In Maze 1, which is the easiest one, there are two possible paths (red path and green path) that the player can take in order to reach the treasure box. In Maze 2, we limit the number of paths that the player can take by adding some obstacles, which are indicated by the light blue square in the second figure, and now the player can only reach to the treasure box along the red line. In this example, the original green path in Maze 2 is a "fake path".

This is one possible way to enhance game-play design such that the difficulty of Maze 2 is higher than that of Maze 1. In Maze 3, we have added some traps to

www.dbooks.org

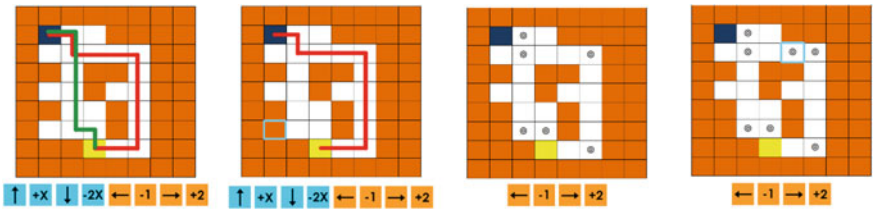


Fig. 10.4 Four different game designs of algebra maze with difficulty levels ordered from easy to hard: Maze 1, Maze 2, Maze 3, and Maze 4

the maze, and the difficulty level will then further increase. Players need to think carefully how to overcome obstacles in order to reach the treasure box. The circles in the third figure represent the traps (the green button in the screenshot). At the beginning, there are only two movement control buttons, namely, the go-leftward and go-rightward buttons. When the player steps on the traps, the movement control buttons will then toggle to upward and downward buttons. In Maze 4, the difficulty level can then be further increased by adding a dead end in the map. This means that, when the player reaches the dead end, he/she will be trapped in that position and cannot move further anymore. In such a case, the player either loses the game or can retry it. The dead end is highlighted in light blue in the fourth figure. By leveling up the maze gradually from Maze 1 to Maze 4, the Algebra Maze will become more challenging and requires the player to think deeply before making a move. These four examples also point out that the elements in the clue set are strongly related to the difficulty-level design. For example, we can leverage the Euclid Algorithm with the clue set design. Students can even get to know and learn important foundational computer science knowledge such as how the Euclid Algorithm works. In addition, the positions of the traps limit the players' moves (e.g., the dead end) and also need to be considered when designing the clue set.

10.4 Mathematics Gamification of Algebra Game

In Algebra Game, unlike the Algebra Maze, we split the clues into two parts, one part is the mathematical operators, i.e., addition "+", subtraction "−", multiplication "*", division "÷", and the other part is the operand such as the x-term or the number as shown in Fig. 10.5. 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 = numerical_solution" through a selection of a finite number of given clues. As shown in the right-hand side of Fig. 10.5, if the player "drags" the division button "÷" and "drops" it on the button "2", then the equation will be divided by two on both sides. Algebra Game is not only about solving a linear equation but it also contains other mathematical insights. For example, consider an equation "x − 23 = 2" with the clues "+, −" and "2,

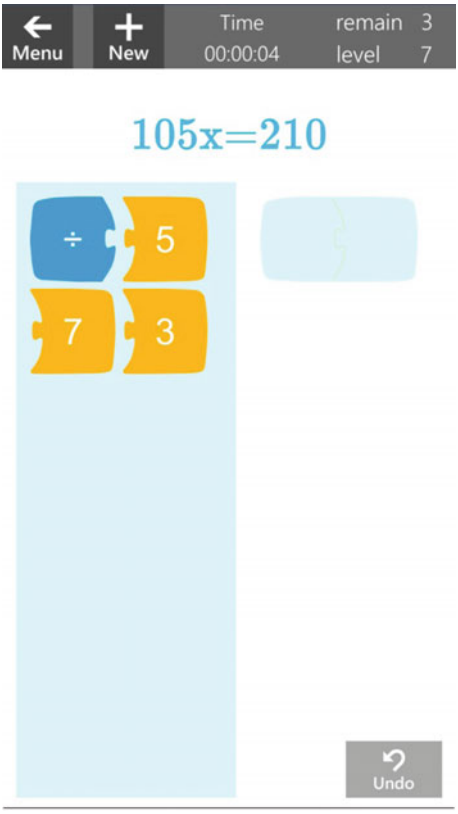


Fig. 10.5 A puzzle instance in the algebra game mobile app game that challenges students to find the right combinations of prime number factors for the number 105 in order to solve the puzzle. Important mathematical results such as the fundamental theorem of arithmetic can be displayed as hint to facilitate a deeper understanding of mathematics for the game players

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 "*, ÷" 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 the shortest sequence of moves to solve the problem. If the player is familiar with the mathematical analysis underpinning the difficulty level design in the Algebra Game, the player can even further develop mathematical insights and intuition while playing the Algebra Game.

www.dbooks.org

The mathematics gamification process also requires analyzing the scoring rule at each puzzle game level that can be evaluated according to different reasonable design criteria. For example, scoring rules 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. A well-designed scoring system can improve the design of the difficulty level at each level for improving students learning experience. 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 the classroom). For example, Fig. 10.5 shows an example of the puzzle. 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 developing their capacity to view mathematics at multiple abstract levels.

10.5 Case Study of Computer Science Challenge Game Tournament

The Computer Science Challenge (CS Challenge) is a tournament organized by the Department of Computer Science at City University of Hong Kong starting since 2016 for both primary and secondary school students in Hong Kong ("CS Challenge", n.d.). The CS Challenge is modeled after eSports-like computer game tournament. 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 in May 2016. One of the computer game in the CS Challenge is the *Algebra Game Challenge*, in which the Algebra Maze and Algebra Game are used for the primary school students (as shown in Fig. 10.6) and the secondary school students (as shown in Fig. 10.7), respectively. The Algebra Game Challenge lasts for a fixed duration of 20 min. 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 Sect. 3 that allows more diverse game-playing dynamics and also enables the use of data collection and data analytics to capture the users' game-playing behavior. The mobile apps in ("The Algebra Game Project", n.d.) were not available to the



Fig. 10.6 Primary school student tournament of algebra maze at the computer science challenge in May 2016

participants beforehand as they were posted online only after the CS Challenge was over (Fig. 10.8).

We describe next the learning efficacy of the first task of Algebra Game and Algebra Maze based on analysis 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 Fig. 10.10, we see that the number of students who have completed the corresponding level has observable reduction at higher levels, e.g., about 20 percent, from Level 9 to Level 10 which can also be observed in Fig. 10.9, the time spent at Level 10 almost doubled that at Level 9. In fact, the number of moves needed at Level 10 is also almost double that at Level 9 as shown in Table 10.1. 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 at 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 at Level 1 is that students are initially warming up (as they get familiar with the game interface and rules). If we omit the information at Level 1 and proceed to compute the correlation coefficient

www.dbooks.org



Fig. 10.7 Secondary school student tournament of algebra game at the computer science challenge in May 2016

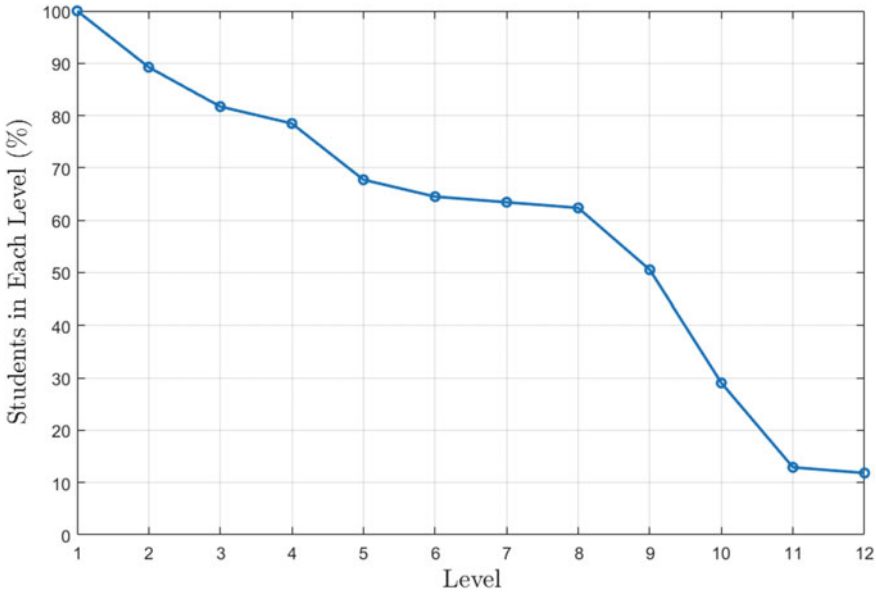


Fig. 10.8 Percentage of the number of students versus the total number of completed levels of algebra game with an unlimited number of levels a priori designed in the algebra game during the 20-min duration for the game tournament

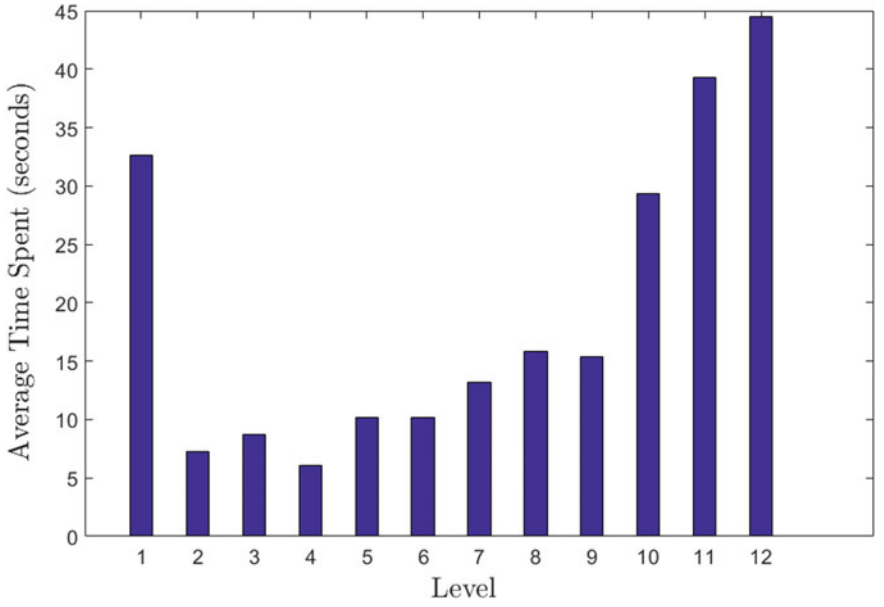


Fig. 10.9 The average time a player spent at each level of the algebra game

Table 10.1 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

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 12 levels being analyzed.

10.6 Further Discussions

From Fig. 10.9, we observe that the average time that students spent on a level in the Algebra Game increases with the level of progression (except for the first level, where students are just getting familiar with the game and learning the rules on how to play the game and thus the time spent is relatively higher). From a game design perspective, an extension is to adapt difficulty levels that are fine-tuned according to the skill or knowledge of the player. In addition, how to classify the difficulty level of the Algebra Game and to integrate that with player's learning curve is an interesting direction that we will address in the future release versions of Algebra Game and

www.dbooks.org

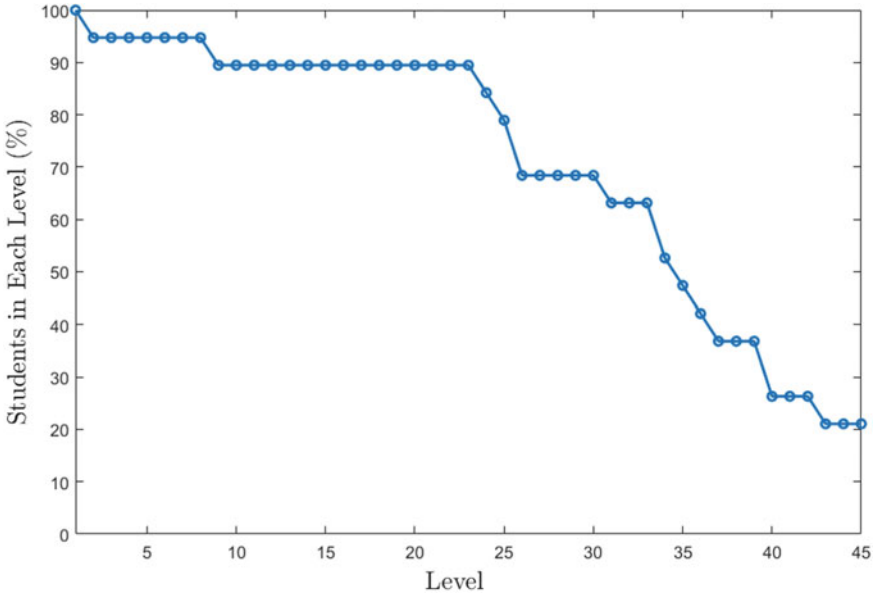


Fig. 10.10 Percentage of number of students versus the total number of completed levels of algebra maze with a total of 45 levels a priori designed during the 20-min duration of the game tournament

Algebra Maze. This will allow the game-playing experience to be adaptive to the personalized learning goal of mastering arithmetical and numeracy skills in algebra manipulations (Fig. 10.10).

As part of classifying the difficulty level of the Algebra Game, we will address the automatic generation of games in the Algebra Game and Algebra Maze so that high-quality levels can be generated at real time. There are several ways to classify difficulty levels of the Algebra Game and Algebra Maze. One possibility is a machine learning approach to quantify automatically the difficulty level in the Algebra Game of the puzzle. We can treat this as a simple regression problem in machine learning: the input is the puzzle parameter of an Algebra Game level (initial equation, number of moves allowed, set of allowed operations, etc.), and the output is a numeric value measuring the difficulty of the given level. We can train this model by feeding it with the puzzles we used in the game tournaments, e.g., the Computer Science Challenge, and their corresponding difficulty measures derived from the average time spent by human players. After the training, we can use the generated model to predict the difficulty level of a particular generated level of Algebra Game. This can, in turn, be presented to general users in the form of automatically generated Algebra Game puzzles with increasing difficulties.

Another enhancement we are currently making to the algebra gamification system is a personalized puzzle recommendation system, where the backend server can push auto-generated personalized Algebra Game/Algebra Maze puzzles to each individual student's device. We call it "personalized" because the system will determine a

specific student's need (e.g., a student might need more practice in factorization of numbers, which can be seen from her past playing record in the system) and push the most suitable puzzles to the student. Once we finish the difficulty level classification task, the system can also push puzzles at different difficulty levels to suitable student groups, so all students from different skill levels can find the game challenging.

10.7 Conclusions

We described preliminary ideas of mathematics gamification and their application to teach computational thinking by cultivating the capacity for logical thinking and problem-solving skills of students using mobile app game software. Through gamifying elementary algebra learning, we described how the logical manipulatives of the mathematical puzzle games, 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 related to abstract algebra. Through competitive game playing in a Computer Science Challenge tournament, we studied the learning efficacy of the algebra gamification software and also experimented with scaling up mathematics learning using mobile app software. In our future work, we are exploring the possibility of extending the ideas mentioned in this book chapter to gamification of other advanced mathematics and computer science subjects in a single mobile app—The Polymath App ("The Algebra Game Project", n.d.).

References

Computer Science Challenge Game Tournament in Hong Kong. http://cschallenge.cs.cityu.edu.hk.

Devlin, K. (2011). *Mathematics education for a new era: Video games as a medium for learning* (1st edn.). A. K. Peters, Ltd.: Natick, MA, USA.

Lavernia, E. J., Vander Gheynst, 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.

Mackay, R. F. (2013, March 1). *Playing to learn: Can gaming transform education?* Graduate School of Education, Stanford University. https://ed.stanford.edu/news/playing-learn-can-gaming-transform-education.

Minsky, M. (1970). Form and content in computer science—1970 ACM turing award lecture. *Journal of the Association for Computing Machinery, 17*(2), 1970.

Novotney, A. (2015). Gaming to learn. *Monitor on Psychology, 46*(4), p. 46. http://www.apa.org/monitor/2015/04/gaming.aspx.

Pope, H., & Mangram, C. (2015). Wuzzit trouble: The influence of math game on student number sense. *International Journal of Serious Games, 2*(4), 5. http://documents.brainquake.com/backed-by-science/Stanford-Pope-Mangram.pdf.

Shapiro, J. (2013, August 29). *Video games are the perfect way to teach math, says Stanford Mathematician*, *Forbes magazine*. https://www.forbes.com/sites/jordanshapiro/2013/08/29/video-games-are-the-perfect-way-to-teach-math-says-stanford-mathematician/#7e90367a385b.

www.dbooks.org

Tao, T. (2012a). *Gamifying algebra.* Terence Tao's Wordpress blog article. https://terrytao.wordpress.com/2012/04/15/gamifying-algebra
Tao. T. (2012b). *Software mockup of algebra game*. https://scratch.mit.edu/projects/2477436 (main Page, n.d.).
The Algebra Game Project. https://algebragame.app.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 11
Mathematics Learning: Perceptions Toward the Design of a Website Based on a Fun Computational Thinking-Based Knowledge Management Framework



Chien-Sing Lee and Pei-Yee Chan

Abstract Mathematics is core to many scientific disciplines as it trains the mind in representation, logic, and various data manipulations. Mathematics is thus the power of reasoning, creativity, problem-solving, and effective communication skills. Mathematics can, however, be difficult to learn if interest and teaching–learning approaches do not link theory with real-world problems. Hence, the goal of this study is to enhance the learning of Mathematics using a multi-stakeholder, fun Computational Thinking (CT)–Knowledge Management (KM) framework. We hypothesize that this framework will help improve stakeholders' perceptions toward the learning of Mathematics and CT as elements of fun can transform knowledge more productively. Brennan and Resnick's (2012) CT involves three aspects: computational perspective, computational practice, and computational concepts. These are implemented in different ways to suit different contexts and needs. Furthermore, KM encourages knowledge creation and sharing in order to produce more innovative outcomes. The findings of this study indicate that there is an increased positive perception toward the Website's objectives, usefulness, and ease of use. We hope that by enabling students to create, share, and discuss fun-based problems for others to solve, they will gain confidence in their own understanding of and creativity in problem-solving.

Keywords Computational thinking · Computational perspective · Computational practice · Computational concepts · Fun knowledge management framework

11.1 Introduction

The application of computational thinking (CT) can be found in most institutions and agencies today. Wing (2006) defines CT as "a universally applicable *attitude* and *skill set* everyone, not just computer scientists, would be eager to learn and use." With

C.-S. Lee · P.-Y. Chan (✉)
Department of Computing and Information Systems, Sunway University, Petaling Jaya, Malaysia
e-mail: chanpy95@hotmail.com; 13018197@imail.sunway.edu.my

C.-S. Lee
e-mail: chiensingl@sunway.edu.my; csleester@gmail.com

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_11

www.dbooks.org

regard to attitude, CT encourages an open mindset challenging boundaries and pursuing refinements. An open mindset corresponds with (a) reformulating a seemingly difficult problem into one solution familiar to us, either by reduction, transformation, or simulation; (b) choosing an appropriate representation of a problem or modeling the relevant aspects of a problem; (c) thinking recursively; and (d) processing alternatives and constraints simultaneously. These further point toward two key computer science concepts, i.e., decomposition and abstraction. Consequently, CT enables better identification of data patterns, better understanding and solving of problems, systems and human behavior, and more effective analysis.

Brennan and Resnick (2012) aim to reach out to the masses, through CT. Hence, they have simplified CT into three aspects: (a) computational perspectives, (b) computational practice, and (c) computational concepts. The Computational Thinking (CT) course website (2017) supports Wing's (2006) and Brennan and Resnick's (2012) propositions, and regards CT as comprising of four components: (a) decomposition, (b) pattern recognition, (c) algorithmic thinking, and (d) generalization.

To us, CT is a means to an end, whereby it can help make the teaching of programming more effective. CT is also an end in itself due to the progressive developmental nature of learning. We adopt the four CT components at the CT course website for this study as the website integrates propositions by eminent founders and practitioners.

11.1.1 Problem

Knowledge of Mathematics is essential in everyday life. From the moment, we wake up in the morning to the minute we go to bed, we depend on our understanding of and capability to perform simple Mathematics-based tasks. Mathematics hones the power of reasoning and problem-solving, leading to better clarity in communication skills and higher creative possibility (Chan, 2010; Snalune, 2015; Biswas, 2015; Kong, 2019).

Our research interest focuses on the use of Mathematics to teach CT skills in a fun way. Mathematics is closely linked to CT as it involves a *pattern recognition* of a problem structure and variables that can be instantiated with different values, such as *decomposition*, to enable modularity and easier problem-solving; *algorithm design*, due to its emphasis on logical reasoning; and *generalization*, i.e., from multiple examples to formulation of principles. Thus, if CT can be used to find the best means of solving problems, then more creative solutions can be ideated and found to overcome difficulties.

Furthermore, increasingly, children are spending much time on the Internet and video games, even violent ones. The American Academy of Child and Adolescent Psychiatry (2015) points out that video game addiction can cause mental health and social problems to children since most of their time will be spent indoors playing online/video games. Furthermore, playing video games for long hours can affect academic performance since students will have problems focusing in class and interacting with others. Christopher (2015) confirms that even when interacting with

others, they tend to only discuss video games. As time passes, their social communication will cease and they will have problems communicating with members of society in general.

Considering the benefits of learning Mathematics and CT skills, it is proposed that children's interest should be diverted from unhealthy video games to more educational, Mathematics-based games. Furthermore, if we can make learning more fun and enjoyable, we can incorporate Knowledge Management to enhance collaborative learning.

11.1.2 Objectives

An earlier study by Lee, Wong, and Ee (2017) use an object-oriented gamification approach to improve students' Mathematical and CT skills. The target students in the study are those studying in university. In this study, our framework adopts an inquiry-based student-centered learning framework, with primary school students (Primary 1–3) as the target users.

Student-centered learning (Table 11.1) supports the use of inquiry approaches such as games. Knowledge Management (KM) approaches complement such inquiry as KM involves knowledge acquisition, sharing, creation, and dissemination. These provide a structure to knowledge-building and collaborative learning.

Moreover, in Kong's (2019) 11 case studies on teacher professional development, he found that for scalability of teaching practices, content and teacher development models need to be developed from the perspective of a multi-stakeholder partnership. Subsequently, principles can serve as top-down strategies to be tested in classrooms, where successful patterns of e-learning can be identified in the long run to enrich students' and teachers' theoretical and practical understanding. Hence, a community approach is adopted in this study, involving parents, community members, and young children, since Primary 1–3 students are still very young and need motivation and support from their parents and peers/community.

In addition, elements of fun are crucial to the very young. Abelson's (2012) reconstructible computational media is one of the early works of fun visual programming. Eventually, fun-based Scratch (Resnick, 2007) came about. Fun in this study is game-based and refers to Koster's (2004) theory of fun for game design. Koster notes that game design requires pattern recognition, and variations from these patterns and inherent game mechanics will pose interesting challenges/inquiries to players.

As such, in this study, we enable students to tinker with Mathematical concepts and practice in various ways to refine their computational perspectives. We hypothesize that designing using Brennan and Resnick's (2012) CT, Ventakesh and Davis' (2000) Technology Acceptance Model, and *a fun KM approach* will improve stakeholders' perception toward (a) the importance of learning mathematical concepts; (b) the mastery of Mathematical (curricular) knowledge and skills; (c) the mastery of critical thinking and communication skills; and (d) online learning.

www.dbooks.org

Table 11.1 Characteristics of student-centered learning which supports the KM approach

Categories of classroom practice	Teacher-centered (TC)	Student-centered (SC)
	Teacher-directed Primarily didactic	Student-directed Primarily interactive
Teacher role	☑ Present information ☑ Manage classroom	☐ Guide discovery ☑ Model active learning ☐ Collaborator (sometimes learner)
Student role	☐ Store, remember information ☑ Complete tasks individually	☑ Create knowledge ☑ Collaborator (sometimes expert)
Curricular characteristics	☑ Breadth – focused on externally mandated curriculum ☐ Focus on standards ☐ Fact retention ☐ Fragmented knowledge and disciplinary separation	☐ Depth – focused on student interests ☑ Focus on understanding of complex ideas ☑ Application of knowledge to authentic problems ☐ Integrated multidisciplinary themes
Classroom social organization	☐ Independent learning ☐ Individual responsibility for entire task	☑ Collaborative learning ☑ Social distribution of thinking
Assessment practices	☐ Fact retention ☐ Product oriented ☐ Traditional tests ☐ Norm referenced ☐ Teacher-led assessment	☑ Applied knowledge ☑ Process oriented ☑ Alternative measures ☑ Criterion referenced ☐ Self-assessment and reflection
Technology role	☐ Drill and practice ☐ Direct instruction ☐ Programming	☑ Exploration and knowledge construction ☑ Communication (collaboration, information access, expression) ☑ Tool for writing, data analysis, problem-solving
Technology content	☐ Basic computer literacy ☐ Skills taught in isolation	☑ Emphasis on thinking skills ☑ Skills taught and learned in context and application

Source Ertmer, Ottenbreit-Leftwich, Sadik, Sendurur, and Sendurur (2012)

This chapter is presented in the following order: literature review, methodology, game design, user testing, findings, significance, and finally conclusions.

11.2 Literature Review

11.2.1 Computational Thinking (CT)

Computing involves not only programming and practicing computing skills but also recursive thinking, model matching, and compositional reasoning. These will enhance algorithmic thinking, parallel thinking, and practical thinking as well. As a result, through CT, the learner develops problem-solving processes and dispositions. These are very important to the advancement of computer systems and functions, especially analysis and design as well as justifications to problems and solutions (Togyer & Wing, 2017).

CT shows how individual knowledge and skills gained from computing subjects can bring positive impact to society. The CT course website (2017) provides some examples. In Economics, one can find cycle patterns in the rise and fall of a country's

economy; in Mathematics, the steps to solve a calculus problem; in Chemistry, the principles for ionization. Hence, people who understand and have knowledge of CT in their studies will see the correlation between theoretical subjects and the environment (between indoor/formal learning and outdoor/informal learning).

Abelson (2012) further points out that for CT to grow beyond re-constructible computational media, it needs to be complemented by computational values which promote open access. This has partly led to the popularity of open-source movements. Consequently, CT's objective of building individual capability and creativity (Weintrop, Holbert, Horn, & Wilensky, 2016) and the open-source movements have influenced many professions and computer science curricula in developing countries.

11.2.2 Game-Based Learning and Gamification

According to Kazimoglu, Kiernan, Bacon, and MacKinnon (2012), digital gameplay involves thinking of the correct solution for the players to move on to the next level. This motivation has driven the video gaming industry to constantly move forward. According to the global game-based learning market, the world profits for game-based learning products reached $2.6 billion in 2013 and are expected to gain a profit of $7.3 billion by 2021. In 2016, a total of $99.6 billion in revenue was earned from worldwide games, which led to an 8.5% increment compared to the revenue earned in 2015 (Adkins, 2016).

There are different types of games including educational games, adventure games, action games, and role-playing games. Table 11.2 presents some of the benefits of these games mentioned in prior literature.

11.2.3 Knowledge Management

KM revolves around knowledge acquisition, sharing, translation/transformation, and dissemination (Nonaka & Takeuchi, 1991; Frappaolo, 2006; Maier, 2010; Jashapara, 2011). This approach is chosen as we would like the students to follow the KM cycle and finally create their own games using Scratch.

11.3 Methodology

First of all, the Software Development Life Cycle (SDLC) is used to design the game and to ensure the quality of the online game portal. For user requirements, we have carried out a questionnaire to survey students and parents' opinion and preferences with regard to our objectives and the proposed Website portal. Next, we have designed and developed the Website according to their opinion and preferences

www.dbooks.org

Table 11.2 Summary matrix of conference and journal article reviews

Functions/Journal	The educational benefits of video games	Effects of game technology on elementary student learning in mathematics	Playing video games could boost children's intelligence (but Facebook will ruin their school grades)	A surprising new study on how video games impact children
Improve problem-solving skills	+	+	+	+
Increase task variety		+		+
Develop and improve social skills				+
Improve technology and education skills	+	+		+
Sharpen and apply knowledge in school	+	+	+	+

and the KM cycle. Stakeholders' involvement is critical and as such, their feedback weighed heavily on our design and development. Consequently, we have adopted agile methodology to design and develop incrementally to cater to user feedback. For design and analysis, ease of use and usefulness, the two key assessment metrics of the Technology Acceptance Model (Ventakesh & Davis, 2000) are also applied in alpha and beta testings.

11.3.1 Website Component Design Based on Computational Thinking (CT)

We designed the Website portal based on the recommendations of Wing (2006), Brennan and Resnick (2012), and the CT course website based on the four main CT concepts: (a) dividing the processes into parts (decomposition); (b) knowing and finding the correct patterns for problem-solving (pattern recognition); (c) writing algorithms for the user to solve problems (algorithm design); and (d) determining the rules and principles of a particular subject (abstraction/generalization).

The mapping of the Website components to the CT concepts is in Fig. 11.1.

Website aims at progressive and incremental improvement, corresponding to Bloom's taxonomy. The most basic Mathematics concept is number counting. Hence, the initial level is to present odd and even numbers to familiarize students with numeric values, and at the higher level is the concept of interval.

From numeric values, *pattern recognition* is developed by progressing through (a) multiple examples in worksheets and (b) Scratch programming. The worksheets,

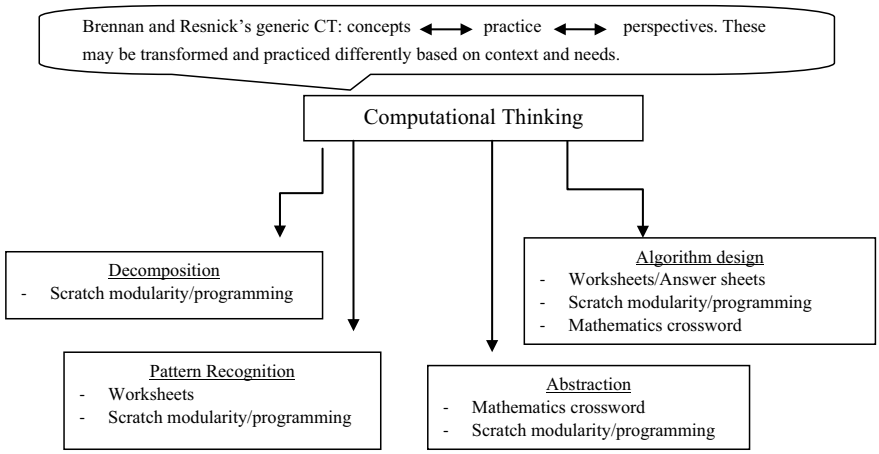


Fig. 11.1 Mapping of website components to CT concepts

which are of different difficulty levels, enable students to not only go through the steps for each concept exercise but also to inductively realize that many problems are similar. In Scratch programming, building blocks can be reused and remixed for different fun outcomes.

Next is *algorithm design* and later, *abstraction*. For solution steps (*algorithm design*), diverse formulas/methods of solving problems can lead to the same answer. We choose a Mathematics crossword puzzle because it enables students to identify different solutions to the same problem. This leads to *generalizing*.

Due to the iterative nature of agile-based SDLC methodology, the sections below present the progression/iterative design process and user testings, which have led to the subsequent versions of the Website.

11.4 Pilot Test: Preliminary Design and Analysis

In the pilot test, a total of 27 responses from different age groups are gathered through a questionnaire to identify their opinion and preferences. Questions are scored on a range of one (not important) to five (most important). There are five different age groups: children (6 years old to 12 years old), teenagers (13 years old to 19 years old), young adults (20 years old to 35 years old), adults (36 years old to 59 years old), and senior citizens (60 years old and above). The five different age groups are categorized as such to derive a more holistic understanding of factors, which would sustain both learning and future system development.

Young adults make up 40.7% of the sample, whereas adults only make up 7.4%. There are 18.6% more female respondents compared to male respondents. During

www.dbooks.org

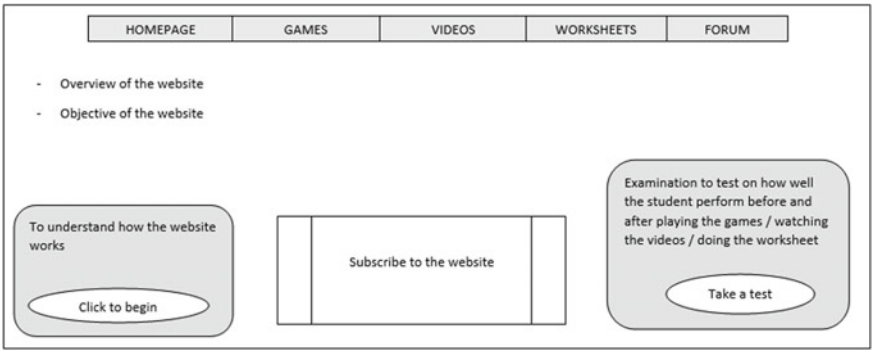


Fig. 11.2 Website's wireframe (main page)

the pilot test, the participants are shown the Website's main page in wireframe format (Fig. 11.2), followed by the survey questions.

A summary of the responses received from the participants is presented in Table 11.3. The feedback obtained is used to further refine the proposed Website portal.

Table 11.3 Summary of the responses from pilot test participants

Question	Answers
How long does the respondent usually spend time studying online?	• More than two hours (37%) • Two hours (22.2%) • One hour (22.2%)
Importance of Mathematical concepts	• Very important (52%)
Importance of understanding the use of Mathematics in the real world	• Important (44%)
Whether they agree that learning Mathematics through games would help improve student's performance	• Agree (92.6%) • Disagree (7.4%)
Whether they agree the best way for students to learn is to find solutions to problems on their own	• Agree (85.2%): If they find the solution, they will get to experience different ways of tackling problems. This can help them gain extra knowledge and is one of the ways to train them to be independent • Disagree (14.8%)
Motivators in learning	• Creativity and discussion (63%), reward (51.9%), feedback (44.4%)
Purpose for having an educational game	• Extra revision (44.4%) • Reward and group work (44.4%)

11.5 Alpha Testing: Design and Development

The purpose of the alpha testing is twofold: (a) to ensure that all functions in the Website portal are working; and (b) to obtain user feedback to refine the system. Users can have an overview of the Website portal on the homepage (Fig. 11.3). They can also access the Mathematics test to determine and improve their level of Mathematics.

Worksheets are divided according to the students' age group to help narrow down the worksheets to the targeted age group:

- For Primary 1, basic Mathematics equations for users are included to familiarize them with numbers and equations.
- For Primary 2, the multiplication timetable and basic problem-solving skills are included.
- For Primary 3, the focus is mainly on money value and problem-solving skills.

Answer sheets are provided for each worksheet so that users can check their performance and see different methods of solving a problem. Figures 11.4, 11.5, and 11.6 are sample screenshots of the questions and answers.

A forum page for parents and students is also included in the Website to enable discussion, information sharing, and knowledge exchange. Moreover, the forum can be used to help users improve their social skills. The game created using Scratch is found in "Fun Time".

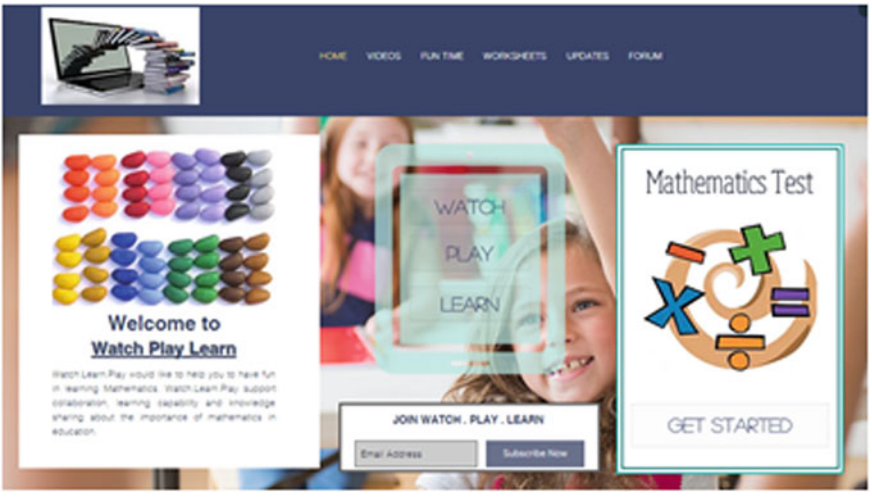


Fig. 11.3 Website homepage

www.dbooks.org

Question	Working
Write the following numerals in words. i) 45 ii) 99	 i) Forty-five ii) Ninety-nine
Answer True or False for the following questions. i) 5 is less than 10 ii) 8 is greater than 7 iii) 2 is greater than 4 iv) 99 is less than 999	 i) True ii) True iii) False iv) True

Fig. 11.4 Primary 1 sample worksheet (numeric value, comparison)

Question	Working
Jeremy has 90 marbles. He gave John 45 marbles. How many marbles does he left?	$\begin{array}{r} {}^{8}\not{9}\,\not{0}^{10} \\ -\ 4\ 5 \\ \hline 4\ 5 \end{array}$ Jeremy left 45 marbles.
There are 12 chocolates in 3 boxes. How many chocolates are there in a box?	$12 \div 3 = 4$ There are 4 chocolates in a box.

Fig. 11.5 Primary 2 sample worksheet (number value, pattern recognition with regard to numbers, numeric value, and operators)

11.5.1 Alpha User Testing

For the alpha user testing, a total of 31 responses from different age groups are collected. Similar to the pilot test, there are five categories of respondents. For the question regarding the *importance of understanding Mathematics in the real word*, 35.5% of them said that it is important. From the range of one (not important) to five (most important), none of them said that Mathematics is not important.

As for the *overall rating* of the Website, 16.1% of the respondents rate it as average, while most users (61.3%) give it a rating of 4. To further elicit user requirements/expectations, user feedback is compiled (Table 11.4) and considered for further testings.

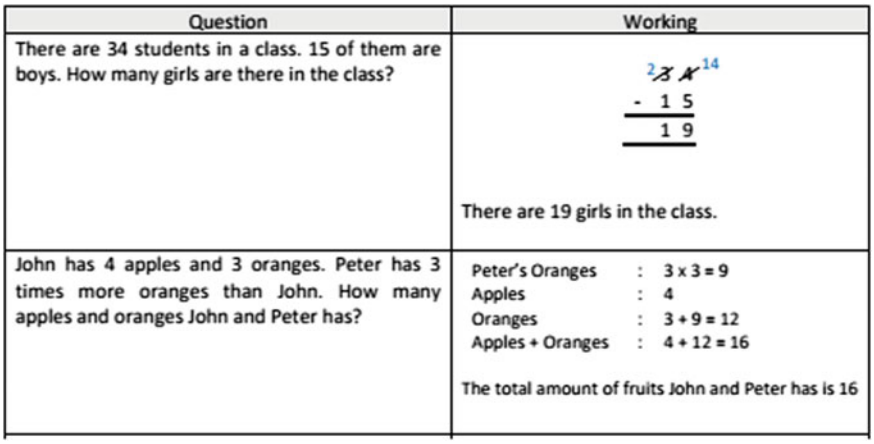


Fig. 11.6 Primary 3 sample worksheet (numeric value, pattern recognition with regard to numbers, numeric value, operators, and algorithm design)

Table 11.4 User's expectations

What do you expect from the website and game?
Keep up the good work!!
Critical thinking skills in solving Mathematical problems (2)
Knowledgeable and fun, enable the user to watch and learn with fun, something interesting
Improve design, strategy game
Add more things, more exercise, more worksheets and give answers, more pictures

11.6 Beta Testing

For the final evaluation process, the design is based on the alpha users' perceptions and experience. We first present the design for the game and then the Website. Users can select to start the game or to understand the game instructions. The voice narration for the game is downloaded from the Watson text-to-speech demo.

Once users start the game, a brief introduction is presented. There are four stages in this game. The first and second stages of the game require users to select all the colored balls containing even and odd numbers in order to proceed to the next level. As they proceed with the game, they would get points if they manage to answer the given questions correctly. Once users manage to complete all the stages, they are required to purchase the "tickets" with their points to watch the space videos as a reward. The reward can demonstrate to users that mastering Mathematical skills is key to earning points. Knowledge translation is involved when users apply Mathematical skills that they have learnt to identify the correct answers.

www.dbooks.org

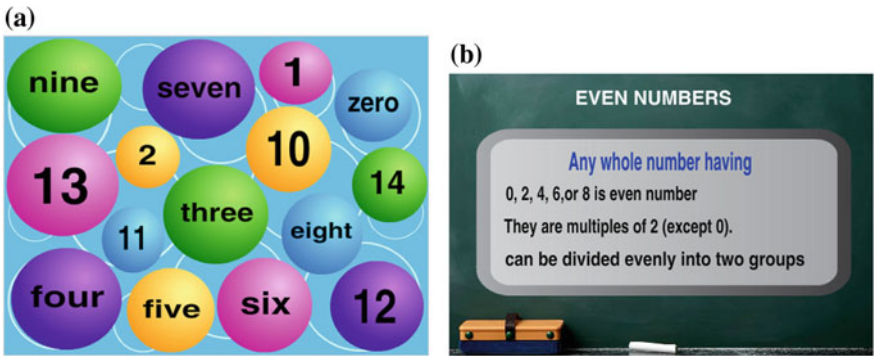


Fig. 11.7 **a** Even number game (number value), **b** even number clip (number value)

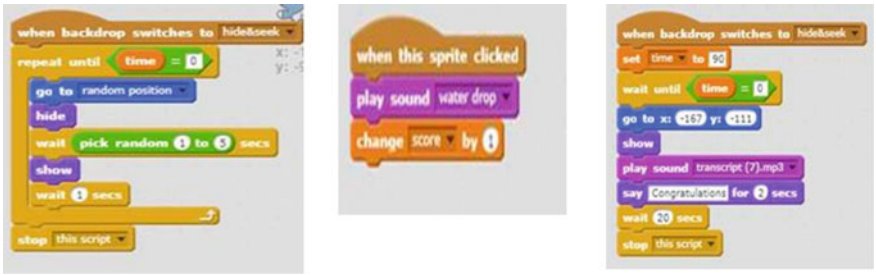


Fig. 11.8 Hide-and-seek code (object-orientation, decomposition, variables, behavior, values)

If users do not know what even and odd numbers are, they can watch a short clip explaining and teaching them what these numbers are. Figure 11.7a, b is sample screenshots of the even number game and clip.

Next is the game and/or reward using Scratch (Fig. 11.8). Students can manipulate the actions, variables, and values for different outcomes.

Figure 11.8 shows a hide-and-seek game developed in Scratch. The seeker is given 90s to find the hider, who can hide at any random position. This game will be repeated until the time runs out. After a couple of seconds, the seeker will show up and start seeking the hider. If the seeker finds the hider at the x- and y-coordinates (e.g., −167, −111), then the hider will be revealed and the user score will increase by 1 point. The word "Congratulations" will then be displayed to let the user feel a sense of achievement.

The variability in values enables users to manipulate the pace of the game, as well as the kinds of encouragement and sound effects they would like to include in the game. Water drop in this example may not be appropriate. So it may be the first value to be changed.

Finally, a Mathematics crossword puzzle enables users to click on the empty boxes within the crossword to fill in the answers. The purpose of the crossword is to show users that similar Mathematical functions such as addition and subtraction can

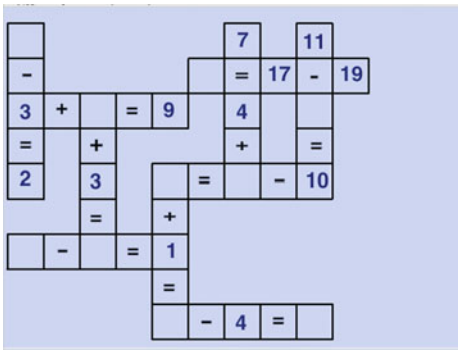


Fig. 11.9 Mathematics crossword puzzle

be found in different Mathematical equations, and that there are different components/methods to arrive at a solution. In Fig. 11.9, users have to think of the solution from multiple perspectives, simulating higher level algorithm design. For example, there are two ways to arrive at the total 10. We hope that with practice, users will understand the link between algorithm design and numeric value, operators, and problem-solving.

Since we are trying to encourage learning in a community, a blog site enables posting of updates regarding the subject matter. The forum enables them to share ideas, knowledge, and problem-solving among users and their parents.

11.7 Comparison Between Alpha–Beta User Testings

For beta testing, a total of 30 responses from five different age groups are collected. The age groups include children (6 years old to 12 years old), teenagers (13 years old to 19 years old), young adults (20 years old to 35 years old), adults (36 years old to 59 years old), and senior citizens (60 years old and above). 56.7% of the respondents comprises the young adults and 26.7% of them are adults.

Table 11.5 shows the comparisons between the alpha- and beta testing results, while Table 11.6 presents user expectations for the future Website and game development. The Likert scale ranges from 1 (least important) to 5 (most important).

Although the perception outcomes are consistent and positive, the stakeholders mainly prefer simple designs and manageable exercises and games. Furthermore, users may find it challenging to find the solutions to problems all by themselves. Hence, adaptations in the sequence and choice of activities would be necessary.

www.dbooks.org

Table 11.5 Comparisons in perceptions between alpha–beta testing findings

TAM constructs	Evaluation questions	Alpha testing				Beta testing			
	Likert scale	2	3	4	5	2	3	4	5
Usefulness	Important to understand the use of Mathematics in the real world?		6.5%	58.1%	35.5%	3.3%	6.7%	36.7%	53.3%
Usefulness/ease of use	Rating of the website?		16.1%	61.3%	22.6%		16.7%	53.3%	30.0%
	Do they think the website…			*Yes*	*No*	*Yes*		*No*	
Usefulness	Will improve students' Mathematics (curricular) skills?			96.8%	3.2%	93.3%		6.7%	
Usefulness	Will improve students' critical thinking and communication (extra-curricular) skills?			96.8%	3.2%	96.7%		3.3%	
Usefulness	Is informative for Primary 1 till Primary 3 learners?			93.5%	6.5%	93.3%		6.7%	
	Do they think the website and game …			*Yes*	*No*	*Yes*		*No*	
Usefulness	Will deliver Mathematics knowledge to children?			87.1%	12.9%	96.7% (game & knowledge most important)		3.3%	
Intention to use the system	Will they allow their kids (presently or in the future) to experience online learning?			5 (22.6%) 4 (61.3%) 3 (16.1%)		96.7%		3.3%	

Table 11.6 Expectations for future *Website and Game* development

What do you expect from the website and game?	
Alpha testing	Beta testing
Critical thinking skills in solving Math problems (2)	To learn and advance in Mathematics
Knowledgeable and fun, enable the user to watch and learn with fun, something interesting, strategy game	To improve my knowledge of Mathematics and have fun. An interesting game with Mathematics
Improve design	More design, pretty user interface
Add more things, more exercise, more worksheets, more pictures, and give answers	Card games that need to apply simple calculation according to standard of difficulty
	It is not the content of the website. Instead, it is the intention of the person using the website/game

11.8 Significance

The role of technology in inquiry-based, student-centered learning is not always easy. The crux of student-centered learning framework is the derivation of principles, and this is more likely to happen if teachers themselves teach using a principle-based approach. Various studies such as Song and Looi (2012) and Kong and Song's (2013) studies have confirmed this reasoning.

The significance of our study lies in:

(a) Validating our design and obtaining technology acceptance via iterative cycles of awareness creation toward the usefulness of each design component. Acceptance of the design and technology implies that the Website content can inspire children's interest in learning Mathematics and that the qualities inspired by Mathematics such as the power of reasoning, creativity, critical thinking, problem-solving ability, and effective communication skills can be more easily developed in the future. This is in line with Chan's (2010) compelling, sustainable, and "disseminable" experimental digital classroom and MIT's Scratch approach.

(b) Contributing to the larger body of research on motivational orientations such as the prominent self-directed studies by Lao, Cheng, Huang, Ku, Chan (2017). Our study is aimed at developing epistemic agency based on the Learning Sciences and the Institute of Electrical and Electronic Engineering's (IEEE) thrust, building on past work by Lee and Wong (2017). Although our study is slightly different from that of Lao, Cheng, Huang, Ku, Chan's (2017) in terms of epistemologies/orientations, we hope our study can help contribute to the larger extant work. It may also be interesting to work with Chien, Chen and Chan (2017).

www.dbooks.org

11.9 Conclusion

Parents should start introducing Mathematical concepts to their children from young. In order to attract their attention to learning, methods of teaching and learning should be fun as it might be difficult to gain their interest once they have lost it.

The design of this study has incorporated Brennan and Resnick's (2012) computational perspective, computational practice, computational concepts, and KM approaches. CT-based Mathematics provides a direct link to principles. Hence, it is important to develop such skills. Our fun Computational Thinking-Knowledge Management (CT-KM) design and approach follow an inquiry-student-centered learning framework. The role of technology in terms of exploration, construction as well as communication in inquiry-student-centered learning parallels that of KM's knowledge acquisition, sharing, creation, and dissemination.

To validate our designs for the learning of Mathematics and CT, the opinions and feedback of users from different age groups are obtained through user testings and questionnaires. Although the target users of the Website are primary school students, comments from young adults, adults, and senior citizens are taken into consideration as they are the stakeholders who will influence and encourage the younger children.

Our findings indicate that the perceptions of different stakeholders toward CT-based Mathematics learning have improved as the Website provides different ways of learning to suit different contexts and needs. It is our hope that schools in Malaysia will also see more creative outcomes to problems. Furthermore, we hope that by enabling students to create, share, and discuss fun-based problems for others to solve, they would feel encouraged to create and share knowledge, and eventually, have a greater appreciation of Mathematics and CT.

Acknowledgements The first author would like to thank Prof. Tak-Wai Chan for his question *what is Mathematics learning?* while she was a Faculty at the Graduate Institute of Network Technology, National Central University, Taiwan, ROC. Thanks also to MIT's Scratch for paving the way, the 2017 Computational Thinking conference for highlighting what CT is and Dr. K. Daniel Wong for prior collaboration on design thinking and CT.

References

Abelson, H. (2012). From computational thinking to computational values. In *Special Interest Group on Computer Science Education (SIGCSE)* (pp. 239–240).

Adkins, S. S. (2016). The 2016–2021 Worldwide game-based learning market. Ambient Insight. In *Serious Play Conference*. Retrieved April 30, 2017.

American Academy of Child and Adolescent Psychiatry. (2015). *Video Games and Children: Playing with Violence*, Retrieved April 12, 2017, from http://www.aacap.org/AACAP/Families_and_Youth/Facts_for_Families/FFF-Guide/Children-and-Video-Games-Playing-with-Violence-091.aspx.

Biswas, C. (2015). The importance of Maths in everyday life. *Guwahati News*. Retrieved April 15, 2017, from http://timesofindia.indiatimes.com/city/guwahati/The-importance-of-maths-in-everyday-life/articleshow/48323205.cms.

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.*

Chan, T. W. (2010). How East Asian classrooms may change over the next 20 years. *25th Anniversary Special Issue of Journal of Computer-Assisted Learning*, January, 2010.

Chien, T. C., Chen, Z. H., & Chan, T. W. (2017). Exploring long-term behavior patterns in a book recommendation system for reading. *Educational Technology & Society, 20*(2), 27–36.

Christopher, D. (2015). *The negative effects of video game addiction. Livestrong.com*. Retrieved April 12, 2017, from http://www.livestrong.com/article/278074-negative-effects-of-video-game-addiction.

Computationalthinkingcourse.withgoogle.com (2017). *What is computational thinking? Computational Thinking for Educators*. Retrieved October 27, 2017, from https://computationalthinkingcourse.withgoogle.com/unit.

Ertmer, P. A., Ottenbreit-Leftwich, A. T., Sadik, O., Sendurur, E., & Sendurur, P. (2012). Teacher beliefs and technology integration practices: A critical relationship. *Computers & Education, 59*(2), 423–435.

Frappaolo, C. (2006). *Knowledge management* (2nd ed.). New York: Capstone Publishing.

Jashapara, A. (2011). *Knowledge management: An integrated approach*. London: Pearson.

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.

Kong, S. C. (2019). Partnership among schools in e-Learning implementation: Implications on elements for sustainable development. *Educational Technology & Society, 22*(1), 28–43.

Kong, S. C., & Song, Y. (2013). A principle-based pedagogical design framework for developing constructivist learning in a seamless learning environment: A teacher development model for learning and teaching in digital classrooms. *British Journal of Educational Technology, 44*(6), 209–212.

Koster, R. (2004). *Theory of fun for game design*. New York: O'Reilly Media.

Lao, A. C. C., Cheng, H. N., Huang, M. C., Ku, O., & Chan, T. W. (2017, Jan). Examining motivational orientation and learning strategies in Computer-Supported Self-Directed Learning (CS-SDL) for Mathematics: The perspective of intrinsic and extrinsic goals. *Journal of Educational Computing Research, 54*(8), 1168–1188.

Lee, C. S., Wong, J. W., & Ee, P. Y. (2017). Gamified mathematics practice: Designing with e-commerce and computational concepts. In *International Conference on Computational Thinking*, July 13–15, 2017, Hong Kong.

Lee C. S. & Wong K. D. (2017). An entrepreneurial narrative media-model framework for knowledge building and open co-design. In *IEEE SAI Computing*, July 18–20, 2017, London, UK.

Maier, R. (2010). *Knowledge management systems: Information and communication technologies for knowledge management* (3rd ed.). Heidelberg: Springer.

Nonaka, I., & Takeuchi, H. (1991). The knowledge-creating company: How Japanese companies create the dynamics of innovation. *Harvard Business Review, 69,* 96–104.

Resnick, M. (2007). Sowing the seeds for a more creative society. In *Learning and leading with technology*, December–January, 2007–2008, 18–22.

Snalune, P. (2015). The benefits of computational thinking. *ITNOW, 57*(4), 58–59. Retrieved November 4, 2017, from http://www.bcs.org/content/ConWebDoc/55416.

Song, Y., & Looi, C. K. (2012). Linking teacher beliefs, practices and student inquiry-based learning in a CSCL environment: A tale of two teachers. *International Journal of Computer-Supported Collaborative Learning, 7*(1), 129–159.

Togyer, J., & Wing, M. J. (2017). Research notebook: Computational thinking–what and why? *Carnegie Mellon School of Computer Science*. Retrieved November 2, 2017, from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why.

Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science, 46*(2), 186–204.

www.dbooks.org

Weintrop, D., Holbert, N. S., Horn, M., & Wilensky, U. (2016). Computational thinking in constructionist video games. *International Journal of Game-Based Learning, 6*(1), 1–17. Retrieved November 2, 2017 from http://ccl.northwestern.edu/2016/WeintropHolbertHornWilensky2016.pdf.

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

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part IV
Computational Thinking in K-12 STEM Education and Non-formal Learning

www.dbooks.org

Chapter 12
Defining and Assessing Students' Computational Thinking in a Learning by Modeling Environment



Ningyu Zhang and Gautam Biswas

Abstract Learning sciences researchers have hypothesized the connections between STEM learning and computational thinking (CT), and this has been supported by studies that have demonstrated the reciprocal relationships between CT and STEM subjects. However, not much work has been done to establish the fundamental set of CT knowledge and practices that need to be taught to enhance STEM and CT learning in K-12 curricula. Therefore, many important aspects of CT are underrepresented in K-12 classrooms. We believe that CT concepts and practices are not only important for computing education but also play an important role in helping students develop modeling and problem-solving skills in STEM domains. In this chapter, we build on our existing framework (Sengupta et al in Educ Inf Technol 25(1):127–147, 2013) to promote students' synergistic learning of science content and CT practices in middle school classrooms. We discuss the primary STEM and CT concepts and practices that we have introduced through our curricular units into science classrooms and discuss how students can learn these practices in CTSiM, a computer-based learning environment developed in our lab (Basu et al in User Model User-Adapt 27, 2017). We present results from studies in middle school classrooms with CTSiM and show that students have strong learning gains in Science and CT concepts. Our assessments also help characterize and understand students' learning behaviors and their link to learning and practicing STEM and CT concepts.

Keywords Computational thinking · Domain content knowledge · Theoretical framework · Learning by modeling · Assessments

N. Zhang · G. Biswas (✉)
Department of Electrical Engineering and Computer Science, Institute for Software Integrated Systems, Vanderbilt University, 1025 16th Avenue South, Nashville, TN 37212, USA
e-mail: gautam.biswas@vanderbilt.edu

N. Zhang
e-mail: ningyu.zhang@vanderbilt.edu

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_12

12.1 Introduction

Recently, Computational Thinking (CT) has been recognized as a framework for developing computer literacy and computing skills among the K-12 computer science (CS) and STEAM (Science, Technology, Engineering, Arts (and Humanities) and Mathematics) communities (Barr & Stephenson, 2011; Grover & Pea, 2013). Industry and governments also consider CT to be one of the primary drivers of the twenty-first--century workforce (Barr, Harrison, & Conery, 2011). In general, CT encompasses a wide range of abilities and practices, such as problem decomposition and composition, algorithm development, reasoning at multiple levels of abstraction, and creating and reusing modular solutions (Wing, 2006). Researchers and practitioners believe that by drawing from the fundamental skills and practices of CT, students can develop analytical and problem-solving skills and practices. These CT practices go beyond the learning of CS and benefit students' understanding of scientific processes, systems design, and human behaviors (Wing, 2006; NRC, 2010; Barr & Stephenson, 2011; NRC, 2011). In other words, CT can benefit K-12 students' learning in other domains such as mathematics and science as they solve problems while *thinking like a computer scientist* (Wing, 2011; Barr & Stephenson, 2011).

A series of studies have shown that applying CT in STEM domains helps students' learning (Basu & Biswas, 2016; Basu et al., 2017; García-Peñalvo, Reimann, Tuul, Rees, & Jormanainen, 2016; Weintrop et al., 2016a). Additionally, CT skills and practices can transfer to other learning and problem-solving contexts (Basawapatna et al., 2011; Grover, 2015), as CT requires a deep understanding of problem-solving when compared against rote learning (Wing, 2006). Therefore, it provides an essential framework for preparing students for future learning (Bransford, Brown, & Cocking, 2000). In addition, Brennan and Resnick (2012) point out that CT includes the practice of designing artifacts (e.g., building models of STEM phenomena), which helps students develop perspectives of the world around them (e.g., a deeper understanding of the role of vegetation in reducing greenhouse gases). Therefore, CT provides a synergistic framework for learning of computational and science concepts and practices (Sengupta et al., 2013; Basu et al., 2017).

The acknowledgment of these potential benefits of CT has led to the inclusion of CT into the K-12 STEM curricula; for example, the Next Generation Science Standards (NGSS) in the United States includes CT as a core scientific practice (The NGSS Lead States, 2013; Barr & Stephenson, 2011). Researchers have also stressed the urgency of introducing CT into K-12 classrooms. However, there has been insufficient effort to establish a key set of CT concepts and practices across K-12 classrooms, and little effort has been made to include CT concepts and practices into other disciplinary curricula.

In this chapter, we extend our earlier framework (Sengupta et al., 2013; Zhang & Biswas, 2017) by

1. Introducing CT with an emphasis on STEM practices through an open-ended learning environment (OELE); and

www.dbooks.org

2. Evaluating students' synergistic learning of science content and CT practices in middle school classrooms.

In Sect. 12.2 of this chapter, we review existing frameworks for CT learning and assessment. In Sect. 12.3, we introduce our STEM + CT framework, the primary CT skills and practices, and our OELE to promote the synergistic learning of domain content and CT. We also discuss how these learning constructs can be assessed in our STEM + CT framework. Finally, in Sect. 12.4, we use the results from our middle school classroom studies to discuss the design of assessments that help us analyze how students develop and exploit the synergy between the STEM and CT concepts.

12.2 Related Work

Wing's seminal paper describes CT as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). Another popular definition states that CT is a problem-solving process that includes characteristics such as problem formulation, data analysis, abstraction, algorithmic automation, solution refinement, and transfer (ISTE & CSTA, 2011). Given the wide scope of CT, there are different ideas of what constitutes CT, and how to integrate CT into the K-12 curricula, especially for STEM topics (NRC, 2010; Brennan & Resnick, 2012). In addition, the close relationships between CT, mathematics, algorithmic thinking, and problem-solving can sometimes veil the core ideas in computation that CT encompasses (García-Peñalvo et al., 2016; Weintrop et al., 2016a). Nevertheless, the various definitions of CT provide the research community and educational stakeholders with many insights into how its various concepts and practices can benefit learning.

Some effort has been made to establish operationalizable frameworks for CT. These frameworks either focus on the constructs that emerge from existing game-based and media-narrative programming environments, or they emphasize STEM concepts and practices. The CT frameworks associated with AgentSheets (Repenning et al., 2000) and Scratch (Brennan & Resnick, 2012) are examples of the former type, and the CT taxonomies proposed by Weintrop et al. (2016a) are examples of the latter type.

The CT framework (Basawapatna et al., 2011) in AgentSheets focuses on the aspects of CT that can be transferred to other learning contexts. In their framework, CT is defined as the recurring program patterns that students acquire while programming games and later reuse in other simulation contexts (Basawapatna et al., 2011). Brennan and Resnick (2012) defined CT as a framework of three components related to programming in Scratch: *what* students should know about programming, *how* to program, and the *sociocultural* aspects of programming. More specifically, computational concepts of their framework refer to the fundamental knowledge of computing, such as how loops and conditionals work in a Scratch program. Computational practices are defined as programming related actions, such as building and

debugging. Finally, computational perspectives describe the learner's computation-related world-view (Brennan & Resnick, 2012).

These frameworks operationalize programming-centered CT constructs in existing environments, but they do not provide explicit evidence of how CT is linked to STEM learning. On the other hand, the CT taxonomies proposed in Weintrop et al. (2016a) emphasize the application of CT in STEM classrooms. Weintrop et al. proposed key CT practices that are naturally linked to STEM concepts and practices (NGSS, 2013), including (1) data, (2) modeling and simulation, (3) problem-solving, and (4) systems thinking. In addition to focusing on *what* students learn about CT, these CT practices define *how* students can learn and apply CT, thus providing a theoretical foundation for integrating CT in STEM classrooms.

Some CT frameworks also include the assessment of CT. For example, computational thinking pattern analysis (CTPA) in AgentSheets matches the recurring program patterns in game design and science simulation contexts to evaluate students' understanding of CT (Basawapatna et al., 2011). Additionally, Scratch uses multimodal assessments including project analyses and interviews to assess its main CT constructs (the CT concepts, practices, and perspectives) (Brennan & Resnick, 2012). These CT frameworks provide important information on students' understanding of CT, but they have their shortcomings as well. For example, students' expression of CT is demonstrated at the program level, assuming that the student understands CT if they use a CT pattern. On the other hand, the *used = learned* assumption poses peril because a student writing correct programs nevertheless may not have made the necessary conceptual connections (NRC, 2010). In addition, these analyses of snapshots of completed programs lose the temporal information and the subtlety to understand students' developmental process of CT. As a result, many fundamental aspects of CT have not received sufficient attention especially in the context of block-based programming environments, except for a few successful assessments (e.g., the Fairly Assessment in Alice, Werner, Denner, Campe, & Kawamoto, 2012; Grover & Pea, 2013). Therefore, more detailed, reliable and formative test instruments need to be developed to enrich CT assessments.

12.3 The STEM + CT Framework

Studies have shown that CT and STEM subjects shared a reciprocal relationship. There is evidence in the literature that students improved their understanding of STEM topics when they are studied in a CT framework (e.g., Basu et al., 2017; Sengupta et al., 2013; Weintrop et al., 2016a, b). Similarly, developing CT concepts and practices in a science learning framework provides a context and a perspective for the better understanding of CT. For example, the NRC (2010) report states that CT concepts and practices are best acquired when studying them within domain disciplines. If students were introduced to CT in programming contexts only, they might not develop the skills to apply the generalized CT concepts across disciplines because of the difficulties in the transfer of learning (NRC, 2011). Additionally, learning CT

www.dbooks.org

concepts and practices in a STEM modeling and simulation framework provides students with the real-world perspective (Brennan & Resnick, 2012) they may need to develop a good understanding of the STEM and CT concepts in context. Therefore, our CT framework uses an integrated STEM + CT approach to foster students' synergistic learning of domain content knowledge and CT through computational modeling and simulation. In the rest of the section, we introduce our STEM + CT framework, the CTSiM learning environment, and the assessment schemes.

12.3.1 The STEM + CT Framework

Figure 12.1 gives an overview of our STEM + CT framework. Central to the framework is the **Practices** applied across STEM domains that use CT methods. There are four types of practices: *Systems Thinking, Problem-solving, Modeling and Simulation,* and *Data and Inquiry*. The STEM + CT practices are the means to support students' synergistic learning and understanding of **Domain Content** and **CT concepts**. The CT concepts include *variables and assignments, sequential execution of statements, loop structures, conditionals, functions*, and *events*. These CT concepts are fundamental to most programming environments.

We use Computational Thinking using Simulation and Modeling (CTSiM) as the **Learning Environment** in our framework to help students foster the key Domain and CT concepts and practices. CTSiM (Basu et al., 2017; Sengupta et al., 2013) is an open-ended learning environment (OELE) that helps students achieve the syner-

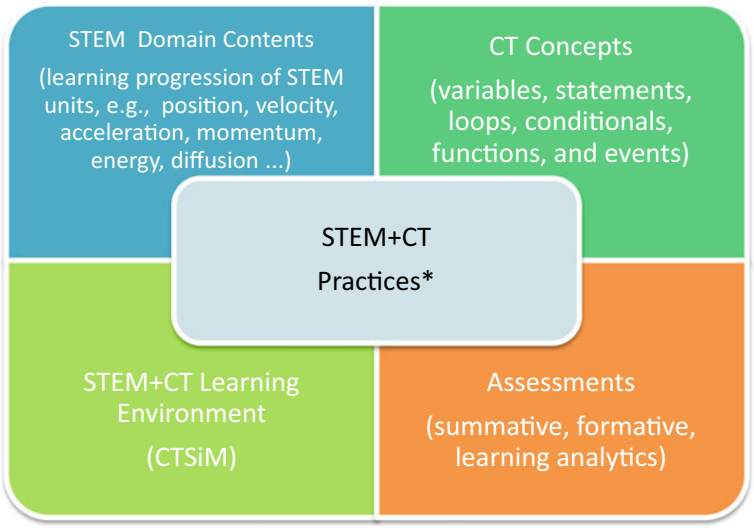


Fig. 12.1 The STEM + CT framework

gistic learning of science knowledge and CT practices using a learning by modeling approach. Additionally, the CTSiM Learning Environment encapsulates **STEM Domain Content** in a progression of units. Finally, the summative and formative **Assessments** evaluate students' improvement in the STEM content and CT concepts, as well as their model-building and problem-solving behaviors and performance.

In this framework, students can synergistically learn STEM and CT concepts and practices as they apply them to develop computational (specifically, simulation) models of the scientific phenomena. Meanwhile, the domain content provides the learning context for how to apply the CT concepts through opportunities for model building and simulation of specific scientific phenomena that students can easily relate to because of their links to real-world phenomena.

Table 12.1 presents the set of STEM + CT practices defined in our framework. It includes four categories of practices commonly applied in STEM domains and 14 practices instantiated in the CT learning environment. Among the four categories, Systems Thinking is the overarching practice as the deep understanding of systems with emergent behaviors is essential for students to prepare for future scientific endeavors (Cheng et al., 2010). However, students can develop misunderstandings when they attempt system-level analyses in an agent-based modeling framework (Wilensky & Resnick, 1999). On the other hand, developing systems understanding

Table 12.1 Related STEM + CT practices

Systems thinking
(ST1) identify the relevant components that make up a system
(ST2) understand component relationships and interactions
(ST3) understand the system at different levels of abstraction, and the behaviors that emerge from these interactions
Modeling and simulation
(MS1) conceptual modeling (formal and abstract thinking)
(MS2) computational modeling (algorithmic thinking and implementation)
(MS3) simulation and execution of models
(MS4) Interpreting and understanding system behaviors (emergence)
Problem-solving
(PS1) dividing into sub-problems
(PS2) modularity and reuse
(PS3) debugging and error-checking
Data and inquiry
(DI1) acquire information from texts
(DI2) set up and run experiments to collect and analyze data
(DI3) organize and display data (table, graphs)
(DI4) interpret plots and other data representations

www.dbooks.org

by applying CT practices provides an approach to help students overcome these problems.

Within our proposed framework, students are expected to study the interactions among agents and between agents and the environment. They can then translate this understanding to identify variables and the relations between relevant variables, and, in this process, study emerging behaviors at different levels. Additional STEM practices, such as *Data collection, Analysis*, and *Inquiry*, *Modeling and Simulation*, and *Problem-solving,* help students develop a deeper understanding of Systems Thinking. Together, these set of fundamental STEM + CT practices serve as the methodology to help learners synergistically acquire the required CT and STEM concepts.

12.3.2 The Learning Environment

In CTSiM, students can model scientific phenomena using a block-structured language that is framed in an agent-based modeling approach (Sengupta et al., 2013). Learners can leverage six types of primary learning tasks in CTSiM. More specifically, students can (1) acquire information of the domain content and CT-related programming concepts from the built-in hypertext resource libraries; (2) represent their system in terms of agents, their properties, and their behaviors using an abstract conceptual model representation; (3) construct runnable computational (i.e., simulation) models that define the agents' behaviors and interactions in the environment; (4) execute the computational models as NetLogo (Wilensky, 1999) simulations to examine the agents' behaviors; (5) compare the behaviors of their computational models to the behaviors generated by an expert model; and (6) conduct science inquiry to understand the relationship between components in a system and the behaviors generated by the components. These learning activities are closely related to and can help students' foster STEM + CT practices. For example, reading the hypertext libraries supports the *Data and Inquiry* practice of acquiring information from texts. The *Data and Inquiry* practice also involves interpreting plots and other data representations. Similarly, constructing the conceptual and computational models directly relates to *Modeling and Simulation* practices. In addition, running the computational models and comparing their results to ones generated from an expert simulation support students *Problem-solving* practices, such as debugging and error-checking. We will discuss in more detail the affordances of the environment in supporting STEM + CT concepts and practices in the introduction to the *Assessment* framework.

The units in CTSiM are developed to complement the middle school STEM curricula with modeling and simulation activities. CTSiM offers a number of units, forming a STEM-learning progression from modeling motion of objects using turtle geometry to investigating complex, emergent systems, such as diffusion of particles in a liquid medium. The curricular modules of CTSiM include (1) turtle geometry, (2) kinematics, (3) mechanics, (4) collision and momentum, (5) particle collision and energy, (6) fish tanks and the carbon cycle, and (7) role of bacteria and the nitrogen

cycle in fish tanks (Basu et al., 2017; Sengupta et al., 2013; Zhang, Biswas, & Dong, 2017; Zhang & Biswas, 2018).

Figure 12.2 shows the user interface of four primary activities in CTSiM: (1) reading the science library, (2) editing the conceptual model, (3) building the computational model, and (4) comparing the behaviors generated by the students' models to those generated by an expert model. In this example, the student used computational blocks from a domain-specific block-based modeling language to implement the update-speed-and-direction behavior of dye molecule agents. When executed, the animation on the left depicts the motion of the molecules as modeled by the student. Students can compare the motion of molecules in their simulation to the motion of molecules in the expert simulation on the right side of the compare tab. In addition, students can also compare the changes in key simulation variables over time to the expert model. Note that students cannot see the implementation of the expert simulation model; they can only observe the behaviors it generates.

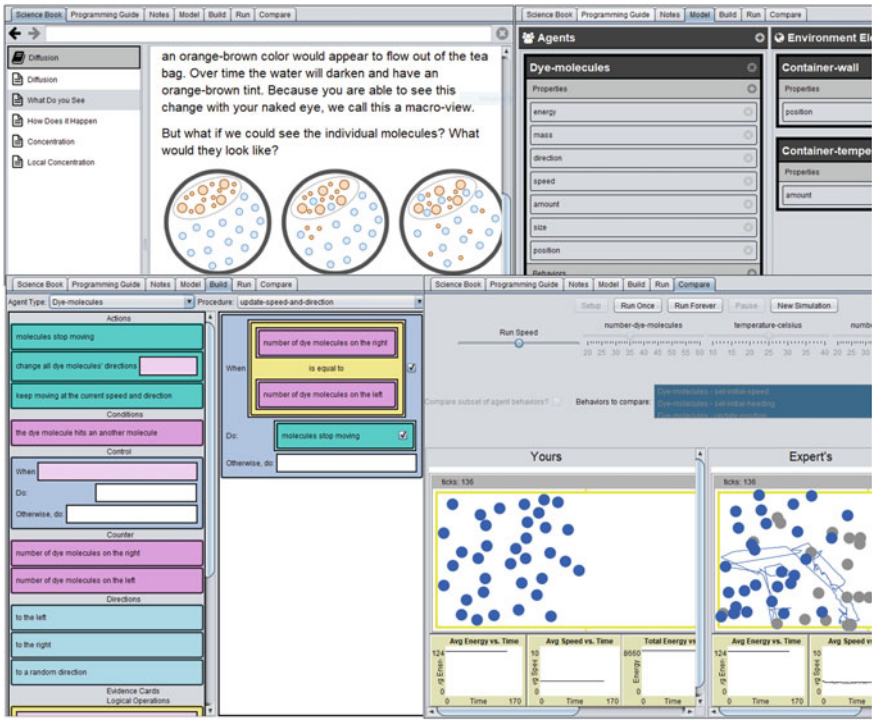


Fig. 12.2 The user interface of CTSiM

www.dbooks.org

12.3.3 The Assessment Framework

Assessments provide information about students' understanding of the content knowledge in a particular modeling unit, and how well they can apply this knowledge to other problems. This information, therefore, can help teachers, researchers, and other stakeholders evaluate the effectiveness of the teaching and learning processes adopted in the approach (Mislevy, Almond, & Lukas, 2003). The summative and formative assessments, as well as the analytics measures that we can derive from students' models, help us evaluate students' progressions in the STEM content and CT concepts, as well as their model-building and problem-solving behaviors and performance. Students' improvement of understanding the STEM domain content and CT concepts are primarily evaluated by the Domain and CT pre- and post-tests. In addition, the STEM + CT practices are assessed in a summative and formative way, using a range of modalities that include: (1) evaluating the correctness of student-generated models and the evolutionary trajectories of these solutions; and (2) analyzing the sequences of actions students perform in CTSiM.

To formally define students' behaviors in the learning environment, we use a hierarchical model to characterize students' actions in CTSiM. Following the combined theory- and data-driven framework developed from Coherence Analysis (CA) (Segedy, Kinnebrew, & Biswas, 2015; Kinnebrew et al., 2017), we define three types of primary sub-tasks in CTSiM: *Information Acquisition* (IA), *Solution Creation* (SC), and *Solution Assessment* (SA). These primary sub-tasks can be further decomposed. Figure 12.3 shows that the lowest level of the hierarchy represents the different actions that students can perform in the CTSiM environment. For example, SA tasks consist of running, checking, comparing, and testing models.

The conceptual models of CTSiM provide an abstract representation for defining agents and the properties of the environment that affect system behaviors. While constructing the conceptual model, students select the properties and behaviors of the agents that they believe are required to construct the model. The set of properties and behaviors that students choose from may include some that are unrelated to the specific modeling task, and being able to identify and reason about whether or not to include a particular property or behavior is an important modeling practice. We have used the "bag of words" (Piech, Sahami, Koller, Cooper, & Blikstein, 2012) methods to examine the correctness of students' conceptual model compared to the

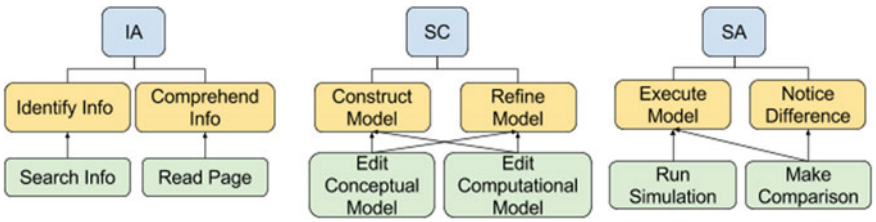


Fig. 12.3 The task model of CTSiM

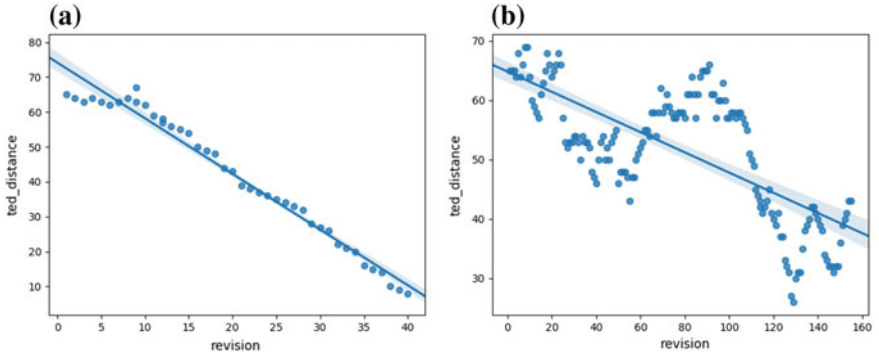


Fig. 12.4 Following two students model building progression using the TED measure

expert models. Another type of summative assessment is based on students' model building performance. We use two measures: (1) model distance, and (2) model trajectory. We calculate the distance between a student-created model and an expert model using the minimal tree edit distance (TED, Bille, 2005) metric, where the models are encoded as abstract syntax trees. In addition, we analyze a student's model building trajectory by computing the slope of distance between the student's models and the expert model as the student performs a sequence of model building actions.

A large model distance indicates an incorrect model, whereas a steeper negative slope indicates faster convergence to the final model. Figure 12.4 shows two students: A's (left) and B's (right) trajectories in building their computational model for the diffusion unit. Both students started with the same empty computational model that had a TED score of 65. Figure 12.4 indicates that student A used much fewer edit actions (~40 edit actions) to build the model compared to student B (~160 edit actions), in building a final model, and the model also had a much smaller final TED (~5 compared to ~45). In summary, Table 12.2 lists the performance expectations and the modality and evidence of the assessment methods for each corresponding practice.

12.4 Results and Discussion

We use results from a recent classroom study to demonstrate students' synergistic learning of STEM and CT concepts and practices with our STEM + CT framework. In the study, 52 6th-grade students from a middle school in the South-Eastern U.S. participated over 14 school days. The students worked on the five units of CTSiM: the two training units, the acceleration unit, the collision unit, and the diffusion unit. Table 12.3 summarizes the daily schedule during the students' hour-long science block. In the rest of the section, we discuss the three types of analyses: (1) the

www.dbooks.org

Table 12.2 The assessment modality and evidence for the STEM + CT practices

	Assessment modality and evidence
ST1	Summative: the correctness of the conceptual models;
ST2	Summative: the domain and CT pre-post questions and the correctness of the computational models;
ST3	Summative: the domain and CT pre-post questions; Formative: average computational edit chunk size
MS1	Summative: the correctness of the conceptual models; Formative: SC-related CA metrics;
MS2	Summative: the correctness of the computational models Formative: SC-related CA metrics;
MS3	Formative: SA-related CA metrics;
MS4	Formative: CA metrics associated with SA → SC, and SA → IA transitions
PS1	Formative: aggregated IA-, SC- and SA-related CA metrics;
PS2	Formative: computational model evolution and SC-related CA metrics;
PS3	Formative: aggregated SA and SC-related CA metrics; CA metrics associated with SA → SC (SA actions followed by SC actions) transitions
DI1	Formative: IA-related CA metrics (supporting action percentage, duration, etc.);
DI2	Summative: domain and CT pre-post questions; Formative: inquiry learning activities in the system;
DI3	Summative: domain and CT pre-post questions; Formative: inquiry learning activities in the system; SA- and SC-related CA metrics
DI4	Summative: answers to the problem-solving inquiry questions; Formative: inquiry learning activities;

learning gains in the STEM domain content knowledge and key CT concepts; (2) the synergy between STEM and CT learning; and (3) students' application of STEM + CT practices as well as their links to the learning gains and model-building performance.

12.4.1 Overall Learning Gains

The pre-post-test results demonstrate students' improved understanding of the STEM domain content and CT concepts. Table 12.4 summarizes the students' pre- and post-test scores, and test statistics in the Mechanics unit, Diffusion unit, and CT. The max scores for the tests were 28, 12 and 34 points, respectively. We ran the non-parametric Wilcoxon Rank Sum Test to assess the statistical significance of the post-to-pre learning gains. The results in Table 12.4 show that students had significant learning gains in domain knowledge (mechanics and diffusion) and CT concepts with moderate effect sizes.

Table 12.3 Summary of daily activities in the CTSiM study

Day(s)	Description of activities
1	Domain and CT pre-tests
2	Agent-based modeling mini-lecture, tutorial of the CTSiM user interface
3–4	Training units, build software agents that draw turtle geometry shapes, practice assigning values and using loops
5–7	Acceleration unit, model the relationship of force, mass, and acceleration of cars, understand acceleration in the Newtonian physics
8–10	Collision unit, model inelastic collision of spheres, momentum, and conservation of momentum in a system
11–13	Diffusion unit, model diffusion of microscopic particles, understand how temperature influences the speed of particles and that diffusion is an emergent behavior of colliding particles
14	Domain and CT post-test

Table 12.4 Pre-post-test results and aggregated test statistics

Test category	Pre mean (std)	Post mean (std)	p-value	Z-stat	Effect size
Acceleration	13.61 (3.40)	15.24 (4.08)	0.003	2.69	0.26
Diffusion	3.81 (2.24)	5.88 (2.40)	<0.0001	4.27	0.42
CT	14.55 (6.27)	18.11 (6.37)	0.0002	3.52	0.35

12.4.2 The Correlations and Synergies in STEM and CT Learning

We then computed pairwise correlations between the learning gains in CT, Acceleration, and Diffusion units, as well as the accuracy of the students' computational models in the Acceleration, Collision, and Diffusion units. Table 12.5 presents the correlation coefficients (Spearman's ρ) between all pairs of performance metrics. The asterisks (*) indicate statistically significant correlations ($p < 0.05$).

Students' learning gains in CT showed moderately high and statistically significant correlations with the domain gains in the Acceleration ($\rho = 0.32$) and Diffusion units ($\rho = 0.27$), providing some evidence that there was synergistic learning of STEM content knowledge and CT concepts. The fact that students who improved more in their understanding of CT also achieved larger learning gains in the STEM content supports the notion for synergistic learning through the CTSiM intervention.

Table 12.5 also shows that all of the computational model distances were negatively[1] correlated with the CT gains. Two out of three units (Collision and Diffusion) had statistically significant correlations with the learning gains ($\rho = -0.34$ and $\rho =$

[1] A larger model distance from the correct model indicates a more incorrect model. Therefore, the negative correlations between the model distance and learning gains indicates better model building ability is related to better performance in the domain and CT learning gains.

www.dbooks.org

Table 12.5 Correlation analysis

	CT gain	Acc. gain	Diffusion gain	Acc. distance	Collision distance	Diffusion distance
CT gain	–					
Acceleration gain	0.32*	–				
Diffusion gain	0.27*	0.18	–			
Acc. distance	−0.14	−0.07	−0.10	–		
Collision distance	−0.34*	−0.22	−0.13	0.21*	–	
Diffusion distance	−0.28*	−0.19	−0.31*	0.33*	0.32*	–

−0.28), respectively. The relation between the acceleration learning gain and acceleration test score had a low correlation value and was not significant. In addition, students' CT pre-test scores showed low correlations with model-building performance, therefore, it is unlikely that students' prior CT knowledge was a significant factor in their model building abilities. Overall, the results provide evidence of synergistic learning of STEM domain and CT concepts as students worked on their model building tasks, and it seemed to improve as students worked through different units. The students' computational model building performance was consistent across the three units, as all computational model distances showed moderate correlation values (and the correlations were statistically significant).

12.4.3 The Use of STEM + CT Practices

Finally, we show how the Coherence Analysis-derived (Kinnebrew et al., 2017) metrics can help characterize students' application of STEM + CT practices. Coherence analysis provides a framework for defining a number of metrics related to individual tasks students perform in the system (e.g., seeking information, building models, and checking models) (Segedy et al., 2015). An introduction to the CA-derived metrics was provided in Sect. 12.3.3. In previous work, we have developed a number of these measures to analyze students' work in CTSiM (Zhang et al., 2017). In this chapter, we extend the collection of CA-derived measures to characterize the students' use of STEM + CT practices. Due to limited space, we only report the analyses on the Diffusion unit. We first ran a feature selection algorithm to select features that produced higher percentages of the total variance in the feature space. We assumed these features would better distinguish students who used the STEM + CT practices from those that did not. This approach also helped reduce the dimensions of the feature space for

clustering. The eight features obtained after feature selection and their descriptions are summarized in Table 12.6. The features are described as percentages and were computed with respect to the total number of actions performed by a student. For example, the feature "Conceptual Model Edit Effort" of a student accounts for the percentage of computational model edit actions among all her actions in CTSiM. Table 12.6 also lists the STEM + CT practices aligned with the chosen features.

We then clustered the student data using the Expectation-Maximization (Gaussian Mixture Model) algorithm to generate probabilistic models corresponding to each cluster. The Calinski-Harabasz information criteria were applied to select the number of clusters, and applying this measure produced three clusters (Zhang et al., 2017). Table 12.7 reports the mean values and the standard deviations of the eight features for the three clusters. We assumed the feature value probabilities for each cluster would explain the differences in the use of STEM + CT practices among students. We also report the results of single-factor analysis of variance (ANOVA) for each

Table 12.6 Selected features for cluster analyses

Feature	Description	STEM + CT practice category
Domain read time	The total time (in seconds) spent on reading domain content	Data and inquiry (DI1)
Conceptual model edit effort	The percentage of conceptual model edit actions	Systems thinking (ST1) Modeling and simulation (MS1)
Computational model edit effort	The percentage of computational model edit actions	Systems thinking (ST2) Modeling and simulation (MS2)
Model test effort	The percentage of testing actions	Modeling and Simulation (MS3) Problem-solving (PS3)
Model comparison effort	The percentage of comparison actions	Modeling and simulation (MS3) Problem-solving (PS3)
Supported comp. model edit	The percentage of supported computational model edits	Problem-solving (PS1) Modeling and simulation (MS2)
Average computational edit chunk size	The average number of consecutive computational edit actions	Modeling and simulation (MS2)
IA → SC rate	The transition probability from IA actions to SC actions	Data and inquiry (DI1) Modeling and simulation (MS1) Modeling and simulation (MS2)

www.dbooks.org

Table 12.7 Selected features for cluster analyses

Feature	Cluster 1 (n = 23)	Cluster 2 (n = 9)	Cluster 3 (n = 20)	F-score	P-value
Domain pre-test	7.14 (4.20)	6.56 (4.28)	5.84 (4.86)	0.42	0.66
CT pre-test	14.57 (3.22)	15.83 (1.05)	13.95 (0.97)	0.27	0.76
Domain gain	1.96 (2.4)	1.89 (2.8)	2.30 (2.8)	0.11	0.9
CT gain	3.39 (6.7)	1.50 (4.8)	4.68 (4.7)	0.97	0.38
Final computational model distance	7.9 (3.21)	2.1 (1.05)	3.0 (0.97)	4.36	0.01
Computational modeling distance slope	−0.50 (0.07)	−0.62 (0.05)	−0.58 (0.04)	3.37	0.04
Domain read time (s)	291 (454)	182 (124)	163 (141)	0.94	0.39
Conceptual modeling editing effort	5% (0.01)	6% (0.01)	7% (0.01)	4.00	0.02
Computational modeling editing effort	15% (0.04)	14% (0.06)	17% (0.07)	0.91	0.4
Supported computational model edits	13% (0.07)	12% (0.11)	16% (10)	0.89	0.4
Average computational edit chunk size	4.57 (0.8)	6.83 (3.5)	5.01 (1.1)	5.96	0.004
Model testing effort	14% (0.04)	10% (0.06)	10% (0.03)	5.02	0.01
Model comparison effort	27% (0.09)	16% (0.13)	14% (0.05)	9.47	<0.001
IA → SC rate	2% (0.01)	13% (0.13)	3% (0.06)	4.38	0.01

feature in Table 12.7 to investigate if the clusters produced statistically significant differences between each other in their learning performances and the CA metrics.

We checked to see if students' prior knowledge in the science domain or CT influenced their learning behaviors. For example, those with higher prior knowledge may find it easier to generate the correct solutions to problems in the learning environment. However, our analysis shows that the students in the three clusters had similar pre-test scores and similar domain and CT learning gains, but there were distinct differences in their learning behaviors that reflected their application of the STEM + CT practices.

Students in Cluster 1 spent the largest amount of time among the three groups in reading the domain library; however, their IA → SC (performing model building actions after their read actions) transition rates were among the lowest. This indicates that although these students read the most, they were not successful in translating the information acquired (IA) into building models (SC) (the differences between the groups were statistically significant). This indicates that Cluster 1 did not show good mastery and use of *Data and Inquiry* practices. Meanwhile, Cluster 1 students

had the highest number of testing and comparison (SA) actions (42% of their actions were SA actions), and their computational model edit chunk sizes were the smallest (again, the differences between the three groups was statistically significant). The implication is that students in Group 1 tended to use more trial-and-error *Problem-solving* and *Modeling and Simulation* practices. As shown in our previous work (e.g., Basu et al., 2017; Segedy et al., 2015; Zhang & Biswas, 2018), these trial-and-error approaches adversely affect students' modeling tasks. These students also had significantly larger computational model distances (mean distance = 7.9) when compared to the other clusters (mean distance = 2.1 and 3.0, $p = 0.01$). In summary, students in Cluster 1 had the least effective model-building behaviors and they did not achieve high domain learning gains. However, these type of tinkering behaviors were not necessarily negative. Cluster 1 students remained engaged in their model building tasks, and their learning gains in CT were quite strong although not significantly higher than the students in Cluster 2.

Compared to students in Cluster 1, students in Cluster 2 spent less time reading the domain library, yet their IA → SC transition rates were high compared to the other groups. Correspondingly, they achieved the best computational model building performance (mean distance = 2.1 and average computational modeling distance slope = −0.62) with the lowest percentages of SC and SA actions. The combined SC (test + comparison) percentages were the lowest (20%), and the combined SA (conceptual model-building + computational model-building) percentages are also quite low (26%). This reflects good debugging and error-checking practices because they did not need excessive SA—actions while debugging because they were able to pinpoint the issues with their computational models quickly. However, their average computational edit chunk size was the largest, indicating that they did not frequently switch from SC to other types of actions. The students in Cluster 2 had the highest IA → SC rate (13%) while the other two clusters only had (2 or 3%). As a result, although these students did quite well on the practices related to *Data and Inquiry* and *Modeling and Simulation*, their *Problem-solving* skills needed improvement. This was also reflected by the fact that they had the lowest percentages of supported computational model edit actions.

Students in Cluster 3 had very similar patterns of learning activity to Cluster 2 except in two features (average computational model edit chunk size and IA → SC Rates). Post hoc Tukey's Honest Significant Difference Test revealed that the differences in these two features compared to Cluster 2 were statistically significant ($p = 0.02$ and 0.03, respectively). The students in this cluster had high learning gains in the domain and CT tests and were efficient model constructors as seen from the slopes of their computational model TED values over time (they had small final computational modeling distances and steep negative slopes similar to cluster 2). Students in this cluster performed the most computational model edits, and their average model edit chunk sizes were in between those of Clusters 1 and 2, which indicates they successfully applied problem-decomposition practices for the larger model building tasks. They performed the smallest number of SA actions, however,

www.dbooks.org

the percentage of supported computational model edits were the highest among all clusters. This reflects the proficient use of debugging and error-checking practices, which was similar to Cluster 2. However, these students' had a low IA → SC transition rate, which implies that they could improve in their *Data and Inquiry* practices. As a result, though they showed good learning gains in the STEM and CT concepts, their computational model building performance was not as good as the students in Cluster 2.

In summary, the clustering results revealed the differences in the use of STEM + CT practices by the students. Although not all of the learning performance metrics and the CA metrics showed significant differences as suggested by the ANOVA results, analyzing these differences in learning behaviors provided insights and links to the students' learning gains and model-building performance.

12.5 Conclusions

In this Chapter, we stressed the benefit and importance of the synergistic learning of STEM and CT concepts and practices. We extended our previous work on the use of CT concepts and practices in STEM classrooms and refined the STEM + CT framework to develop students' synergistic STEM- and CT- learning. The STEM + CT framework defines (1) the practices that are frequently applied in both STEM and CT learning contexts, (2) the STEM domain content knowledge, (3) the set of key CT concepts required for computational modeling, (4) the open-ended learning environment, CTSiM, that fosters students' learning of these STEM and CT concepts and practices, and (5) the assessment framework that provide summative and formative measures for evaluating student performance and learning behaviors. We then used results from a recent CTSiM classroom study in the U.S. to demonstrate how learning can be defined and analyzed with our STEM + CT framework. The results show that students' model-building performances were significantly correlated to their STEM and CT learning, and students' distinct model-building and problem-solving behaviors in CTSiM were indicative of the model-building performance and learning gains.

In future work, we will refine the set of key STEM + CT practices and the assessment framework such that it will be compatible with other learning modalities, such as collaborative learning. We will generalize the CTSiM framework and extend it to different STEM domains, such as the C2STEM learning environment for high school physics. Our overall goal is to study the feasibility and effectiveness of the CTSiM learning environment in multiple contexts and domains.

References

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology, 38*(6), 20–23.

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., 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—SIGCSE '11*, 245. http://doi.org/10.1145/1953163.1953241.

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.

Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science, 337*(1–3), 217–239.

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).

Cheng, B., Ructtinger, L., Fujii, R., & Mislevy, R. (2010). *Assessing systems thinking and complexity in science (large-scale assessment technical report 7)*. Menlo Park, CA: SRI International.

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., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

International Society for Technology in Education (ISTE), & Computer Science Teachers Association (CSTA). (2011). Operational definition of computational thinking, 1030054. Retrieved from http://www.iste.org/learn/computational-thinking.

Kinnebrew, J. S., Segedy, J. R., & Biswas, G. (2017, April). Integrating model-driven and data-driven techniques for analyzing learning behaviors in open-ended learning environments. *IEEE Transactions on Learning Technologies 10*(2), 140–153.

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.

National Research Council (U.S.). (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, D.C: National Academies Press.

National Research Council (U.S.). (2011). *Report of a workshop on the pedagogical aspects of computational thinking*. Washington, D.C: National Academies Press.

NGSS Lead States. (2013). *Next generation science standards: For states, by states*. Washington, DC: The National Academies Press.

Piech, C., Sahami, M., Koller, D., Cooper, S., & Blikstein, P. (2012). Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 153–160).

Repenning, A., Ioannidou, A., & Zola, J. (2000). AgentSheets: End-user programmable simulations. *Journal of Artificial Societies and Social Simulation, 3*(3), 351–358.

Segedy, J. R., Kinnebrew, J. S., & Biswas, G. (2015). Using coherence analysis to characterize self-regulated learning behaviors in open-ended learning environments. *Journal of Learning Analytics, 2*(1), 13–48.

www.dbooks.org

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., et al. (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.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. In *The 43rd ACM Technical Symposium on Computer Science Education* (pp. 215–220).

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., & Resnick, M. (1999). Thinking in levels: A dynamic systems approach to making sense of the world. *Journal of Science Education and 489 technology, 8*(1), 3–19.

Zhang, N., & Biswas, G. (2017). Assessing students' computational thinking in a learning by modeling environment. In S.-C. Kong (Ed.), The Education University of Hong Kong, Hong Kong (pp. 11–16).

Zhang, N., Biswas, G., & Dong, Y. (2017). Characterizing students' learning behaviors using unsupervised learning methods. In E. Andre, R. Baker, X. Hu, M. M. T. Rodrigo, & B. du Boulay (Eds.), (pp. 430–441). Cham: Springer.

Zhang, N., Biswas, G. (2018). Understanding students' problem-solving strategies in a synergistic learning-by-modeling environment. In C. Penstein Rosé et al (Eds.), *Artificial intelligence in education. AIED 2018. Lecture Notes in Computer Science* (Vol. 10948). Cham: Springer.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 13
Roles, Collaboration, and the Development of Computational Thinking in a Robotics Learning Environment



P. Kevin Keith, Florence R. Sullivan and Duy Pham

Abstract Robotics is a robust vehicle for supporting the development of computational thinking in students. Educational robotics activities unfold in a multidimensional problem space that requires the integration of programming, building, and environmental aspects of the activity. Students working collaboratively with robotics have the opportunity to adopt roles within the group that are aligned to these multiple dimensions (e.g., programmer, builder, and analyst). Group roles are an important element of all collaborative learning, but especially in a Computer-Supported Collaborative Learning (CSCL) environment, as the roles help to regulate group activity and learning. In this observational, microgenetic case study, we investigated the relationship of the roles middle school-aged girls adopted to the collaborative interactions they engaged in, and, ultimately to the development of computational thinking evidenced as a result of role participation. The video and audiotaped data used in this study were collected at a 1-day introduction to robotics workshop for girls. Our mixed methods approach included sequential and qualitative analysis of the behavioral and verbal interactions of two groups of girls (n = 6) who participated in the workshop. Our results indicate that the emergence of distinct roles correlates with periods of collaboration and periods of parallel solo work, which, in turn, had an impact on student's engagement in computational thinking including solution planning, algorithmic and debugging operations, and the design of the robotic device. Moreover, students who engaged in greater levels of collaboration selected more difficult challenges to solve within the robotics environment. Suggestions for future research are provided.

Keywords Robotics · Collaboration · Computational thinking · Girls · Roles

P. Kevin Keith · F. R. Sullivan (✉) · D. Pham
College of Education, University of Massachusetts, Amherst, 813 N. Pleasant St., Amherst, MA 01003, USA
e-mail: fsullivan@educ.umass.edu

P. Kevin Keith
e-mail: pkk@umass.edu; pkkeith@educ.umass.edu

D. Pham
e-mail: dpham@umass.edu

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_13

www.dbooks.org

13.1 Introduction

In this chapter, we examine the relationship of group roles to collaboration and computational thinking in an educational robotics setting. Robotics is an immersive activity (Rieber, 2005) that unfolds in a multidimensional problem space, featuring programming, building, and environmental aspects. Due to this multidimensionality, robotics is an activity that lends itself well to collaborative group learning; students can take on different roles including programmer, builder, and analyst in a team effort to solve the robotics problem. Moreover, the external, 3D nature of two of the three activities (building and testing in the environment), supports reasoning in the third, 2D activity (programming). Robotics is an activity of growing interest for children and youth as evidenced by their global participation in the FIRST Lego League (FLL) (First Lego League, n.d.) Moreover, our recent literature review of robotics studies indicates that it is a collaborative activity that has the potential to develop students' computational thinking abilities (Sullivan & Heffernan, 2016). Yet, while this potential exists, there is little research specifically devoted to understanding the relationship of student collaborative learning with robotics and the development of computational thinking.

Meanwhile, the development of computational thinking is increasingly recognized as an important ability for life in the new millennium. Indeed, in the context of the US, there is a national movement to integrate computer science education into K12 education (National Science Foundation, 2016). This movement is powered by the increasingly ubiquitous nature of computing and computer science in all fields, the growing demand for computer scientists, (US Bureau of Labor Statistics, 2017), and a more diverse corps of computer scientists in the US workforce. Our work is aimed at bringing computer science learning opportunities to girls, in order to improve their awareness and knowledge of computer science, as well as to engage their interest in the field.

In this chapter, we report on a recent study in which we examined how collaborative arrangements in the multidimensional problem space of robotics influences group participation and engagement in computational thinking for middle school-aged girls. As will be demonstrated through our analysis, students who share the main roles afforded by the robotics environment (i.e., programmer, builder, and analyst) were able to develop a greater understanding of robotics and engage in computational thinking more frequently.

We chose to work with middle school-aged girls, and it is at this age that people begin to explore the possibility of future identities (Ji, Lapan & Tate, 2004). And, it is through exposure to an area of study, and opportunities for accomplishments within that area that support a feeling of self-efficacy and further interest in the field (Ali, Brown, & Loh, 2017). Therefore, in conducting research with middle school-aged girls, we aim to create knowledge that will support the development of successful educational interventions for girls that will improve girls interest and participation in computer science. Our chapter is organized in the following way: first, we provide a brief definition of computational thinking from the literature, we then provide

an overview of research on educational robotics, how it relates to computational thinking, and to collaborative learning and role adoption as an emergent quality of participation. We then present our research questions, our methods for investigating these questions, our results, and our interpretation of these results. We end with suggestions for practice and future research.

13.1.1 Computational Thinking

Computational Thinking (CT) reflects the habits of the mind engaged in by computer scientists including, but not limited to "…solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006). CT can be thought of as a broad foundation consisting of the heuristics used by computer scientists and as a way to think about the diverse thinking skills associated with computing. The Computer Science Teachers Association (CSTA) has disseminated a definition of CT as including: formulating problems in ways that enable us to use a computer to solve them, and automating solutions through algorithmic thinking. They further indicate that these skills are important because they create a tolerance for ambiguity, allow for persistence in working with difficult problems, and provide an opportunity to practice collaborating with others to achieve a common goal (Computer Science Teachers Association, 2016). An understanding of the aspects of teaching CT is important because it has been argued that CT skills are essential skills that should be taught to all school-aged learners (Lee, et al., 2011). Not only because these skills are used by computer scientists, but influence an abundance of other fields (Wing, 2006) and people with a command of these competencies will be better positioned to participate in a world where the computer is ubiquitous (Grover & Pea, 2013).

13.1.2 Educational Robotics and Computational Thinking

Robotics kits are computational manipulatives that enable student engagement in computational thinking and doing (Sullivan & Heffernan, 2016). Designed and programmed robotics devices provide immediate, concrete feedback to students on the efficacy of their programs, thereby promoting reflection and debugging analysis (Sullivan & Heffernan). These activities—designing, programming, testing, and debugging—form a troubleshooting cycle students typically enact as they work with robotics systems (Sullivan, 2011). Hence, robotics is an ideal activity for engendering computational thinking and doing in students.

Novices engaged in robotics study, begin by building a robotic device, usually using blueprints provided by Lego as part of the Mindstorms kit. They then learn how to write simple algorithms to make the robotic device move (Sullivan, 2008). This process is facilitated by the graphical programming language used in the Lego

www.dbooks.org

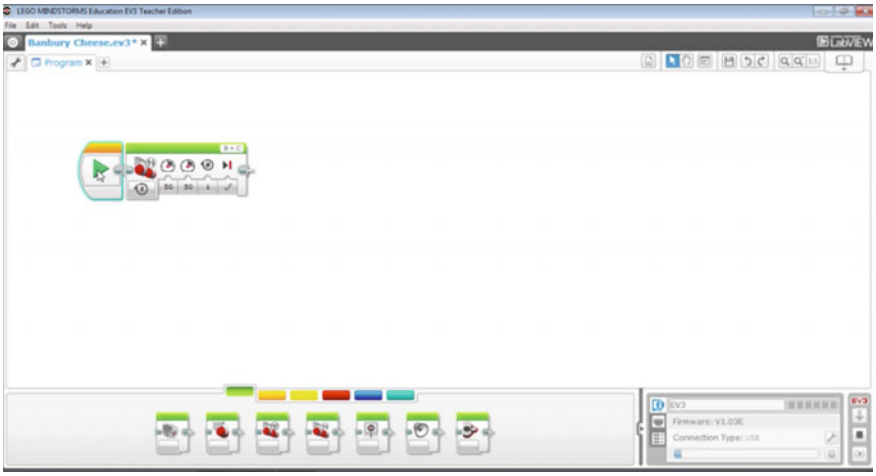


Fig. 13.1 EV3 Mindstorms graphical programming language

Mindstorms robotics kit. This programming language is a modified version of LabVIEW (see Fig. 13.1). LabVIEW is a graphical programming language developed by National Instruments, whose power derives from its ability to make "…computer science accessible to non-programmers. LabVIEW object-oriented programming brings some of the most advanced programming techniques into the grasp of beginning users" (National Instruments, 2014).

The graphical nature of the programming language allows students to immediately begin programming, writing algorithms to move the robot. The physical nature of the robotic device provides students with concrete and immediate feedback on the efficacy of the algorithm they have written (Sullivan & Heffernan, 2016). The iterative nature of the robotic activity, embodied in the troubleshooting cycle (Sullivan, 2011), allows the students to experiment with elements of the programming to broaden and deepen their knowledge of programming and to engage in computational thinking. This is supported as each of the graphical icons in the programming language has variable properties that can be set. As students test their robotic device in the environment, they begin to think about how to more precisely program the movement of the robot within the given environment (e.g., to avoid obstacles). Having the ability to vary the speed of the robot, the angle of a turn, or the distance a robotic device will travel, allows for a more precise program, more complex algorithms, and more practice with computational thinking to solve the robotics problems with algorithms. Therefore, robotics is a very robust activity for stimulating the development of computational thinking.

13.1.3 Collaborative Learning with Robotics: Emergent Roles

When working collaboratively, students take on roles as the group seeks to regulate its work. Group roles are an important element of all collaborative learning, but especially in a Computer-Supported Collaborative Learning (CSCL) environment (Hoadley, 2010), such as robotics. According to Stijbos and De Laat (2010), roles come into being in one of two ways: scripted or emergent. Scripted roles are those that is assigned by a teacher to facilitate the process of collaborative learning, and include precise instructions on how students should interact with one another in the group setting to promote learning. This is contrasted with emergent roles that "emerge spontaneously or are negotiated spontaneously by group members without interference by the teacher or researcher" (p. 496).

Research on scripting collaborative roles for students demonstrates mixed results. While there is a fair amount of support for the efficacy of providing students with scripts for how to interact with one another while working collaboratively (Fischer, Kollar, Mandl, & Haake, 2007; O'Donnell, 1999). There is also research that indicates that scripts lose their efficacy when students are engaged in collaborative learning over longer periods (Rummel & Spada, 2007); and that other approaches, for example, students observing a model of collaborative learning, are more effective than scripts in enabling successful collaborative learning in student groups (Rummel & Spada, 2009). Moreover, providing students scripts for collaborative learning interactions has been criticized as overly directive, depriving students of the opportunity to engage in the type of thinking that will lead to creativity in problem-solving (Dillenbourg, 2002).

Meanwhile, robotics learning environments are multidimensional problem spaces which afford multiple roles that may be taken up in an emergent fashion, including the role of programmer, builder, and 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). Such vying for control can create tension in the group and interfere with opportunities to learn (Sullivan & Wilson, 2015). Therefore, it is not only the functions of the role that are important, but the shifting of roles, and the negotiation of roles that may also affect student learning (Sullivan & Wilson, 2015; Wise, Saghafian, & Padmanabhan, 2012).

In collaborative learning settings, work is meant to be done by the entire group and through these social interactions new knowledge is built (Vygotsky, 1978). However, collaborative groups do not always work well together, hence, these learning interactions may not take place (Dillenbourg, 1999). Successful collaborative group work requires ongoing, well-coordinated group interactions (Barron, 2003). Barron has argued that the accomplishment of this coordination occurs through a complex array of cognitive and social tasks, which may be categorized into three areas of collaborative interaction: shared task alignment, joint attention, and mutuality. Collaborative groups may demonstrate varying degrees of these interactions, indicating both higher and lower levels of coordination. In this way, the level of coordination

www.dbooks.org

will have an impact on the group's learning outcomes. Recent research has shown that students who have adopted emergent roles in group work can successfully achieve higher levels of coordination (Sullivan & Wilson, 2015). Given this, and the fact that emergent roles may better support creative thinking (Dillenbourg, 2002) scripted collaborative learning was avoided. In our study, students were asked to collaborate, they were not assigned specific roles, but they had access to the three primary roles associated with the activity: programmer, builder, and analyst. Here, we examine how the specific emergent roles the students inhabit impacts both their collaborative interactions and their engagement in computational thinking.

This research proceeds from the Vygotsky (1978) perspective, which holds that through direct interaction with the robotics tools and engagement in collaborative group discussions, students will develop knowledge and ideas about computation. Here, we aim, through descriptive and qualitative analysis, to characterize the nature of girls' computational thinking as they "do" robotics. This analysis will serve as a starting point for designing effective curricular and pedagogical scaffolds for supporting girls' development of computational thinking abilities and computer science knowledge.

Our focus is on how emergent roles in the multidimensional problem space of robotics relates to collaboration and to different types of computational thinking. For the purposes of this study, we define three working modes: working individually, working cooperatively (defined as working jointly toward a solution through a divide and conquer approach), and working collaboratively (defined as working jointly toward a solution through discussion and dialogue). This analysis is accomplished through systematic observational methods and sequential analysis which allows for the understanding of behavior as it unfolds over time (Bakerman & Quera, 2011). The goal of this research is to improve our knowledge of how the emergent roles enabled in a robotics learning space are taken up and how they relate to collaborative interactions, engagement in computational thinking and learning outcomes as measured by the difficulty of challenges undertaken. This knowledge will assist teachers and curriculum developers in creating robotics learning settings that best support student learning in computer science. Our specific research questions are presented below.

13.1.4 Research Questions

RQ1. What are the role transitions made by novice programmers in this study?
RQ2. How do different roles in a robotics programming environment relate to different types of collaboration?
RQ3. How do different types of collaboration in a robotics programming environment relate to different types of computational thinking?
RQ4. How does engagement in specific types of computational thinking relate to student learning of robotics as measured by the difficulty of challenges undertaken?

13.2 Methods

This observational case study took place at a 1-day, all-girl introduction to robotics event called "Girls Connect." The students spent 2 hours learning how to design and program the robot (Lego® EV3), they then spent the remainder of the day working on solving robotics challenges. The participants in the workshop included 17 girls, ages 8–13 (M = 11.725) who attended five different schools in New England. Purposeful sampling was used to select students from various backgrounds and geographic areas from the pool of students who volunteered for the event. Students were drawn from schools that were struggling academically; four of the five schools who sent student participants were not meeting state standards for student academic performance. All of the participants were working with robotics for the first time. The "Girls Connect" event was designed to introduce girls to the FIRST LEGO League (FLL)® in order to stimulate their interest in the FLL and robotics. The workshop featured the FLLs 2011 challenge: "Food Factor." This challenge includes 11 missions of varying degrees of difficulty. The students were allowed to select the mission(s) they wished to solve. In the morning of the 1-day workshop, the girl participants did team building exercises, learned a little about programming, learned about the food factor challenge, and built a basic robotic car from blueprints. After lunch, the girls devoted themselves to solving the robotics missions that were laid out on the food factor challenge board. In our study, the aggregate of missions attempted across all groups was seven. In other words, collectively the six groups of girls attempted to solve seven of the missions. The students were divided into six teams (five teams of 3 and one team of 2); girls from the same schools were on the same team. Each of the six teams were given color-coded t-shirts to wear for the day. For example, one team wore green t-shirts, one yellow, etc. The t-shirts bore the "Girls Connect" logo and were presented both as commemorative gifts to participating girls and also to function as an aid to the researchers in keeping track of who was on which team as the girls roamed about the room.

The data analyzed in this study are drawn from two of the teams who participated. For purposes of analysis, we selected a team that appeared to demonstrate higher levels of coordination (worked collaboratively), and a team that appeared to demonstrate lower levels of coordination (worked in parallel), based on viewing of the videotapes. The purpose of making this selection is to aid analysis of the relationship of role to collaboration and computational thinking. By selecting a more collaborative team and a less collaborative team, we are better able to delineate the relationship of our constructs of interest. The data consists of the afternoon problem-solving activities and discussions, and includes 3 h of video for each of two teams. Pseudonyms of Anna, Becky, and Cindy were used to identify each of the girls on the light blue team, and Janet, Kaylee, and Lisa on the dark blue team. Consent was obtained from the parents of the participants and assent from the participants themselves.

We collected audio and video data at the 1-day event. Each of the girl participants in the study wore a wireless microphone. Each group of girls had their own worktable, a LEGO Mindstorms EV3 robotics kit, and a laptop computer. Two challenge arenas

www.dbooks.org

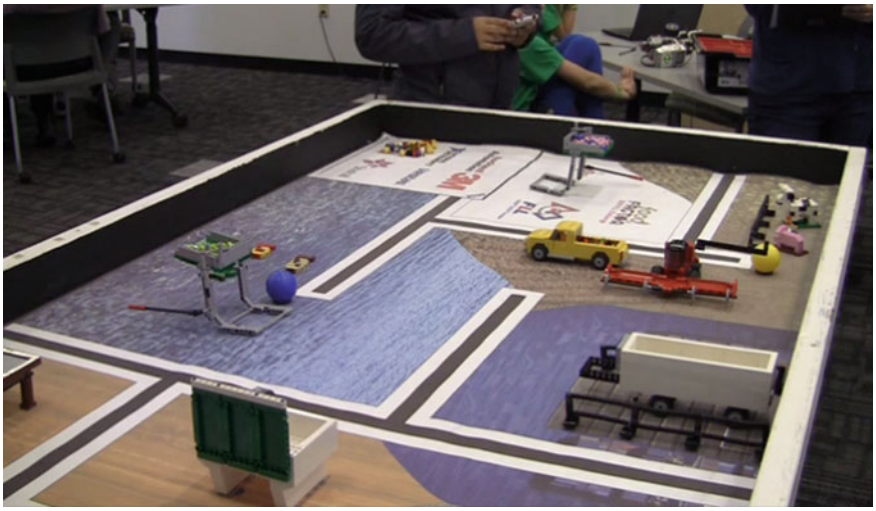


Fig. 13.2 FLL challenge arena

were set up in the room so that the girls could test their solutions (see Fig. 13.2). A stationary video camera was mounted at each group worktable to capture the building and programming of the robots. Two additional cameras were focused on each of the challenge arenas. From these data, we created a video and audio recording of each group's activity and discussion for the day as they moved between their individual worktables and the challenge arenas. See Fig. 13.3 for an illustration of the room and camera setup. A professional transcriptionist transcribed all group talk. We also ran a screen capture program on each groups' laptop. In this way, we collected all of the robotics programming activity engaged in by each group. This data includes the final robotics program(s) created by each group. The data analysis unfolded over four distinct phases including: (1) behavior analysis of the roles students enacted and collaborative interactions observed; (2) discourse analysis of student talk related to computational thinking; (3) descriptive statistics related to the observed roles and collaborations; and (4) learning outcomes analysis based on a challenge difficulty score, role enactment and observed collaboration.

13.2.1 Phase I—Behavior Analysis: Roles and Collaboration

The first phase was to record onset and offset times of certain behaviors as observed on the videos. The role of the student and the type of collaboration were coded as continuous and mutually exclusive. This means that all 3 h of video data were coded (for each student) and no overlapping of the codes occurs. The unit of analysis for this phase of data coding was "shift of focus." In other words, when students

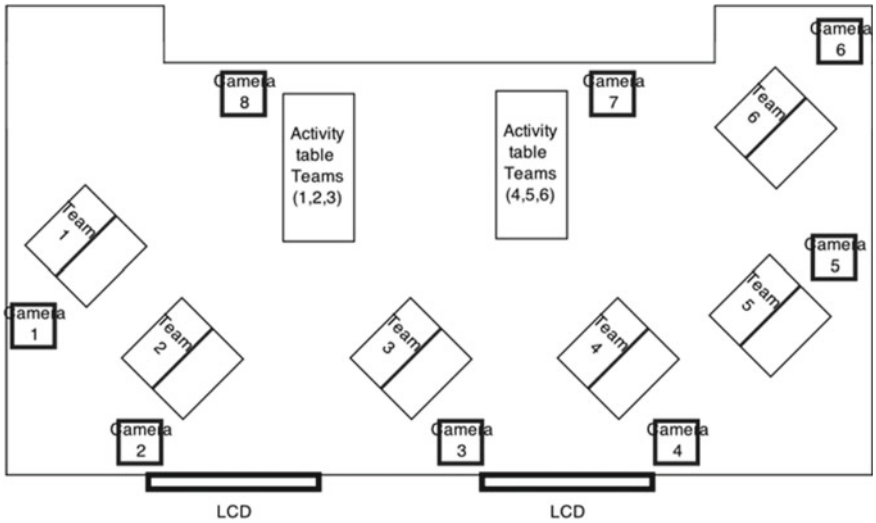


Fig. 13.3 Research video camera setup

switched their attention from one activity to another activity, we initiated a new code (Chi, 1997). The codes for the emergent roles were derived from attending the event and watching the video recording numerous times and included Programmer, Tester, Builder, Analyst, and Other.

The codes for collaboration are based on Forman and Cazden's (1985) codes for participation in groups as markers of coordination, discussed above. They included Parallel (little to no focus on the group), Cooperative (Working together, focused on own results), Collaborative (Working together and sharing ideas), and External (Focused on something outside of the group).

Inter-rater reliability was assessed by training a second coder. A timed-event alignment kappa (Bakerman & Quera, 2011) was used since it allows for tolerances between onset and offset times. Results for inter-rater reliability for the role were $\kappa = 0.83$ and for collaboration were $\kappa = 0.92$. In order to achieve this kappa, the coders reviewed disagreements in their coding and resolved conflicts.

13.2.2 Phase II—Discourse Analysis: Computational Thinking

This phase focused on the discourse related to the robotics activity. First, we segmented the 3-hour conversations into Individual Troubleshooting Cycles (TSC) for each group. As discussed above, the troubleshooting cycle is iterative and consists of building the device, writing a program, testing the program, diagnosing and debug-

www.dbooks.org

Table 13.1 Computational thinking coding system

Major level	Subcategory	Code	Description
Analysis		A	Any planning of a mission solution, including developing ideas or approaches for mission solutions that are specific to the physical space the robot must negotiate, like the initial placement of the robot
Algorithmic thinking	Variable	ATV	When quantity is discussed in determining how far something should turn or move. Often will be thought of as degrees, seconds, or revolutions
	Operation	ATO	When discussing the type of programming block that should be used. Most programs in the data set will only use two blocks, turn or move. Often will be used as turn left, turn right, move forward, move backward
Debugging	Observation	DO	When students describe the movement of the robot during a test, or discuss the outcome of the test in terms of the robots interaction with the pieces on the gaming arena
Designing		D	When working on building additions to the robot. Or any discussion about the design of the robot including physical changes that need to be made

ging the program, and revising the program, or revising the design of the built device (Sullivan, 2011). The transcripts were then coded using a priori codes we developed based on the work of Wing (2006) and Barr and Stephenson (2011). These codes have been synthesized to be relevant for the activities and type of behavior expected and observed for novice programmers in a robotics environment. Table 13.1 presents the coding scheme.

Two undergraduate students were trained in the use of the coding scheme and all utterances from both groups over the course of the challenge period (3 h in the afternoon) were coded. This chapter is concerned with student's computational thinking. Therefore, all comments that were not related to computational thinking were coded as other. Inter-rater reliability was calculated utilizing Krippendorff's alpha (Krippendorff, 2004). Results for inter-rater reliability for the discourse were $\alpha = 0.901$, which indicate that inter-rater reliability for this study was high.

13.2.3 Phase III—Descriptive Statistics: Roles

The first step in phase III was to calculate descriptive statistics to summarize the coded observations of student behavior. The total time and relative duration were

calculated for role and collaboration. Then, once we had segmented, coded, and scored the discourse data, we tabulated the instances of the CT codes in each TSC for each group. As a result, we created one table of data for each group. The rows of the tables were the eight CT codes described above. The columns listed out all the TSCs of the group. In each cell of the tables, we put the number of instances we observed for each CT code at each TSC. Using those tables as input, we conducted one-way ANOVAs to compare the means of the number of instances of the CT codes across all the TSCs. For each group, the null hypothesis we tested was whether the discussions underlying the codes occurred at the same frequency on average across the TSCs. To visually inspect the frequency differences among the groups, we then created boxplots for the instance counts of each CT code for each group. We used those plots to visually assist the comparison of how frequent the discussions were with each group.

The second step in phase III was to calculate the joint probabilities of roles and type of collaboration. A joint probability is the probability that an event will occur given another event. This is also called Lag(0) (Bakerman & Quera, 2011) analysis since the calculation describes the co-occurrence of events in the same time frame. Transitional probabilities, or Lag(1) (Bakerman & Quera, 2011) were also calculated. These show the probability of one event following another event.

13.2.4 Phase IV—Difficulty Score Calculation: Learning Outcomes

The last phase of the analysis included examining the difficulty of the programs that each of the two groups attempted as a proxy for the level of learning each group had achieved. We reasoned that the more confident each of the teams felt over the course of the afternoon, the more likely they would be to select more challenging tasks to accomplish on the game board. We developed a rubric to score these programs. This rubric is based on the difficulty of the mission tasks presented on the FLL challenge arena. There were 11 missions on this particular arena. However, the students in our study only attempted seven of those missions. The missions (also referred to as challenges in this chapter), consisted of moving the robot to different sections of the game board in order to act upon materials placed on the game board. In the Food Factor challenge, students learn about bacteria, overfishing, and other aspects of food safety and food production. The missions relate to these ideas and consist of moving materials, maneuvering around materials, and collecting materials.

The first author on this chapter is a computer science professor and an expert in the field. The first author calculated the difficulty of each of the missions by creating an optimal solution for each task. Due to the fact that students were novices, with no prior programming or robotics experience, the challenge solutions were simple and relied primarily on moving the robot forward and backward and turning the robot. These elements, forward/backward and turning, could be programmed by

www.dbooks.org

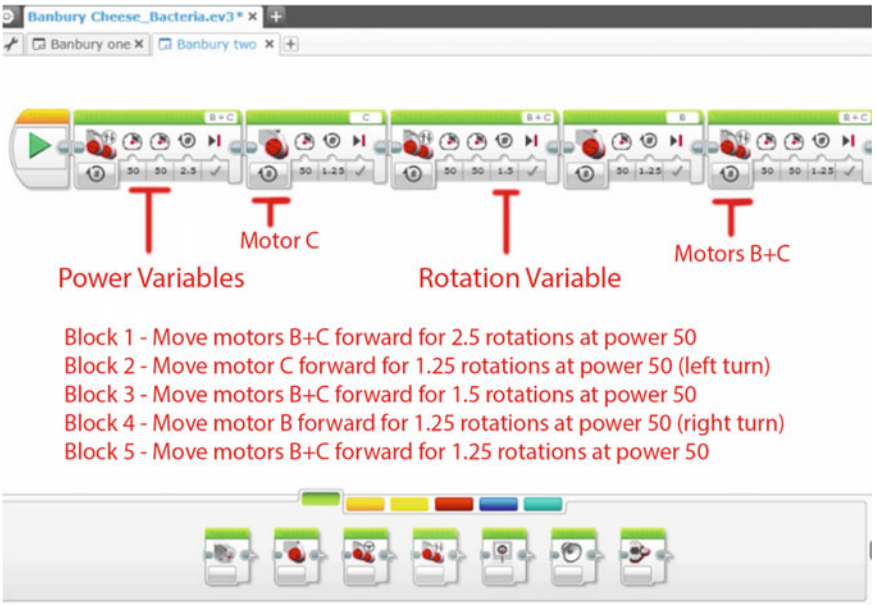


Fig. 13.4 Annotated student program

adjusting power to the motors, and/or setting the number of rotations of the attached wheels on the robotic device. We have annotated an example of a student program, drawn from the data, to highlight the nature of student programs (see Fig. 13.4). The annotations highlight the meaning of the icons in each block, as well as provide an English language translation of the meaning of each block as arranged in the program. After we had developed optimal solutions, we then compared the relative difficulty of each solution based on three variables including: (1) the number of programmed moves forward/backward needed, (2) the number of turns needed, and (3) the distance needed to be crossed on the board to attempt that mission (based on # of wheel rotations). We reasoned that attempting a mission that was far away required more precision in navigating the board to arrive at the mission. Therefore, for the far missions, we doubled the sum of turns + moves to account for the added difficulty presented by the distance. Student programs were then scored based on the difficulty of the mission attempted. Table 13.2 presents the scoring rubric by the mission for the seven missions attempted by participants in the study.

In the final step, we focused on examining the relationship between the frequency of the discussions that are the most relevant to computational thinking and the overall scores for the groups. Those discussions were those that received the codes A, ATO, ATV, D, and DO. For each group, we computed the frequency means of the five codes across all the TSCs. Then, we plotted those means along with the overall program scores in the same scatter plot for each group. To guide the interpretation of the relationship, we used different colors and symbols for each group and each CT code.

Table 13.2 Final program scoring rubric by mission

Mission name	# of turns req. (a)	# of moves req. (b)	Distance traveled (near or far)	Score = (a + b) × 2
Blue Ball	0	2	Near	2
Green Bacteria	0	2	Near	2
Yellow Ball	2	4	Near	6
Harvester	4	6	Near	10
Pink Bacteria	4	6	Near	10
Pizza	2	3	Far	10
Red Bacteria	3	4	Far	14

13.3 Results

13.3.1 Role Transitions

In answer to research question #1—What are the role transitions made by novice programmers in this study?, we present the results of our behavior analysis of transitional probabilities; this gives us an overall view of how the students organized themselves. Transitional probabilities in this analysis are indicated by the arrow (direction) and percentage (probability). The transitional probabilities in this analysis specify the probability of the next role taken up by the student given their current role. For example, in Fig. 13.4, there is a 76% probability that for Anna, the role of succeeding programmer would be tester. The light blue Team (Fig. 13.5) took a divide and conquer approach to solving the programming challenges. This is evidenced by the lack of continuity shown in their paths between roles. Each student has a unique path, indicating variation in student activity. Anna has taken on a primary role of programmer, Becky the role of builder, which left Cindy with no primary individual role, but she did take part in the collaborative role of analyst. The dark blue team (Fig. 13.6) have taken up a more collaborative strategy that is highlighted by the similarity of their role transitions. In other words, in visually examining the role transitions for the dark blue team, there is less variation, indicating that the students were jointly involved in sharing the roles and collaborating.

13.3.2 Collaboration

In answer to research question #2—How do different roles in a robotics programming environment relate to different types of collaboration?, we first determined the amount of time spent in the types of collaboration (parallel or collaboration). No episodes of cooperation were observed. Students were either working together with

www.dbooks.org

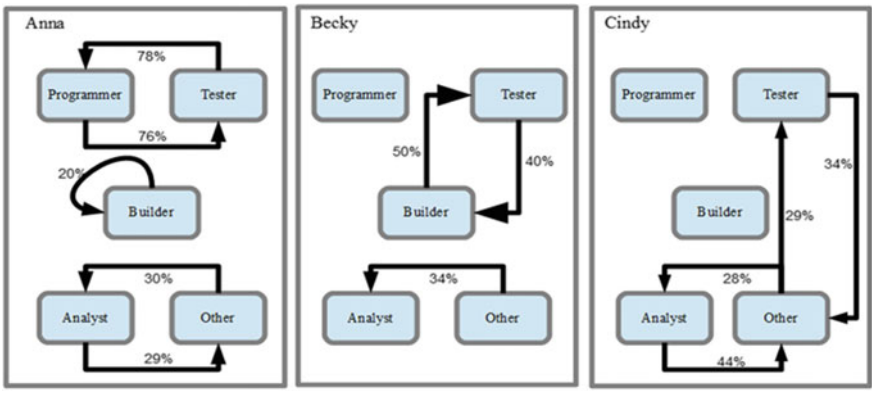


Fig. 13.5 Roles transitions by light blue team

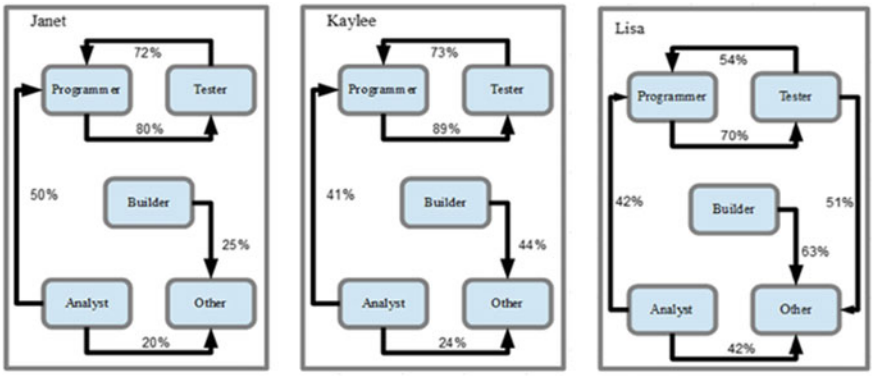


Fig. 13.6 Roles transitions by dark blue team

one set of materials (collaboration) or working alone (individual). We then calculated the duration and relative duration of collaboration for each of the students, presented in Figs. 13.7 and 13.8.

The data indicate that the students on the light blue team spent at least half of the time working collaboratively. However, the students on the dark blue team were jointly attentive at least 80% of the time. The understanding of collaboration is further explored by examining the joint probabilities between the role and the type of collaboration. Notable outcomes from this analysis show that when a team divides up the roles and a member is involved in a primary role, they perform that role in a non-collaborative way. For example, when Anna assumed the role of programmer, there was a 72% chance that she would be working alone and a 27% chance that she would be working collaboratively. This is contrasted to the dark blue team where no one took on a primary role, but each took turns sharing the roles during the day. For example, when Janet assumed the role of programmer, there was an 84% probability

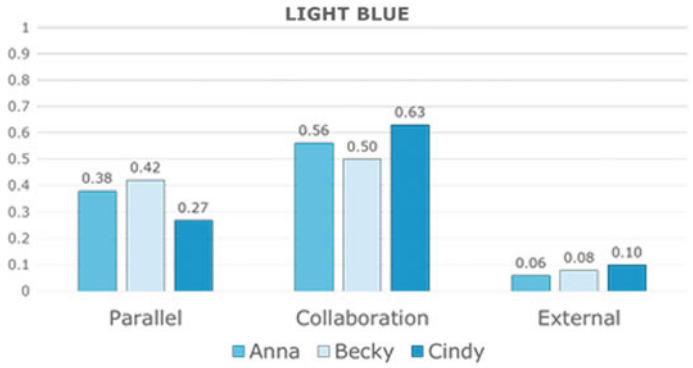


Fig. 13.7 Collaboration light blue

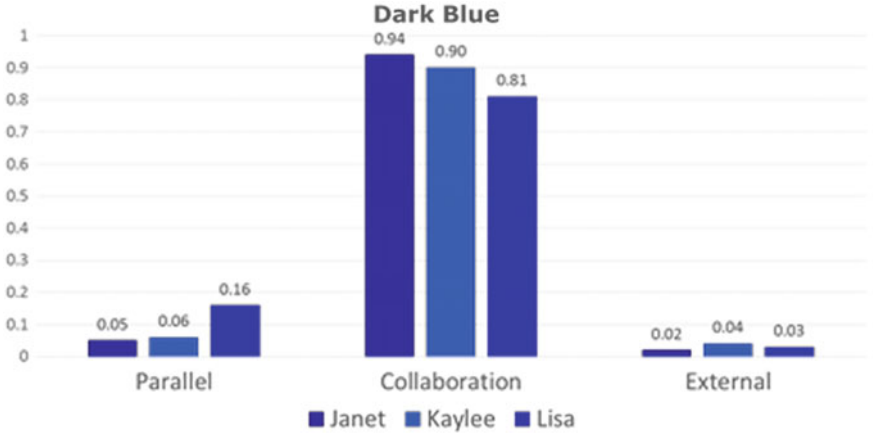


Fig. 13.8 Collaboration dark blue

that she would be working collaboratively. Overall, and for both groups, the two roles of tester and analyst do stand out as having a higher probability of being collaborative for all students.

13.3.3 Computational Thinking

In answer to research question #3—How do different types of collaboration in a robotics programming environment relate to different types of computational thinking?, we first present the frequency and relative frequency of the different computational thinking codes for each student in each group, across the afternoon (Table 13.3).

www.dbooks.org

Table 13.3 Frequency and relative frequency of CT codes per student

	Analysis		Alg. thinking — variable		Alg. thinking— operation		Design		Debugging	
	Freq	Pct (%)	Freq	Pct (%)	Freq	Pct (%)	Freq	Pct (%)	Freq	Pct (%)
Anna	66	34	13	7	58	30	36	18	23	12
Becky	124	48	1	0	20	8	105	40	11	4
Cindy	81	29	5	2	34	12	123	45	34	12
Janet	97	26	133	35	62	16	31	8	53	14
Kaylee	61	19	105	33	70	22	36	11	50	16
Lisa	46	23	60	30	22	11	45	21	32	16

The idea of the variable is a foundational concept in computer programming (Soloway, Ehrlich, Bonar, & Greenspan, 1982). However, the total frequency of discourse concerning the variable is 19 for the light blue team as compared to 298 for the dark blue team. This pattern was set early in the process. When Anna was learning to use the software, she would bench test the robot. Through this process, she was learning how the values of the variables affected the movement of the robot. However, she did this in complete silence. The dark blue team completed the same process but did this collaboratively.

To shed further light on students' computational thinking, we present a short vignette of the dark blue team's discussion as they discuss changes to their algorithm (Table 13.4).

Next, we present boxplots (Fig. 13.9) for each of the two groups that depict the group's CT coded discourse frequency per troubleshooting cycle. The bold line represents the median and the size of the box the variance. The light blue team engaged in 92 complete troubleshooting cycles over the afternoon, the dark blue team completed 78 troubleshooting cycles.

Analysis of the box plots revealed differences across the two groups as regards CT talk. While the light blue team spent a lot of time discussing the physical design of the robot, the dark blue group spent more time discussing the algorithms needed to program the robot, and in particular, variable aspects of the algorithm. 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 values of the variables. The dark blue group discusses the variables more, and the value of the variables more. It may be reasonable to argue that they are learning more about the robotics environment by discussing the variables in the context of the gaming environment.

In answer to research question #4—How does engagement in specific types of computational thinking relate to student learning of robotics as measured by the difficulty of challenges undertaken?, and to further explore the impact of the differences in group CT-based conversations, we developed an analysis of the difficulty of the challenges each group sought to solve over the course of the afternoon. Table 13.5 presents missions attempted per group.

Table 13.4 Dark blue group CT discussion example

Speaker: Utterance	CT Code
J: That distance just needs to be a little longer. Yeah, we got it there.	DO
K: This one a little longer and then we add the thing. Because it goes waaa.	DO
J: We can have it go diagonal and then diagonal and then straighten itself out, but I…	A
K: Okay so the last turn is a little longer right?	ATO
J: Yeah should be like two.	ATV
K: So two point five?…wait I mean not two point five, two, one, one point…	ATV
J: Wait, was it the turn or the distance, oh that's the, oh that's moving it.	ATO
K: Distance was good the turn I think was a little short because…	ATO
J: No I think it, the distance should be longer when it goes.	ATO
K: Yeah, yeah that's right, sorry, so one point five do you think?	ATV
J: Um lets make it two,	ATV
K: Okay	O
J: Lets make it two.	ATV
K: Okay. Right, now…	O

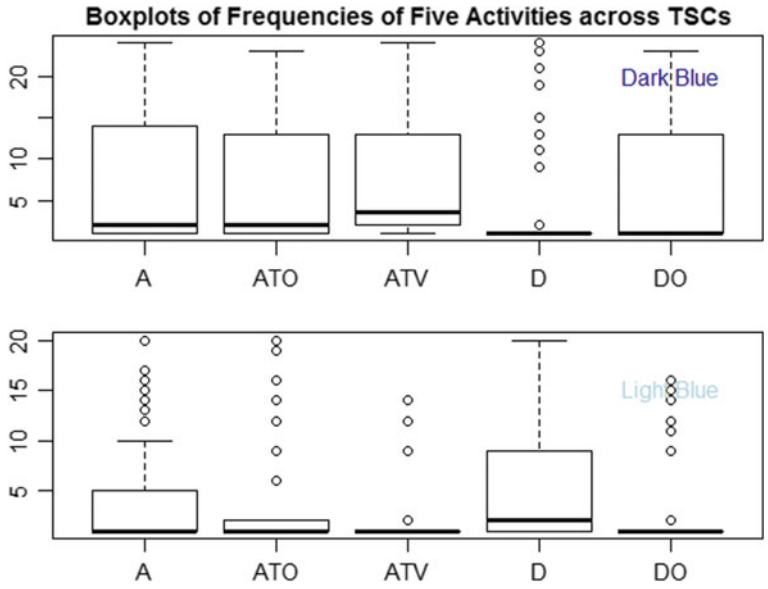


Fig. 13.9 Boxplots of for the two teams

www.dbooks.org

Table 13.5 Missions attempted per group

	Score	Dark blue team	Light blue team
Blue Ball (pollution reversal)	2		
Green Bacteria (disinfect)	2		x
Yellow Ball (poll. reversal)	6		
Harvester (Corn)	10		x
Pink Bacteria (disinfect)	10	x	
Pizza and Ice Cream	10		
Red Bacteria (disinfect)	14	x	

As can be seen from this table, the dark blue team attempted two missions that were rated as more difficult, whereas the light blue team attempted two missions that were relatively simpler. We take this as evidence of the knowledge the dark blue team built, working collaboratively over the course of the day. With a greater level of knowledge shared among the group, the group chose to attempt more challenging missions.

The final analysis consists of a scatter plot, which visualizes the relationship of student scores on their final programs (y-axis) and the amount and type of CT talk each group engaged in during each TSC (x-axis). The scatter plot is presented in Fig. 13.10.

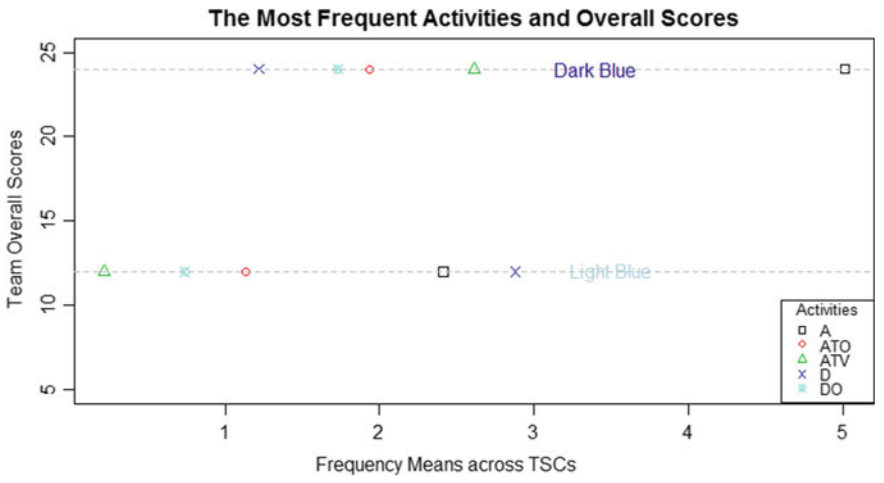


Fig. 13.10 Scatter plots for the most frequent CT codes and program scores

The scatter plot indicates that the dark blue team, who had the highest programming score, also had the highest mean for all of the computational thinking codes, except for the design code. This team spent a lot of time discussing the algorithm they were working to develop. Meanwhile, the light blue team spent a lot of time discussing the design of the robot. The design relates to the physical elements.

13.4 Discussion

This study investigated the relationship of roles to collaboration and computational thinking in the multidimensional problem space of robotics. Our analysis indicates that the emergent roles commonly adopted in robotics learning environments, emerged in this study. Moreover, the roles afforded by the environment: (a) influenced the type of discourse that was used to discuss the activity and (b) affected the common understanding of the different systems in a robotics environment. Indeed, these emergent roles played a part in the level of collaboration that occurred within the group. However, it is not the nature of the roles that mattered, but rather how the roles were negotiated within the groups themselves that influenced collaboration and student learning outcomes.

In the light blue group, the roles were adopted early on and adhered to, roles were not shared among the group members, and therefore, collaboration was hindered. Since Becky and Cindy had little chance to program, they were not able to engage in much algorithmic thinking at either the variable or operational level. For example, together, Becky and Cindy made 54 algorithmic thinking operations comments overall, whereas, Anna, the main programmer, made 58 such comments overall. Meanwhile, in the dark blue group, the three girls shared the main roles, each taking a turn as programmer, builder, tester, and analyst. This led to a common language and a common understanding that allowed for collaboration and, arguably, greater understanding of programming and robotics, as evidenced by the group's selection of more challenging tasks. Importantly, as noted above, this group also worked with the variable aspects of the programming icons. Through engaging with the variables of power and rotations, the group was able to develop more control over the movement of their robotic device and feel confident to attempt the more challenging tasks. It is probable that the joint engagement in programming supported the group's deeper engagement with the activity. Also, as shown in the scatter plot, the dark blue group spent more time discussing abstract elements of the activity (the algorithms), whereas the light blue group spent more time discussing the physical design of the robot, which is, arguably, less abstract. It is likely that the collaborative discussions had in the dark blue group supported the group's computational thinking.

Moreover, we note that the greatest probability for collaboration for both groups occurred when students were enacting the tester and analyst roles. This stands to reason, testing the robot is the activity where all group members can equally participate through observation and discussion. And the role of the analyst was enacted subsequently to testing. In other words, students would test the robot, jointly observe

www.dbooks.org

the execution of the program, and then jointly enact the analyst role in discussing the executed program. Therefore, this result is not surprising, but expected. It is the external and manipulative nature of the robotic activity that supports robust cognitive engagement (Sullivan & Heffernan, 2016).

Collaboration in groups is difficult, but due to the expense of educational robotics, many teachers are forced to create groups. In order to increase the probability that groups will collaborate, scaffolding is necessary (Winne, Hadwin, & Perry, 2013). However, in order for scaffolding to be successful, we must understand the conditions for interactions and the interactions that are indicative of learning (Hoadley, 2010). This research illustrates the importance of scaffolding role rotation in robotics activity. While some may be tempted to interpret these findings as support for a scripted approach, we resist such an interpretation. Rather than providing students with scripts for how to interact, we believe that a simple enforcement of the idea of role rotation could support a group in building enough intersubjective understanding to develop high levels of coordination in the group. Robotics is a very creative activity (Sullivan, 2017). Groups should be supported to share roles, but also given freedom to explore problem solutions in an authentic way. Perhaps it is time to think about a middle ground between completely emergent roles and highly scripted roles. We would advocate for a middle ground in the case of robotics and other highly creative activities. For example, one way to structure this would be to have students intentionally switch roles after a certain amount of time. In this way, the roles would rotate among students and would support collaboration as students would gain knowledge and share it with one another. Another idea would be to introduce a wide-screen multi-touch display, in place of a laptop, for programming. It is possible that such a technology would create more access to the programming tool and allow for greater conversations among students.

Future research should focus on providing varying levels of scaffolded support to students working with robotics. Is it enough to enforce a role rotation within groups, or will students need more support to engage in collaboration? We believe that collaboration within a group is influenced by a multitude of factors. However, we also believe that setting certain conditions for participation may enable higher levels of coordination, for example enforcing role rotation. Future research studies could create conditions that include role rotation and those that support emergent roles. Engagement in collaboration and the development of computational thinking could then be measured in each of the conditions. Moreover, environments where students are developing computational thinking abilities often have a material technological component. Future research should examine how scaffolding student participation by prescribing shared control of the material artifacts affects student discussions and the ability to coordinate the work of the group. For example, as mentioned above, we think that a wide screen multi-touch display may shift collaborative interactions by virtue of allowing students to more closely observe programming activities. In this scenario, students who are not directly manipulating the software could still meaningfully participate through developing a deeper understanding of the programming blocks, thereby improving their ability to reason about the program and recommend possible changes to programs by virtue of close observation. Again, such research

may include conditions (widescreen, multi-touch display, vs. laptop). Such studies will improve our ability to provide robust collaborative robotics learning environments for students.

References

Ali, S. R., Brown, S. D., & Loh, Y. (2017). Project HOPE: Evaluation of health science career education programming for rural latino and european american youth. *The Career Development Quarterly, 65,* 57–71.

Bakerman, R., & Quera, V. (2011). *Sequential analysis and observational methods for the behavioral sciences*. New York: Cambridge University Press.

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.

Barron, B. (2003). When smart groups fail. *Journal of the Learning Sciences, 12*(3), 307–359.

Chi, M. T. H. (1997). Quantifying qualitative analysis of verbal data: A practical guide. *Journal of the Learning Sciences, 6*(3), 271–315.

CSTA. (2016). *Computational Thinking.* Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf.

Dillenbourg, P. (1999). *Collaborative learning: Cognitive and computational approaches*. New York, NY: Elsevier Science.

Dillenbourg, P. (2002). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. In P. A. Kirschner (Ed.), *Three worlds of CSCL. Can we support CSCL* (pp. 61–91). Heerlen, NL: Open Universiteit Nederland.

Fischer, F., Kollar, I., Mandl, H., & Haake, J. (Eds.). (2007). *Scripting computer-supported communication of knowledge. Cognitive, computational, and educational perspectives*. New York, NY: Springer.

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.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42*(38), 38–43. https://doi.org/10.3102/0013189X12463051.

Hoadley, C. (2010). Roles, design, and the nature of CSCL. *Computers in Human Behavior, 26*(4), 551–555.

Ji, P. Y., Lapan, R., & Tate, K. (2004). Vocational interests and career efficacy expectations in relation to occupational sex-typing beliefs for eighth grade students [Electronic version]. *Journal of Career Development, 31*(2), 143–154.

Jones, A., & Issroff, K. (2005). Learning technologies: Affective and social issues in computer-supported collaborative learning. *Computers & Education, 44*(4), 395–408.

Krippendorff, K. (2004). *Content analysis, and introduction to its methodology* (2nd ed.). Thousand Oaks, CA: Sage Publications.

Lee, I., Martin, F. L., Denner, J., Coulter, R., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37. https://doi.org/10.1145/1929887.1929902.

National Instruments (2014). *LabVIEW object-oriented programming faq*. Retrieved online at http://www.ni.com/white-paper/3574/3n/.

National Science Foundation (2016). *Building a foundation for CS for all.* https://www.nsf.gov/news/news_summ.jsp?cntn_id=137529.

O'Donnell, A. M. (1999). Structuring dyadic interaction through scripted cooperation. In A. M. O'Donnell & A. King (Eds.), *Cognitive perspectives on peer learning* (pp. 179–196). Mahwah, NJ: Erlbaum.

www.dbooks.org

Rieber, L. (2005). Multimedia learning in games, simulations, and microworlds. In R. E. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (pp. 549–567). New York: Cambridge University Press.

Rummel, N., & Spada, H. (2007). Can people learn computer-mediated collaboration by following a script? In F. Fischer, I. Kollar, H. Mandl, & J. Haake (Eds.), *Scripting computer-supported communication of knowledge. Cognitive, computational, and educational perspectives* (pp. 47–63). New York, NY: Springer.

Rummel, N., & Spada, H. (2009). Learning to collaborate while being scripted or by observing a model. *Computer-Supported Collaborative Learning, 4,* 69–92. https://doi.org/10.1007/s11412-008-9054-4.

Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. (1982). What do novices know about programming. *Directions in Human-Computer Interaction,* 87–122.

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, 26*(4), 495–505.

Sullivan, F. R. (2008). Robotics and science literacy: Thinking skills, science process skills, and systems understanding. *Journal of Research in Science Teaching, 45*(3), 373–394.

Sullivan, F. R. (2011). Serious and playful inquiry: Epistemological aspects of collaborative creativity. *Journal of Educational Technology and Society, 14*(1), 55–65.

Sullivan, F. R. (2017). The creative nature of robotics activity: Design and problem solving. In M. S. Khine (Ed.), *Robotics in STEM education: Redesigning the learning experience* (pp. 213–230). Springer: The Netherlands.

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. https://doi.org/10.1080/15391523.2016.1146563.

Sullivan, F. R., & Wilson, N. C. (2015). Playful talk: Negotiating opportunities to learn in collaborative groups. *Journal of the Learning Sciences, 24*(1), 5–52. https://doi.org/10.1080/10508406.2013.839945.

United States Bureau of Labor Statistics (2017). Employment of women on non-farm payrolls by industry sector, seasonally adjusted. Retrieved from https://www.bls.gov/news.release/empsit.t21.htm.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes.* (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.) Cambridge, MA: Harvard University Press.

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

Winne, P. H., Hadwin, A. F., & Perry, N. E. (2013). Metacognition and computer-supported collaborative learning. In C. Hmelo-Silver, A. O'Donnell, C. Chag, & C. Chinn (Eds.), *International handbook of collaborative learning* (pp. 462–479). New York, NY: Taylor & Francis.

Wise, A. F., Saghafian, M., & Padmanabhan, P. (2012). Towards more precise design guidance: Specifying and testing functions of assigned student roles in online discussions. *Educational Technology Research and Development, 60*(1), 55–82.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 14
Video Games: A Potential Vehicle for Teaching Computational Thinking



Sue Inn Ch'ng, Yeh Ching Low, Yun Li Lee, Wai Chong Chia and Lee Seng Yeong

Abstract Previous studies in computer science education show that game playing is negatively correlated with success in introductory programming classes in (Wilson, Shrock ACM SIGCSE Bulletin, vol 33, pp 184–188, 2001). However, informally, we observed that students who have previous gaming experience take to programming tasks easier than those without gaming experience. There have also been recent studies that show that playing strategic video games can improve problem-solving skills which is an essential skill in program design. This chapter presents the findings of our study to identify if a correlation between previous gaming experience (game playing) and individual computational thinking (CT) skills exists. To achieve this, a survey was administered on undergraduate students undertaking an introductory computing course to collect data on their gaming history and an individual assignment on Scratch. Each project was subsequently analysed to determine the level of mastery of core CT skills. Cochran–Armitage test of trend was then executed on each CT skill category with respect to the coded gaming experience. The results obtained during our analysis shows a correlation between gaming experience and specific categories of the CT skills domain particularly in the area of abstraction and problem-solving and user interactivity. The outcome of our study should be beneficial as ways to leverage on students' gaming experience in the classroom will also be discussed.

Keywords Computational thinking · Game-based learning · Games and learning · Scratch assignment · Introductory programming

S. I. Ch'ng (✉) · Y. C. Low · Y. L. Lee · W. C. Chia · L. S. Yeong
Sunway University, Subang Jaya, Malaysia
e-mail: sueinnc@sunway.edu.my

Y. C. Low
e-mail: yehchingl@sunway.edu.my

Y. L. Lee
e-mail: yunlil@sunway.edu.my

W. C. Chia
e-mail: waichongc@sunway.edu.my

L. S. Yeong
e-mail: leesengy@sunway.edu.my

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_14

14.1 Introduction

Computational thinking has been touted as a twenty-first century skill that is as important as reading, writing and arithmetic (Wing, 2006). The term computational thinking was initially coined by Seymour Papert in his book Mindstorms (p. 182) (Papert, 1980) and further elaborated in (Papert, 1996; Barba, 2016). It was only in the communication by Wing (2006) did the term became popularized. In this communication, Wing described computational thinking as 'solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science'. Since then, different researchers and technological groups have suggested that computational thinking involves a number of subskills; with each entity adding their own interpretation as to the key skills that encompass computational thinking. Many educational programmes are then devised around these skill definitions to introduce computational thinking to children. However, many of these programs use programming tools and environments to expose students to computational thinking bringing about the misconception that computational thinking is equated to computer science and subsequently equivalent to 'programming'. As highlighted in (Fletcher & Lu, 2009), teaching computational thinking is not as simple as repackaging CS1, or CS0, and teaching it at an earlier stage. It is also acknowledged that without problem analysis skills, students who are proficient at programming languages would fail to create their own solutions (Koulouri, Lauria, & Macredie, 2015). However, problem-solving skills is not something that can be developed by every student over a short period of time particularly a semester long class. It requires students to experience and reflect on the consequences of their actions. This takes time, practice and effort that sometimes a semester-long course does not permit.

Digital games have been around since the creation of computers and long before computational thinking was popularized and labelled as an important skill. The advent of mobile computing platforms allows digital games to be easily obtained via the internet and gaming to be done anywhere and anytime. Video games which were once confiscated in classrooms are now being adopted by educators as a key teaching tool (Shreve, 2005). This comes as no surprise because digital games contain interactive, engaging and immersive elements that have educational affordances (Frazer, Argles, & Wills, 2008; Gee, 2005). According to (Klopfer, Osterweil, & Salen, 2009), the use of games in formal education can take the form of two approaches: (a) adoption of commercial, off-the-shelf (COTS) games or (b) the application of games in the traditional classroom setting. For the field of CT education, works reported for the former approach are rare whereas the latter approach can take the form of game design assignments or use of serious games. Game design assignments are assignments whereby students are given the task to design and create games to demonstrate the application of learnt technical concepts (Basawapatna, Koh, & Repenning, 2010; Leutenegger & Edgington, 2007; Monroy-Hernández & Resnick, 2008). Computer science (CS) educators have also explored the use of specially created games, also known as serious games, incorporated into traditional lesson plans so that students

www.dbooks.org

learn technical concepts through gameplay (Kazimoglu, Kiernan, Bacon, & MacKinnon, 2012; Liu, Cheng, & Huang, 2011; Muratet, Torguet, Jessel, & Viallet, 2009). Despite studies (Becker, 2001; Kazimoglu, Kiernan, Bacon, & MacKinnon, 2012; Liu, Cheng, & Huang, 2011) reporting an improvement in student engagement and motivation towards the CS content, these studies do not investigate the adoption rate of these games as leisure activities at the end of the course or the effects of extended usage of the designed serious games on students' problem-solving and/or programming skills over time.

On the other hand, children and adults learn best by playing. The work by (Ch'ng, Lee, Chia, & Yeong, 2017) delineates the gameplay elements possessed by popular COTS games for different game genres that support key skills in computational thinking. However, the study did not determine if there is indeed a correlation between playing video games and the mastery of key computational thinking skills. In this chapter, we therefore present our research design and findings to answer the research question on whether past gaming experience does influence specific computational thinking skills. If so, video games can be used as a vehicle to train students to think logically in a fun environment as an alternative to the forceful use of serious games or soldiering through the learning of a programming languages to teach students computational thinking.

14.2 Computational Thinking Skills

The main idea behind the computational thinking movement is that knowledge and skills derived from the field of computer science has far-reaching applications that can be beneficial to other fields too. However, since its conception, there have been different definitions about the skills that encompass computational thinking skills. Some of these definitions can be tightly coupled to programming while others are more loosely defined and general. For example, the CT skills listed by Moreno-León, Robles, & Román-González (2015) is more closely related to programming while the definitions provided by (Lee, Mauriello, Ahn, & Bederson, 2014) is general in nature with little reference made to programming. Barr and Stephenson (Barr & Stephenson, 2011) provided examples on how the nine core[1] CT concepts and capabilities may be embedded in different discipline activities. Table 14.1 lists the different definitions of CT skills by different parties. A look at these definitions shows a repetition and overlap in some of the skills defined such as abstraction, algorithm design and problem decomposition.

It is a common practice for researchers (Berland & Lee, 2012; Kazimoglu, Kiernan, Bacon, & MacKinnon, 2012) in the field of CT education to usually formulate their own definition of CT skills by rationalizing from literature and existing defini-

[1]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 14.1 Table of comparison listing the difference skills encompassing CT defined by different parties

Organization/Researchers	CT skills	Definitions
Google for education (Google, 2018)	Abstraction	Identifying and extracting relevant information to define main idea(s)
	Algorithm design	Creating an ordered series of instructions for solving similar problems or for doing a task
	Automation	Having computers or machines do repetitive tasks
	Data collection	Gathering information
	Data analysis	Making sense of data by finding patterns or developing insights
	Data representation	Depicting and organizing data in appropriate graphs, charts, words or images
	Decomposition	Breaking down data, processes or problems into smaller, manageable chunks
	Parallelization	Simultaneously processing or smaller tasks from a larger task to more efficiently reach a common goal
	Pattern generalization	Creating models, rules, principles, or theories of observed patterns to test predicted outcomes
	Pattern recognition	Observing patterns, trends and regularities in data
	Simulation	Developing a model to imitate real-world processes
BBC Bitesize (BBC, 2018)	Decomposition	Breaking down a complex problem or system into smaller, more manageable parts
	Pattern recognition	Looking for similarities among and within problems
	Abstraction	Focusing on the important information only, ignoring irrelevant detail
	Algorithms	Developing a step-by-step solution to the problem, or the rules to follow to solve the problem
Barr and Stephenson (2011)	Data collection	The process of gathering appropriate information
	Data analysis	Making sense of data, finding patterns and drawing conclusions
	Data representation	Depicting and organizing data in appropriate graphs, charts, words or images

(continued)

www.dbooks.org

Table 14.1 (continued)

Organization/Researchers	CT skills	Definitions
	Problem decomposition	Breaking down tasks into smaller manageable parts
	Abstraction	Reducing complexity to define main idea
	Algorithms and procedures	Series of ordered steps taken to solve a problem or achieve some end
	Automation	Having computers or machines do repetitive or tedious tasks
	Parallelization	Organize resources to simultaneously carry out tasks to reach a common goal
	Simulation	Representation or model of a process. Simulation also involves running experiments using models
Berland and Lee (2012)	Conditional logic	Use of 'if-then-else' construct
	Algorithm building	Set of instructions
	Debugging	Act of determining problems in order to fix rules that are malfunctioning
	Simulation	Modelling or testing of algorithms or logic
	Distributed cognition	Rule-based actions
Lee, Mauriello, Ahn, and Bederson (2014)	Algorithmic thinking	Logical steps required for constructing a solution to a given problem
	Decomposition	Process of breaking down a large problem into smaller sub-problems or details
	Pattern recognition	Identification of similar phenomenon
	Pattern generalization and abstraction	Solving problems of a similar type because of past experience solving this type of problem
	Unarticulated instances	Performing an action as a result of thought process that could not be articulated
Kazimoglu, Kiernan, Bacon, and MacKinnon (2012)	Problem-solving	Decomposing the problem to generate alternative representations and to determine if the problem can be solved computationally
	Building algorithms	Construction of step-by-step procedures for solving a particular problem
	Debugging	Analysing problems and errors in logic or in activities
	Simulation	Demonstration of solution to proof that it works

(continued)

Table 14.1 (continued)

Organization/Researchers	CT skills	Definitions
	Socializing	Brainstorming and assessment of incidents/strategies among multiple parties
Moreno-León, Robles, and Román-González (2015)	Abstraction and problem decomposition	The ability to break a problem into smaller parts that are easier to understand, program and debug
	Logical thinking	Use of instructions to perform different actions depending on the situation
	Synchronization	Use of instructions to enable the characters to perform in the desired order
	Parallelism	Use of instructions to make several events can occur simultaneously
	Flow control	Use of sequence and repetition blocks to control the behaviour of characters
	User interactivity	Allows players to perform actions that can provoke new situations in the project through the use of input peripherals such as keyboard, mouse or webcam
	Data representation	Ability to set information for all characters in order to execute the program properly

tions. For our work in this chapter, we will use the CT skills defined by (Moreno-León, Robles, & Román-González, 2015).

14.3 Methodology

Data was collected from 736 first-year undergraduate students taking an 'Introduction to Computers' course at a private university in Malaysia. This course teaches students the basic concepts of what computers are, how computers store and process information, communicate with each other and applications of computers in daily life. The course also covers a brief introduction to programming, more specifically software design lifecycle, basic programming constructs and the different types of tools that can be used to develop software applications. Students were taught how to use Microsoft© Office tools to solve problems and basic game design using MIT Scratch (MIT, 2016) during the practical sessions of the course. Scratch was chosen as the development platform so that students can focus on the design of the solution instead of the syntax of a particular programming language. The students were from two different schools—School of Computing and School of Business. Table 14.2 shows the composition of students from each school.

For the individual assignment, students were tasked to design a Catching Name Game—a game where the objective of the game is to collect characters that appear

www.dbooks.org

Table 14.2 Gender distribution and composition of students from each school

Gender	Computing	Business	Total
Male	114	213	327
Female	41	368	409
Total	155	581	736

on the screen to spell out words. The students were given the freedom to determine the type of gameplay that they wish to submit for the assignment but the game must include components of their own name inside the game to reduce the possibility of students passing off someone else's work as their own. An online questionnaire was administered on the students after they have submitted their Scratch Assignment to collect information on their gaming habits. Through the questionnaire 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 the favourite video game), refer to Appendix 1 for the full questionnaire. 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 gameplay of that particular game.

The starting age at which students reported their first foray into games and the responses from the open-ended question was 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 nonexistent/invalid games for either instance were ignored in the study. If the game title provided by the respondents at either point of times—young and current—is valid, the respondents were categorized as having 'previous gaming experience'. The assessment of computational thinking skills of the students' Scratch project was done using the free web-based tool Dr. Scratch. The tool analyses each Scratch project in seven CT dimension (Moreno-León, Robles, & Román-González, 2015). Each dimension was then given a score that ranges from 0 to 3 according to the criteria provided in Table 14.3. The addition of the partial score from each dimension yields a CT Score and, based on this score, different feedback was also provided by the website to provide the student's information and suggestions on improvement.

Since each of the CT dimension constitutes an ordinal variable and 'Gaming Experience' is a nominal categorical variable, Cochran–Armitage test of trend was utilized to investigate the relationship between each CT dimension and 'Gaming Experience'. All statistical analysis was conducted using SAS Enterprise Guide software.

Table 14.3 Description for each CT dimension assessed by Dr. Scratch (Moreno-León, Robles, & Román-González, 2015; Robles, Moreno-León, Aivaloglou, & Hermans, 2017)

CT dimension	Basic (1 point)	Developing (2 points)	Proficient (3 points)
Abstraction and problem decomposition	More than one script and more than one sprite	Definition of blocks (creation of custom blocks)	Use of clones (instances of sprites)
Parallelism	Two scripts on green flag	Two scripts on key pressed or on the same sprite clicked	Two scripts on when I receive message, or video or input audio or when backdrop changes to
Logical thinking	Use of 'If' blocks	Use of 'If … else' blocks	Logical operations
Synchronization	Wait	Message broadcast, stop all, stop program	Wait until, when backdrop change to, broadcast and wait
Flow control	Sequence of blocks	Repeat, forever	Repeat until
User interactivity	Green flag	Key pressed, sprite clicked, ask and wait, mouse blocks	Webcam, input sound
Data representation	Modifiers of sprites properties	Operations on variables	Operations on lists

14.4 Results and Discussion

Based on the results obtained in Table 14.4, it was observed that there is a strong evidence (p-value = 0.0100) of an association between the CT dimension of Abstraction and Problem Decomposition and gaming experience of students. A plausible explanation for this correlation is that all games, regardless of game genre, have goals/missions and a reward mechanism that entices players to continue playing—an attribute which makes games engaging. Players would then try to find ways to maximize these rewards while minimizing damage to their game characters during the gameplay (Ch'ng, Lee, Chia, & Yeong, 2017; Gee, 2008). 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, which may differ greatly depending on the game, that they have at hand. It was noted in (Adachi & Willoughby, 2013) that these were also the exact features that promote problem-solving skills. We posit that, perhaps, this is the attribute of COTs games that provide informal training to its players in the CT dimension of Abstraction and Problem Decomposition.

It was also observed that there is a weak evidence (p-value 0.0470) to support the hypothesis of an association between the CT dimension of User Interactivity and gaming experience. A possible explanation to this phenomenon is that when

www.dbooks.org

Table 14.4 Cochran–Armitage trend test results

CT dimension	Cochran–Armitage z-statistic	*p*-value
Abstraction and problem decomposition	2.3274	*0.0100*
Parallelism	−0.3318	0.3700
Logical thinking	0.6346	0.2628
Synchronization	−0.2861	0.3874
Flow control	0.9793	0.1637
User interactivity	−1.6750	*0.0470*
Data representation	0.5875	0.2784

students are exposed to video games and through repeated play over the years, they will indirectly pick up the basic elements needed to interact with computer software such as use of keyboard to input text and mouse to make selections; compared to those who have minimal or no exposure. Further investigation needs to be conducted if the same observation applies to those who are exposed to repeated general software usage and not only video games to determine if this observation holds.

Since the *p*-values of the Cochran–Armitage z-statistic is greater than 0.05 for the other CT dimensions considered in our study, we conclude that there is insufficient evidence from our sample to support the hypothesis that there is an association between 'Parallelism', 'Logical Thinking', 'Synchronization', 'Flow Control', 'Data Representation' and gaming experience of the students, respectively.

14.5 Implications for Educators and Researchers

Our findings show that there is a correlation between previous gaming experience and the CT dimensions of Abstraction and User Interactivity. At this moment, we cannot tell which particular aspects of specific COTs video games that support the cultivation of these particular skills or why the correlation exists without further investigations. However, our findings does encourage the idea that COTs games possess the potential to cultivate skills as suspected by Klopfer, Osterweil, and Salen (2009), Shreve (2005). The question now is: How do we actually harvest it to make it work for CT education?

The report by Klopfer, Osterweil, and Salen (2009) has presented creative ways in which games can be incorporated into classrooms to support different aspects of learning. The most common approaches that are currently utilized for CT education is the use of games as programming and reflective systems via game development assignments and serious games respectively. However, the creation of serious games takes time and skills that most educators do not possess and pale in comparison

to the expertise possessed by the games industry. An alternative is to use COTs games instead of serious games. The online content provided by (BBC, 2018) on thinking computationally utilizes a simple platform game to illustrate each aspect of CT skill without any programming. It should also be possible for educators to do the same in the classroom. For example, students can be tasked to pick and play their own choice of game from a pool of games (Blizzard, 2018) and subsequently asked to share about the challenges that they faced while trying to clear particular levels of the game (abstraction and problem decomposition) and how they overcome those challenges (algorithm). Another approach is to group students according to the choice of games that they have previously picked to have smaller discussions and knowledge exchange on how particular levels of the game are solved. A simple worksheet is included in Appendix 2 for students to work on before the face-to-face session on introduction of CT concept Abstraction and Problem Decomposition. The students' work can also be used as examples for discussion on the CT component of Algorithmic Thinking.

Another issue that educators face with incorporating games into the classroom is the struggle to cover the mandated curriculum and have games within the same block of time. An unorthodox solution to this problem is to leave play outside the classroom, in its original place, so that students have the freedom to play (explore, experiment and fail without penalties). The limited face-to-face time within the classroom can be used to facilitate discussions on what students did and to guide them to reflect on their own observations or actions during the play. Class activities that can be done within the classroom may be something as simple as free discussions of their gaming experiences of popular game titles. When conversations are started by students and led by students, students are given the opportunity to identify their own strengths and build their confidence in communication and social skills. The study by (Berland & Lee, 2012) observed that players exhibit skills to identify problems and build solutions based on their own observation and actions during gameplay. The same attribute can probably be observed on online gaming forums where players virtually gather to discuss issues that they face while trying to complete levels of a video game. However, with this approach, it is difficult to standardize the learning outcome or control the actions of the students since it is not within the teachers' power to control students' actions outside the classroom as it is within the classroom. A solution to this is to assign students the task to create a piece of work (video/written piece/blog) at the end of the semester demonstrating how they identify and solve problems that they face while playing their favourite games. For those who are unenthusiastic about gaming, contrary to common beliefs that everyone plays video games, the act of gaming can also be replaced with their own hobbies—crafting, sports, collecting items.

The idea of incorporating video games into education is an idea that has been around for some time (Annetta, 2008). In fact, video games have been reportedly used to teach city planning (Terzano & Morckel, 2017) and English as a Second Language (Miller & Hegelheimer, 2006) with positive feedback. Video game players were also found to fare better at surgical skills in the study conducted by (Rosser et al., 2007) and were noted to have better graduate attributes than non-gamers in

www.dbooks.org

(Barr, 2017). For the field of CT education, we believe that video games have the same potential to be used as an effective tool for teaching and learning within and outside the classroom. The results reported in our study provide preliminary proof of this. In the future, we plan to obtain concrete evidence advocating the use of video games as a potential vehicle for learning and teaching computational thinking by implementing the ideas put forth in a real classroom environment.

Appendix 1: Survey—Video Game Experience

We are conducting a research about the relationship between previous gaming experience and game development. Your response to these questions will not affect your grades for this subject. Please answer the following questions.

1. Scratch Project URL: _____
2. Gender

   ☐ Yes
   ☐ No

3. Do you play video games when you were younger?

   ☐ Yes (Go to question 3.)
   ☐ No (Go to question 6.)

4. How old were you when you started playing video games?

   ☐ Less than 6 years old
   ☐ 7–9 years old
   ☐ 10–12 years old
   ☐ 13–15 years old
   ☐ 16–18 years old

5. Name your favourite video game when you were young. _____
6. How often do you play this game back then?

   ☐ Rarely
   ☐ Occasionally
   ☐ Frequently

7. Do you actively play video games over the past two years?

   ☐ Yes
   ☐ No

8. Name your current favourite video game. _____

9. How long do you spend each day, on average, playing your favourite video game?

☐ <1 h
☐ 1–2 h
☐ 3–4 h
☐ 3–4 h
☐ 5–6 h
☐ >6 h

Appendix 2: Homework Exercise—Describing My Favourite Game

Your task in this exercise is to describe the steps that you take to play one of the games that you frequently play at home. Games in this case can be any type of games ranging from board games, video games on the personal computer, mobile phone or television consoles or even physical activity game.

Components	Instructions
Game title	*Name of the chosen game*
Game description	*Provide a short description of the chosen game*
Goal of the game	*State the main goal of the game that players must fulfil to win the game*
Game strategy	*Describe the steps that you take as a player to achieve the goal of the game. You can include screenshots or images to aid your explanation. Be as detailed as possible in your description so that another new player can use your description as a walkthrough to complete the game on his own*
Sub-objective(s) of the game	*[Optional] Some games have mini-games/missions embedded inside the game itself. If the game has many mini-games/missions, select one and state the goal that players must fulfil in order to win or complete the mini-game/mission*
Mini-game/Mission strategy	*[Optional] Describe the steps that you take as a player to achieve the goal of the mini-game/mission within the game. You can include screenshots or images to aid your explanation. Be as detailed as possible in your description so that another new player can use your description as a walkthrough to complete the same mini-game or mission on his own*

www.dbooks.org

References

Adachi, 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.

Annetta, L. A. (2008). Video games in education: Why they should be used and how they are being used. *Theory into Practice, 47*(3), 229–239.

Barba, L. A. (2016). Computational thinking: I do not think it means what you think it means. Retrieved January 11, 2018, from http://lorenabarba.com/blog/computational-thinking-i-do-not-think-it-means-what-you-think-it-means/.

Barr, M. (2017). Video games can develop graduate skills in higher education students: A randomised trial. *Computers & Education, 113,* 86–97.

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.

BBC. (2018). BBC—Introduction to computational thinking. Retrieved from https://www.bbc.co.uk/education/guides/zp92mp3/revision/1.

Becker, K. (2001). Teaching with games: The minesweeper and asteroids experience. *Journal of Computing Sciences in Colleges, 17*(2), 23–33.

Berland, M., & Lee, V. R. (2012). Collaborative strategic board games as a site for distributed computational thinking. *Developments in Current Game-Based Learning Design and Deployment, 285*.

Blizzard. (2018). Blizzard entertainment: Classic games. Retrieved January 24, 2018, from http://eu.blizzard.com/en-gb/games/legacy/.

Ch'ng, S., Lee, Y., Chia, W., & Yeong, L. (2017). Computational thinking affordances in video games. *Siu-Cheung KONG The Education University of Hong Kong, Hong Kong, 133*

Fletcher, G. H. L., & Lu, J. J. (2009). Education Human computing skills: Rethinking the K-12 experience. *Communications of the ACM, 52*(2), 23–25.

Frazer, A., Argles, D., & Wills, G. (2008). The same, but different: The educational affordances of different gaming genres. In *Eighth IEEE International Conference on Advanced Learning Technologies, 2008. ICALT '08.* (pp. 891–893). IEEE.

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.

Gee, J. P. (2008). Learning and games. In K. Salen (Ed.), *The ecology of games: Connecting youth, games, and learning* (pp. 21–40). MIT Press.

Google. (2018). Google for education: Exploring computational thinking. Retrieved January 11, 2018, from https://edu.google.com/resources/programs/exploring-computational-thinking/#!ct-overview.

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.

Klopfer, E., Osterweil, S., & Salen, K. (2009). *Moving learning games forward*. Cambridge, MA: The Education Arcade.

Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE), 14*(4), 26.

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction, 2*(1), 26–33.

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.
Miller, M., & Hegelheimer, V. (2006). The SIMs meet ESL incorporating authentic computer simulation games into the language classroom. *Interactive Technology and Smart Education, 3*(4), 311–328.
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.
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, 46,* 1–23.
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.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning, 1*(1), 95–123.
Robles, G., Moreno-León, J., Aivaloglou, E., & Hermans, F. (2017). Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning. In *2017 IEEE 11th International Workshop on Software Clones (IWSC)* (pp. 1–7). IEEE.
Rosser, J. C., Lynch, P. J., Cuddihy, L., Gentile, D. A., Klonsky, J., & Merrell, R. (2007). The impact of video games on training surgeons in the 21st century. *Archives of Surgery, 142*(2), 181–186.
Shreve, J. (2005). Let the games begin. Video games, once confiscated in class, are now a key teaching tool. If they're done right. *George Lucas Educational Foundation*.
Terzano, K., & Morckel, V. (2017). SimCity in the community planning classroom: Effects on student knowledge, interests, and perceptions of the discipline of planning. *Journal of Planning Education and Research, 37*(1), 95–105.
Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. In *ACM SIGCSE Bulletin* (Vol. 33, pp. 184–188). ACM.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 15
Transforming the Quality of Workforce in the Textile and Apparel Industry Through Computational Thinking Education



Bessie Chong and Ronald Wong

Abstract The goal of introducing computational thinking education in a business setting is not just for technology adoption or invention but it is considered to be a vital skill for empowering employees to address problems critically and systematically. This chapter presents a case study of Esquel Group, a world leader in the textile and apparel sector, to leverage computational thinking (CT) in learning and education to lift staff competence. Esquel believes in the era of Industry 4.0, employees should embrace a new way of thinking by combining logical reasoning and computational power that would empower them with the ability to think independently. Therefore, the "You Can Code" campaign was initiated in 2015 to the staff at all levels with over 1,200 participants from 10 different locations in the first 10 months. By learning how to build a mobile application using MIT App Inventor, the problem-solving learning process is woven into the employees' minds on capacity building. They can learn how to decompose problems, synthesize ideas, and develop an algorithm to solve problems in a structural and creative way. The campaign plays a catalytic role to empower employees as confident users of technology with computational thinking ability and helps the company to nurture a culture of innovation, problem-solving, and collaboration. After the success of the first run, the company tried to further engage young colleagues to stimulate computational thinking and innovation. A series of more comprehensive App Inventor Workshops, including developing mobile apps with Internet of Things and Arduino, was introduced and implemented locally by the young generation. It shows innovation in learning and also demonstrates that Esquel is a nontraditional company which strives to make a difference.

Keywords Computational thinking · Vocational learning · App inventor · Computational action · Employee empowerment

Bessie Chong—Ph.D., Director of Group Training and Talent Management, Esquel Group.
Ronald Wong—Former Associate Director of Corporate Communications, Esquel Group.

B. Chong (✉) · R. Wong
Esquel Group, 13/F, Harbour Center, 25 Harbour Road, Wanchai, Hong Kong, China
e-mail: Chongbe@esquel.com

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_15

15.1 Introduction

The textile and apparel industry has long been regarded as "traditional" and "old-fashioned". It is probably not an industry the new generation aspires to join. Founded in 1978, Esquel started as a shirt maker. Over the last 40 years, Esquel 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 offers a one-stop solution, from concept to rack. The annual sales turnover was US$1.3 billion in 2016.

Esquel employs more than 57,000 diversified workforce globally, united under the corporate 5E culture—Ethics, Environment, Exploration, Excellence, and Education, and driven by the mission of "Fun People Serving Happy Customers". It operates with an aspiration of "Making a Difference" by creating a positive impact on the employees, communities, and environment. The key employee development strategy is to "groom people from within". As a nontraditional company in a traditional industry, Esquel encourages and empowers employees to innovate and to challenge the status quo by placing great emphasis on learning and people development in facilitating employees to transform and upgrade.

15.1.1 Business Challenges and Opportunities

In this day and age, the competition in the industry of textile and apparel manufacturing is fierce. All players face structural challenges from rising labor and material costs, reduced profit margin, and shortened order lead time to the shortage of skilled labors. The rise of fast fashion further disrupts the industry by demanding quicker production cycles and rapid prototyping in small orders. The traditional manufacturing model of long lead time and mass production would find it challenging to survive.

However, the textile and apparel manufacturing industry employed over 75 million people worldwide (Stotz & Kane, 2015) with an aggregate export amount of 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 while enabling sustainable growth amidst the changing environment? Many players in this industry migrates their manufacturing bases and chase after cheap labor to stay competitive. On the contrary, Esquel decided to stay in locations where it has good operating conditions and to cultivate the local talent pool. Esquel strives to improve labor productivity to offset rising wages. The company recognizes the importance to improve the added-value of their people, to provide an inclusive work environment and to pay them well by integrating them into the technology, rather than replacing them with technology.

www.dbooks.org

The advent of the Fourth Industrial Revolution, also known as Industry 4.0, is associated with the development of global industrial networks, to which all production processes of a wide variety of enterprises will be connected throughout the global supply chain. As a result, computer interaction environment is developed around the modern human (Yastreb, 2015). That means employees would work with cyber-physical systems in a smart factory environment and make use of the Internet of Things (IoT) technology and data collected to streamline operation, empower lean supply chains, and make timely decisions. Ultimately, it will increase supply chain agility, adaptability, and alignment (Lee, 2004) that foster productivity and efficiency. The Fourth Industrial Revolution provides Esquel with an opportunity to sustain its leading position in apparel operations.

While digitalization is bringing great impact to the current business models and operations in the global economy, enterprises need to identify the new value creation opportunity in the process of moving to digital business. Esquel is expected to tap into the digitalization process and optimize its supply chain. However, to facilitate this transformation, it is vital for the employees to have some understanding of computer programming regardless of the profession they are in. Programming will soon become a basic job skill for everyone. The rise of robotics and artificial intelligence calls for new skills and competencies. The new age employees need to be equipped with a new set of skills in order to master technology, explore new possibilities and convert the new ideas into actions. More importantly, we need to train them on how to think systematically through developing their programming ability.

15.1.2 People Challenges

Without the right people, the effectiveness of technology would not be maximized or create a positive impact on the business results. Among Esquel's 57,000 employees, only 3.4% of them are equipped with formal technical qualifications, 12% of them possess a college diploma or above, with 38% of them were born before personal computer became popular. The fear of using technology and the shortage of computer-literate employees soon become the barrier of transformation. The challenge is how can we turn employees into confident technology users? How can we empower them to come up with continuous improvement ideas and solve the daily work problems systematically and independently?

A campaign is needed to drive this transformation and to inspire the employees to participate in the revolution. It would be a huge challenge as the target group is highly diversified in culture, age, and education, and spreads over 9 countries in 20 operation sites. It would also be hard to keep the learning momentum.

15.2 "You Can Code" Campaign (2015–2016)

Programming is a skill that helps people learn how to think systematically. By developing computational thinking, people can break down complex problems into manageable parts, look for similarities among and within problems and identify different recommendations step by step. For people who don't have technical knowledge, computational thinking may seem too abstract and programming may seem too technical. The fun and practical "App Inventor" application developed by the Massachusetts Institute of Technology (MIT) was therefore identified as the main 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 programming into visual, drag-and-drop building blocks. The easy-to-use interface would change employees' perception of technology adoption and overcome the fear of using IT through this campaign. It further develops 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, Good, Voogt, & Fisser, 2017). By equipping employees with computational thinking ability, the company can empower them to become innovative problem solvers, collaborators as well as process owners. Yadav et al. (2017) further stressed that 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.

> 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).

In Esquel, whether they are workers, general staff, managers, or executives, employees are required to have the right attitude, ability to solve problems and to turn ideas to practical solutions to boost productivity. The "You Can Code" campaign aims to drive a sustained cultural transformation to turn the less technically minded employees into confident users of technology and even the creator of technology.

15.2.1 Champaign Design and Implementation

This campaign is designed around how to change **A**ttitudes, upgrade **S**kills, and build **K**nowledge, as shown in Fig. 15.1.

It is impractical if only IT colleagues are involved in providing classroom training and expect employees to change their attitude towards technology. Therefore, the role of IT throughout the campaign is purposely downplayed to convince all employees that programming can be trained to less technically savvy people. An "all-in"

www.dbooks.org

approach was adopted. The campaign was carried out in five development phases: (1) Pioneering, (2) Modeling, (3) Changing, (4) Cultivating, and (5) Realizing, as shown in Fig. 15.2.

The first round of "You Can Code" campaign started in 2015– 2016. The campaign used top-down and bottom-up approaches to engage colleagues from all levels. Platforms included Yammer (Esquel's internal social media networking tool), WeChat, Intranet, company's TV broadcasting, as well as traditional channels such as notice boards and promotional booths at factories were used to educate colleagues and promote the training workshops and "Esquel's App Challenge". Through 28 workshops, over 2,430 training hours were provided to 1,200 participants, including 1,100 employees and 100 of their children from 10 different locations in the first 10 months (Fig. 15.3). The strategy of teaching the children and letting them teach their parents back was proved to be effective. Overall, the impact was encouraging and a lot of positive feedback was received:

> Something looks complicated but can be very user-friendly for us in building an app. Useful and valuable information/tools can be shared with the company!
>
> — A Sales Manager in Hong Kong

> The introduction of the online programme 'App Inventor' is useful for non-professionals to build our own app.
>
> — A Senior Sales Executive in Hong Kong

> Easy to operate for dummies. All ordinary people can participate in creating an app without the support of IT.
>
> — An Engineering Officer from a Garment Factory in China

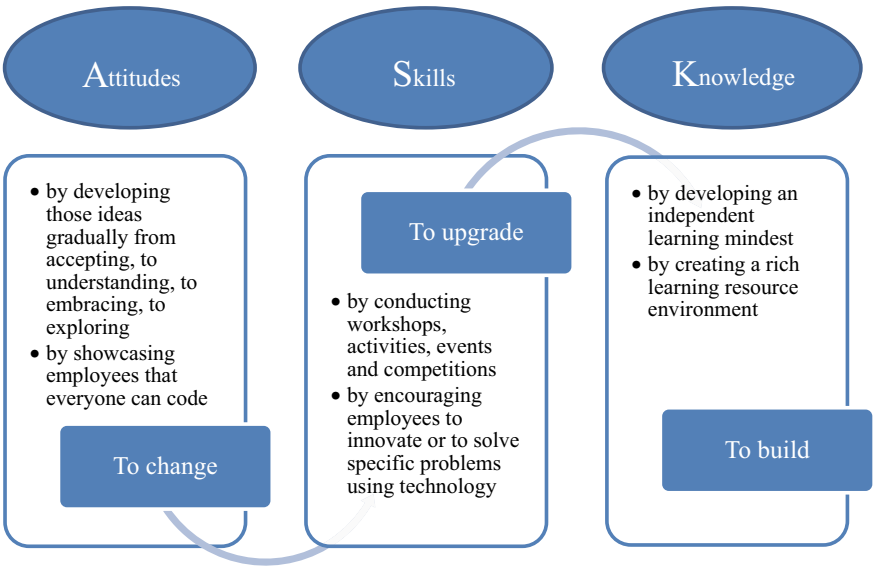
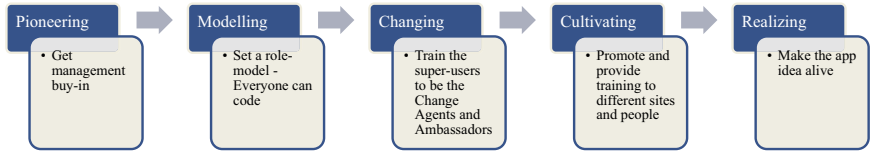


Fig. 15.1 ASK model



Workshops for the senior management team and board members were conducted to collect their feedback and get their buy-in. About 90 percent of them attended the training. Some of them also became Esquel's pioneers.	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 of 'If I can code, you can code too.'	About 50 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.	The master trainers launched a series of workshops and fun days for staff and their kids in order to cultivate the skills and mindset. The ambassadors set up information and promotional booths to educate frontline operators. A total of almost 800 people were trained.	The first "Esquel's App Challenge" Competition was organized to encourage the application of the new skills. Over 400 mobile app ideas were submitted. Many interesting and practical apps were developed.

Fig. 15.2 Five development phases and achievements



Fig. 15.3 Young colleagues and children of Esquel staff members learn how to code in an hour

15.2.2 Value Created from the Campaign

The most important impact of this enterprise-wide campaign 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 part of the IT routine tasks, and some are even able to build prototypes by themselves. This, in turn, allows IT professionals to focus on enterprise-level app development.

The above prototype mobile apps (Table 15.1) were developed by non-IT colleagues based on their local needs. These applications help save time and improve efficiency, while the broader benefits are incalculable. Department heads and IT

www.dbooks.org

team are reviewing many more bottom-up initiatives from employees to further enhance production efficiency. This campaign is an example of Esquel's commitment to upgrading their workers to become more knowledgeable. And at the same time, the campaign also reinforces Esquel's employer brand as a caring and nontraditional company.

15.2.3 Employee Empowerment—From Reactive to Proactive, from Follower to Owner

In previous days, the company needed to actively explore innovative solutions to engage and help employees to embrace technology. Now, we can see employees are actively looking for innovative solutions to improve their work and life.

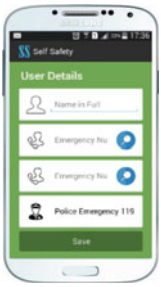
In the process of developing their own mobile applications, employees started to integrate computational thinking into their everyday work. They took attempts to analyze problems by breaking them down and identifying the root cause, instead of jumping to quick fixes. The story of Yang Hua Mei illustrates how a basic programming training can bring an impact on a sewing worker.

Yang Hua Mei is a young woman from the southwest of China with an immense interest in fashion design and a desire to build a career in apparel manufacturing (Fig. 15.4). 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. During the campaign, Hua Mei learned the basic technical skills, such as computational thinking, logical reasoning, and simple programming to turn her undeveloped fashion ideas to life. By the end of the campaign, Hua Mei and two other colleagues built an app allowing users to mix-and-match their wardrobe.

> 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—no matter how simple an app might seem, building one requires many steps and logical thinking!, said Hua Mei.

She also realized that the basic programming technique equipped her with computational thinking ability, which in turn helped her to 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 learned how to code, whenever she came across problems, she would simply ask the technician to fix it or change some machine parts by herself. She had never bothered to understand the problems, the root causes, and thought about how to prevent them in the future. But now, she becomes proactive in learning technical skills and starts to integrate the computational thinking ability to solve her daily work problems. She also aspires to evolve from a sewing worker to a technician one day.

Table 15.1 Examples of the mobile apps developed in this campaign

Applications	Functions
	***Safety app for women*** A Sri Lankan colleague developed this app with a GPS function for female colleagues leaving work at night. Female colleagues can set the time and contact and if they do not arrive home at the specified time, the contact number will be dialed.
	***Parking space and free bike locator app*** This app helps colleagues to find a parking spot or an available bicycle to be borrowed near operation sites through its real-time parking space and bicycle availability information.
	***Recruitment app*** This app helps Human Resources colleagues to streamline some manual work during the recruitment process, such as marking test paper and personnel data entry. Applicants can key in their basic personal information in the app and conduct some simple aptitude tests.
	***Newborn baby photo sharing app*** This app functions like a greeting card. It allows new parents (the colleagues) to send their first newborn baby's photo to their friends with the baby information and Esquel's hospital logo.

www.dbooks.org

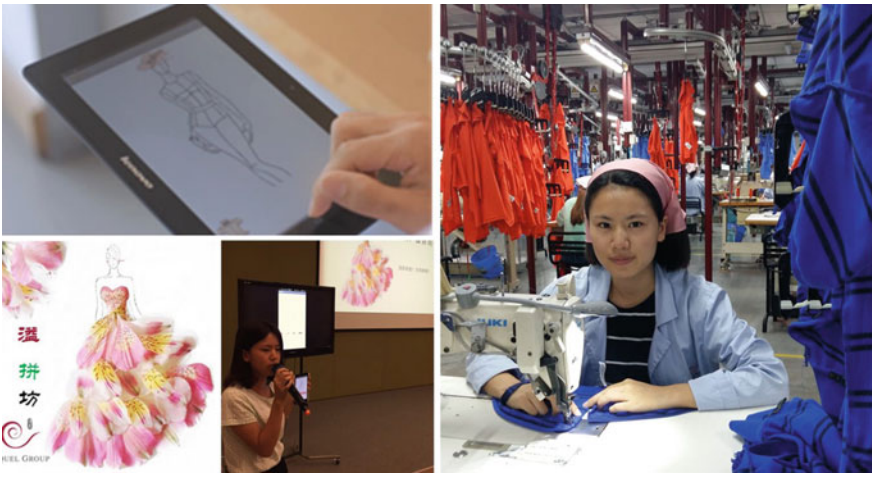


Fig. 15.4 Interface of the wardrobe application developed by Yang Huamei's team and a photo of Yang Hua Mei at work and training

15.3 From Computational Thinking to Computational Action

This first phrase of "You Can Code" campaign was completed in 2016. It helped the employee to leverage technology to solve their daily issues and improve productivity. Indeed, it started the momentum. The rise of mobile app initiatives after "You Can Code" campaign is the result of the empowerment through computational action.

15.3.1 Development of Esquel Carpool App

Esquel Carpool App is an impactful commercialized mobile app developed by a factory colleague using software such as Objective-C, Java, and Node.js. It aims in solving the air pollution, traffic congestion and carbon footprint challenges. The impact is enormous.

According to the data provided by the Chinese Environmental Protection Bureau, 15−30% of the pollution comes from the car emission (Chinese Environmental Protection Bureau, 2016). With an exponentially increasing number of cars, traffic congestion is now seen everywhere in China.

Esquel's largest operation base is located in Gaoming, Foshan. It has about 23,000 employees working in several factories that are spread over the city of Gaoming. Around 40% of Esquel employees work and live there. Employees commute from home to these working locations in similar timing and similar routes every day. Many of them take company shuttle or city bus and always suffer from waiting in a long

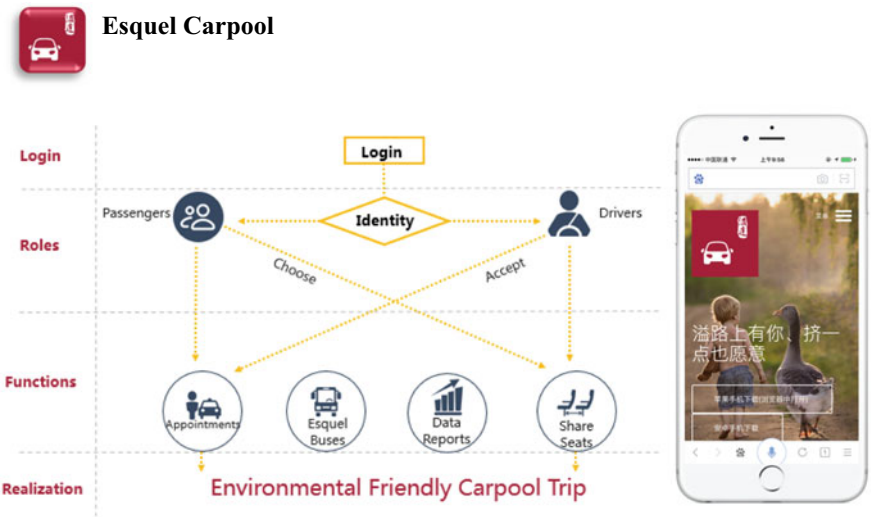


Fig. 15.5 Design map and login page of Esquel Carpool application

line under the 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 not better than the others. Driving to work is not at all pleasant when they have to be stuck in traffic and fight for the limited 200 parking spaces available around the factories. Most of the time, employees have to park far away and take another 10-minute walk back to the office.

How can Esquel make a difference for the colleagues so that they can save time in waiting for the bus, fighting for the traffic or looking for parking space? How can they save on gasoline bills while reducing carbon footprint?

Can something be done to change their lifestyle and behavior, reduce the environmental impact, and inspire others to contribute to building a green city?

15.3.2 The Idea of Esquel Carpool App

To tackle the above challenges, an employee in Gaoming, Guangdong, China factory initiated an idea to develop an app to facilitate the carpool process in Esquel. Inspired by the "You Can Code", the Esquel Carpool App was born in June 2016 (Fig. 15.5). It shows that colleague looks proactively at the technology for improving their lives and finding ways to develop this application after the "You Can Code" campaign.

To start with, employees can use their staff ID to login, and select to be either the passengers or drivers. Passengers can publish their needs (e.g., where and when they want to go) or select the available seats from the drivers. Drivers can publish the number of available seats to the passengers, or directly select the passengers

www.dbooks.org

through the app. After that, they can form a group chat to communicate directly for boarding arrangement. What's more, this app can also share the real-time location of the company bus and show the carpool usage report.

15.3.3 Impact from Esquel Carpool App

Within the first 24 months, this app has already recorded more than 139,083 carpools, with the saving of more than 105,703 liters of gasoline. It avoided the emission of 243.1 tons of carbon dioxide. This app helps to realize the benefits of carpooling on saving the environment. More importantly, it provides a platform to make a connection with colleagues from different departments that promotes the caring culture.

The company is committed to provide this app for free to any companies and organizations, and the app is readily available in the open-source community GitHub.

15.4 From Programming to Internet of Things

15.4.1 Success of "You Can Code" Campaign

Esquel is the first commercial entity to adopt App Inventor to train employees in computational thinking. Even though computational thinking is rather conceptual and hard to develop in a short period of time, the company has managed to change attitudes, upgrade skills, and build knowledge through the development of the mobile app.

The first round of "You Can Code" campaign received an initial success. The "all-in" approach encouraged everyone to engage in the campaign. Many employees, including board members and sewing workers, joined the fun and easy "1-hour programming" workshops. Employees' kids were also invited, who in turn, influenced and motivated their parents to learn programming. The campaign engaged people from primary students to Ph.D. graduates aged from 6 to over 60.

It successfully engaged all levels of staff members by enrolling board members and senior managers as pioneers, ambassadors, and trainers. Some even modeled the skills and trained their teams at their sites. They jointly promoted the notion of "If I can code, you can code too", and successfully changed the perception that senior staff are conservative and less tech-savvy.

15.4.2 Continuity of "You Can Code"

The project team understands that what they have done is just a small step. In order to build a programming culture and independent critical thinking skills, more efforts are required. Indeed, the project team also understands what the employees have learned in the 1-hour App Inventor workshop is not sufficient. The next step is how to deep dive into what they have learned to embrace the computational thinking skills and unleash their creativity. Thus, the project team needs to provide a platform to enable the participants to organize and analyze information logically and creatively. Then, they can break the challenges down into small pieces, and approach them by using programmatic thinking techniques. The exercise would reiterate the interest of the participants to learn, create, and modify their own applications.

With this in mind, the "You Can Code 2.0" campaign started in November 2017. The project team aims to take one-step further from creating a simple mobile application to developing a stimulated real-life application in the workplace. The company plans to teach colleagues essential skills to build a microcontroller-based embedded system, which is monitored by a mobile app developed by App Inventor.

The project team aims to train the participants basic programming technique connected with physical devices as an example. Then, the participants can replicate the setting and methodology to create their own IoT or Arduino prototypes. More importantly, the project team wants to teach the logic behind those programming techniques. Once the participants figure out most of the intelligent production and process systems found in the work environment are in fact programmed by the basic computer logic, they will have confidence in making recommendations and asking questions on the current practices and systems.

15.4.3 You Can Code 2.0

The explosive growth of the "Internet of Things" is changing how things are being operated. It allows people to innovate new designs and products at work and home. The capturing of big data means a huge opportunity for operation efficiency improvement. Esquel has many "Internet of Things" applications, for example, auto-guided vehicles and drones are adopted in the factory for transportation use. At the backend, a knowledge base is established and linked to the intelligent control system, which is further connected to smart devices. Thus, if the colleagues can understand the way to connect those devices to the backend system through the internet, they will be motivated to build applications and devices that could ease the manual work. Indeed, IoT comes with a combination of software and hardware. Colleagues can apply what they have learned to program the Arduino and build a smart system by connecting basic sensors and actuators for automation.

To launch the "You Can Code 2.0" workshop, a progressive and systematic learning path was designed. Three levels of workshop, according to the difficulty and

www.dbooks.org

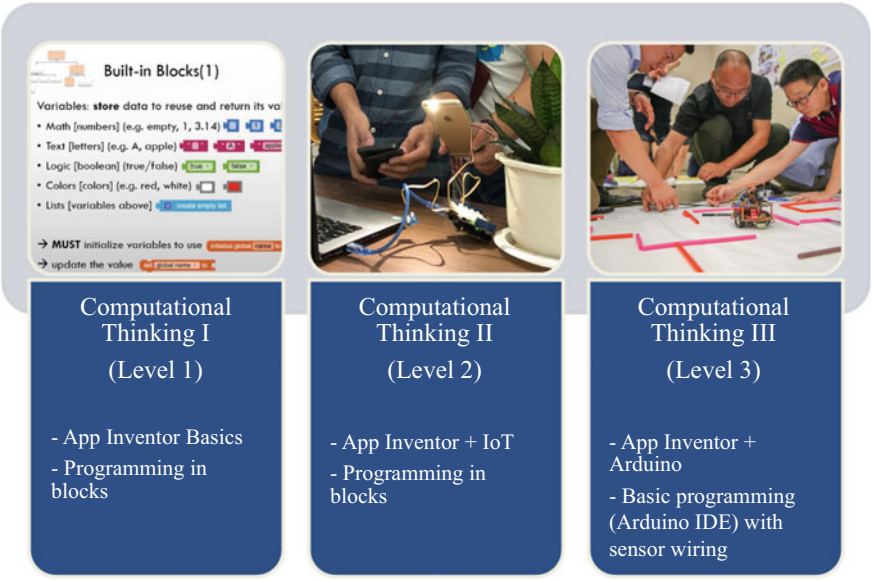


Fig. 15.6 Three levels of workshops for "You Can Code 2.0"

complexity of the app, were organized (Fig. 15.6). We targeted to recruit those who have not joined the App Inventor workshop before or those who are interested in using technology. As a result, majority of participants are young generation.

Level 1 workshop is a beginner course focusing on basic App Inventor techniques. It aims to help new participants to equip with basic skills, and to refresh the previous practice. At this level, not all block building instructions are given, except basic functions and interface of the App. Some instructions are deliberately left blank, and only partial block building instructions are provided. Participants need to complete the instructions and build the blocks according to their needs and understanding. Custom design interface and functions are recommended to demonstrate their creativity, the comprehensiveness of logic, and the ability of the computational thinking.

Level 2 workshop focuses on programming technique applications on a micro-ecosystem. Participants are required to develop a Smart Plantation Monitoring System based on IoT to monitor a small plant growth. The mobile app collects humidity and light data from the plant via a sensor node. Colleagues will then base on the data collected to build a decision-making mechanism. With the combination of the wireless sensor network, embedded development, and data transmission, colleagues will receive recommendation instantly, such as watering or giving more light to keep the plant healthier. This workshop further examines how computer programming would aid decision-making through an ecosystem experiment.

Level 3 workshop focuses on providing a simulated operation environment for problem-solving. Participants are taught on how to program the Arduino to perform

certain tasks, such as turning on the lights, making a device turns, moving forward and backward, and escaping from a maze.

After the completion of all three levels of the workshop, a second round of "Esquel's App Challenge" competition will be organized to further stimulate their creativity and problem-solving ability. To facilitate a series of workshops to different locations, more ambassadors and change agents are identified. Therefore, Esquel has incorporated this basic App Inventor training into their management trainee (MT) training curriculum. After receiving the training, the MTs will be responsible to teach colleagues at their local sites. About 100 MTs, spreading across different locations will have an exposure to the technology and take ownership in fostering the learning culture in their own locations. The project team strongly believes that peer learning is more effective in promoting the use of technology and will overcome their fear of technology adoption. More local needs on improving work-related or life-related issues can be solved by themselves with flexibility. The active participants will be the voice and the next generation of leaders who strive make a difference.

15.5 Conclusion

Esquel is forward-looking by building future skills to enhance productivity and uplift employees' competence to master technology. The "You Can Code" campaign upskills employees, opens their minds to technology and equips them for life. It also opens the potentials for organizational digital transformation. This aligns with the company's vision of making a difference and empowers teams for continuous learning and problem-solving. Esquel is committed to boost the programming skills of its colleagues and build its organizational capability. The computational thinking training will be used as a core curriculum of the Esquel University, the enterprise university, to provide a standardized learning experience to all Esquel employees in the future. It also supports the company's determination in moving towards a digital organization.

This is just the beginning of Esquel using modern technology beyond work to make lives better. The "You Can Code" campaign provides an opportunity to colleagues so they can come up with their own initiatives to make Esquel a better working environment and eventually benefit the local community. Different departments and operations are actively exploring and developing new mobile applications to improve work efficiency and work–life quality; and at the same time, the Company is coordinating a steering committee to embrace all different mobile apps under one master app, namely Esquel Pass, to provide a one-stop solution to employees.

Looking to the future, Esquel will continue to empower employees to innovate and solve problem critically and independently with computational thinking and technique.

www.dbooks.org

References

Chinese Environmental Protection Bureau. (2016). 我国雾霾成因及对策. *紫光阁*(3), 82–83. Retrieved from http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST2016&filename=ZIGU201603061&uid=WEEvREcwSlJHSldRa1FhdkJkcGkzRm1aVDg1WXA3WG9XMmJobkw3NEM5UT0=$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4ggI8Fm4gTkoUKaID8j8gFw!!&v=MjU1MTJySTlEWllSOGVYMUx1eFlTN0Ro.

Lee, H. L. (2004). The triple-A supply chain. *Harvard Business Review, 82*(10), 102–113.

Stotz, L., & Kane, G. (2015). *Facts on the global garment industry.* Retrieved February, 2015, from 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 http://stat.wto.org/StatisticalProgram/WSDBViewData.aspx?Language=E.

Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017). *Computational thinking as an emerging competence domain.* Retrieved January, 2017, from https://www.researchgate.net/publication/307942866.

Yastreb, N. (2015). The internet of things and the fourth industrial revolution: the problem of humanitarian expertise. *Journal of Scientific Research and Development, 2*(8), 24–28.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part V
Teacher and Mentor Development in K-12 Education

www.dbooks.org

Chapter 16
Teaching Computational Thinking with Electronic Textiles: Modeling Iterative Practices and Supporting Personal Projects in *Exploring Computer Science*



Deborah A. Fields, Debora Lui and Yasmin B. Kafai

Abstract Iterative design is an important aspect of computational thinking in which students learn to face challenges and persevere in fixing them. Yet we know little of how school teachers can support students in using iterative practices. In this chapter, we consider the teaching practices of two experienced computer science teachers who implemented a new 8-week long unit on making electronic textiles in their classrooms. Electronic textiles are sewn, programmable circuits with sensors and actuators on personal artifacts that provide various opportunities to learn through mistakes. Through analyses of observations and interviews with students and teachers who implemented the unit, we identified several teaching practices that supported values of iteration, revision, and working through mistakes. These included teachers modeling their own processes and mistakes in making projects, teachers modeling students' mistakes to the wider class, and supporting personalized projects that resulted in unique "bugs" or challenges for each student. After sharing examples of these practices, we consider student and teacher reflections on the ways that mistakes and iteration supported student learning.

Keywords Computational thinking · Iteration · Computer science education · Electronic textiles · Teaching practices

D. A. Fields (✉)
Utah State University, 2830 Old Main Hill, Logan, UT 84322, USA
e-mail: deborah.fields@usu.edu

D. Lui · Y. B. Kafai
University of Pennsylvania, 3700 Walnut Street, Philadelphia, PA 19104, USA
e-mail: deblui@upenn.edu

Y. B. Kafai
e-mail: kafai@upenn.edu

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_16

16.1 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).

Iterative design is an important aspect of computational thinking that involves engaging in an adaptive process of design and implementation where students learn to face challenges and persevere in fixing them (Brennan & Resnick, 2012). Yet one glaring absence in the work on iteration in computational thinking is a lack of understanding exactly how *teachers* can support such CT practices in their 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 provide 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 by connecting technology, content, and pedagogy (Mishra & Kohler, 2006).

In this paper, we focus on how two high school teachers supported iterative practice as a core CT practice during their implementation of an eight-week (~40 h) electronic textiles unit within their classrooms during the year-long *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, 2013a). 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 teachers employ to support students' iterative practice during the e-textiles unit?" Our discussion focuses on the teachers' modeling and personalization strategies to make iteration accessible in students' work, particularly through classroom practices that support iteration.

www.dbooks.org

16.2 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, "[t]hinking 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, 1989). 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 projects and applications in which learning programming can occur, whether in game design, robotics, creating apps, or constructing wearables such as e-textiles (e.g., Kafai & Burke, 2014). Here, efforts are driven by the recognition that teaching and learning programming need to be contextualized in ways that engage students' existing interests.

Because teaching computational thinking is newer than teaching programming, research has generally focused more broadly on conceptual or hypothetical contexts (Grover & Pea, 2013). One goal has been to define the actual nature of computational thinking in terms of cognition and its relationship to existing disciplines such as mathematics and engineering (NRC, 2011). Other work has focused on developing CT-focused curricula for K-12 contexts such as *AP Computer Science Principles* (e.g., Guzdial, 2016), *Exploring Computer Science* (Margolis & Goode, 2016), or in science and mathematics classes (e.g., Tofel-Grehl et al., 2017; Weintrop et al., 2016). Finally, 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 focuses 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 computational thinking tools, content, and pedagogy intersect. Approaching this effort, Mishra and Kohler (2006) described *technological pedagogical content knowledge*, relating education with digital technology more broadly, but not specifically with computing or computer science. Instead, deeper understanding on computer science teaching is still in early stages compared to other disciplines such as math and sciences. For the most part, research on actual computer science teaching, with a focus on CT, has focused 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, Pirman, & Gray, 2016). The area that overlaps most with CT is focused on algorithmic thinking (Ragonis, 2012). A few scholars such as Margolis, Goode, and Ryoo (2015) have

studied CS teachers' pedagogical practices, such as inquiry-based strategies, but not specifically how pedagogy connected with CT or CS concepts.

Our work contributes to this newly emerging body of *computational pedagogical content knowledge* by examining how experienced computer science teachers teach CT using electronic textiles. Early studies of e-textiles 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, 2014a). One study (Kafai et al., 2014a) 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. For example, this project included concepts such as sequencing, since students had to properly sequence code in order to coordinate behavior of the sensors and lights in their project, as well as practices like remixing and reusing code, since students had to modify a sample program in order to accommodate different circuit diagrams, sensor types, and intended behaviors. However, while that study identified ways that making e-textiles can support CT, the unit was taught by researchers and pedagogy was not a focus of the study.

In this chapter, we focus on how e-textiles and pedagogy can be used to support the essential CT practice of *iterative design*, looking at it as a core computational thinking practice alongside pedagogy. There are many areas of computational thinking to address, so why did we choose iterative design? This choice stems from an analysis of teaching practices that supported equity and broadening participation in two ECS classrooms that piloted an e-textile unit (Fields, Kafai, Nakajima, Goode, & Margolis, 2018a). We noted in our analysis that teachers' support for iteration happened at multiple levels: whole class modeling, individual student work, collaborative student work, and in the design of the classroom learning environment itself. Thus iterative design stood out as an area of equitable teaching practice that simultaneously broadened participation in computer science while introducing and reinforcing an important practice of computational thinking.

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 and is therefore considered a key *conceptual idea* within CT, alongside conditional logic, abstraction, and algorithms (Brennan & Resnick, 2012; Grover & Pea, 2013; Wing, 2006). However, the act of iteration itself is also thought of as an important *practice* within CT in and of itself, when actually engaging with computational problems and projects. As outlined by Brennan and Resnick (2012), iterative design is defined as the cycle of prototyping, testing, and revision, that requires students to engage in a continually "adaptive process," throughout the course of creating a computational artifact, where one's goals "might change in response to approaching a solution in small steps" (p. 7). Iterative design is also a key practice highlighted in new secondary school computer science curricula such as the AP Computer Science Principles course, which reinforces the importance of iterative software development as "essential knowledge" in many different learning objectives in the course (e.g., CollegeBoard, 2016: principles EK 1.1.1B, 3.1.1A, 4.1.1A, 4.1.1D, 4.1.2G, 5.1.2A, 5.1.3C, 5.5.1J). From this perspective, iteration is not only a

www.dbooks.org

concept that can be applied into the body of a computer program itself, but can also refer to the purposefully incremental process of creating a computational artifact. Here, it is not only the act of iteration that matters, but an awareness and explicit acknowledgment of its importance in tackling problems in a systematic way—basically, learning how to think in an iterative way about real-world issues.

Iterative practices are also important within the field of engineering where they involve engagement with trial-and-error processes, along with revision and refinement of ideas over time (Barr & Stephenson, 2011; Lee et al., 2011). Within engineering education, these processes of iteration and revisions are more formally structured into classroom practice through the model of the engineering design process, which highlights the steps of prototyping, testing, and redesign (Tayal, 2013). From this perspective, the practice of iterative design should be considered something to be supported within both CT curricula and contexts. However, because CT-focused curricula and pedagogy are newer, there remains a critical need to highlight how iterative design as a practice can be supported through pedagogical interventions.

Using e-textiles affords different opportunities to observe teaching strategies to support iterative design because they (1) integrate CT within both programming (i.e., software design) and engineering (i.e., circuit design, physical craft) and can illustrate how teachers make connections between these contexts; (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 distinct student interests. Focusing on two classrooms from the *Exploring Computer Science* (ECS) program (Goode et al., 2014), we examined what strategies these experienced ECS teachers used in their implementation of the new e-textiles curriculum unit (Fields, Lui, & Kafai, 2017; Fields et al., 2018a, b). In this chapter we focus on strategies that support *iteration* as a key computational thinking process that can be difficult to implement in classrooms.

16.3 Methods

16.3.1 Context

Our e-textiles unit is embedded within the *Exploring Computer Science* (ECS) initiative, which comprises a one-year introductory computer science curriculum with a 2-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, 2011). 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 broaden the range of computer science activities by including e-textiles. The curriculum unit 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. A final version of the curriculum can be found at http://exploringcs.org/e-textiles.

The ECS e-textiles unit implemented for this study consisted 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). The projects were as follows: (1) a paper-card using a simple circuit, (2) a "stitch-card" with one LED sewn as a simple circuit, (3) a wristband with three LEDs in parallel, (4) a felt project using a preprogrammed LilyTiny microcontroller and 3–4 LEDs, (5) a classroom-wide mural project where pairs of students created portions that each incorporated two switches to computationally create four lighting patterns, and (6) a "human sensor" project that used two aluminum foil conductive patches that when squeezed generated a range of data to be used as conditions for lighting effects. Student artifacts included stuffed animals, paper cranes, and wearable shirts or hoodies, all augmented with the sensors and actuators.

In Spring 2016 two high school teachers, each with 8–12 years of computer science classroom teaching experience, piloted the e-textiles unit in their *ECS* classes in two large public secondary schools in a major city in the western United States. Both schools had socioeconomically disadvantaged students (59–89% of students at each school) with ethnically non-dominant populations (i.e., the majority of the students at each school include African American, Hispanic/Latino, or southeast Asian students).

16.3.2 Data Collection and Analysis

The study is part of a larger design-based implementation research study (Penuel, Fishman, Cheng, & Sabelli, 2011) where the goal is to develop and revise an e-textiles unit over the course of 3 years, attend to problems of practice in the classroom, develop better theories of pedagogy related to making and computing, and support classrooms in sustainable changes as they bring making to computer science. This paper reports on the first year of the study, where two teachers implemented the curriculum for the first time. Two researchers gathered data focused on teacher practice in the classroom, visiting each class equally, four days a week (about 8 weeks, with

www.dbooks.org

interruptions from holidays, testing, and other school obligations). The researchers documented teaching with detailed field notes, in-class video and audio recordings, and pictures/videos of student work, supplemented by three interviews with the teachers before, during, and after the unit, and brief focus group interviews with students at the end of the unit.

The analysis of field notes involved constant comparative analysis (see Charmaz, 2011) to (1) identify computational thinking practices exhibited during the e-textiles unit, and then (2) compare these with the larger corpus of computational thinking practices identified in the AP Computer Science Principles curriculum (see Fields et al., 2017). Through this process, iteration stood out a key area of learning. Then the team re-coded the data to find all of the teaching practices in this area. Finally, the team compared findings from observational data with the interviews from teachers and students to see whether these practices came up from participants' perspectives and to understand these two areas in greater depth.

16.4 Findings: Contexts for Iteration: Creating a Classroom Culture Valuing Mistakes and Revisions

Within the e-textiles unit, students engaged with iteration at many stages: prototyping, testing, and revising designs while tackling bugs and problems that arose in the process. This was evident through the changes that students made in their projects, including improvements in circuit diagrams, changes in and expansions of code, and visible alterations in the physical projects themselves. In fact, iterating on project ideas and implementation was one of the most frequent things we coded across the data. Earlier work in e-textiles has documented similar changes in student design (Fields, Kafai, & Searle, 2012; Kafai et al., 2014b). The focus of our findings here is on how the teachers supported a culture of iteration and refinement in their classrooms. What practices did they use to create an environment where sharing about mistakes and seeing them as a means of learning was valued? Furthermore, how did this culture support students' awareness and acknowledgement of iteration as a key perspective that could be used in service of creating a computational artifact? Below we outline three main areas of teaching practices that helped to develop a classroom culture of iteration: teachers' modeling of their own mistakes and teachers' modeling of students' mistakes, all with a focus on students' personal designs.

16.4.1 Teachers Modeling Their Own Mistakes

One key teaching practice involved teachers promoting *their own mistakes, errors, and less-than-perfect projects* in front of the classroom. When introducing a new project, for instance, Ben or Angela would show their own sample creations (which

were made during teacher professional development for the unit). This not only served to give students ideas but also allowed the teachers to showcase their own experience of revision and iteration, and coach students on tips for dealing with this process. Consider the way Angela shared her work in her introduction to the LilyTiny project in class, as highlighted within our field notes

> So let me tell you a little story. When I was working on one of my projects, I didn't think I needed a plan. "I'll just do this," I thought.' Angela went on to explain that she worked for two days on her project and then eventually had to take most of it apart because it didn't work. 'If you don't plan, it's going to take you more time to take things out and fix it than it would to do it right the first time. So you're going to draw it all out, and you'll use that blueprint and then you'll have your little map. (160405 field notes[1]).

Here Angela told a self-deprecating story of her process of creating her project. She highlighted how *not* having a predetermined plan for her e-textiles project—specifically, mapping out circuitry connections beforehand—meant that she ended up making numerous mistakes while crafting and eventually had to redo her entire project. She framed the circuit diagram as a way of creating a well-thought out plan for implementation, a skill which is helpful not only with regard to crafting but also within the context of creating a complex computational artifact that encompasses multiple modes (i.e., sewing, circuitry, programming). It serves as both a time-saving measure to prevent costly mistake fixing, but also to help students structure their construction time more efficiently. Note that the knowledge Angela shared was very pragmatic; she did not ask students to trust her on her authority alone (i.e., "have a plan because I said so") but rather because of her personal experience.

The other teacher, Ben, similarly highlighted his own process of dealing with mistakes when using his own project to aid in teaching. During a programming lesson, Ben shared his personal project code as an exemplar and sample for students to remix while using a preassembled e-textiles circuit board (the LilyPad Protosnap)

> Ben: As an introduction, "everyone please open up my.pdf called Function Code. And I want pairs to inspect the code and discuss what the code is going to do." Two students who were absent the prior day pointed to the switch and said that Number 2 should turn on… They seemed confused. At that moment, Ben realized his error—the code that he shared was something he wrote up for his own (extra) project that he is making that will involve switches and buzzers —his variables refer to the wrong ports on the Protosnap microcontroller. (160524 field notes).

Ben went on to explain this mistake to the class: "My apologies. I was messing around with the buzzer last night [on my own project]," and then explained how the connections were different from the LilyPad Protosnap boards students were using. However, rather than starting over on the task, Ben highlighted his error to students, inviting them to participate in how it could be fixed so that the code matched with the boards (160524 field notes).

[1]We use double quotation marks (" ") to show exact words of participations and single quotation marks (' ') to show paraphrased words that occur most often in field notes where conversations were typed in the moment rather than audio recorded and transcribed.

www.dbooks.org

In modeling their own imperfect processes of creation and addressing their mistakes, the teachers, Angela and Ben, thereby promoted a classroom culture of iterative practice, valuing process over product. While everyone was encouraged to make full, working projects, the teachers stressed that the actual experience of creation and learning would most likely include moments of failure, and subsequent revisions and iterations. Students were encouraged to think that it was okay not to be perfect the first time (or the second, third, fourth) they did something. Perfection, in these cases, could prevent students from moving forward in their learning.

16.4.2 Teachers Modeling Students' Mistakes

Just as the teachers showed their own projects and processes in front of the entire class, they also showcased students' challenges, mistakes, and in-process projects in order to promote the practice of iteration and revision. For instance, Angela added a journal question after the completion of the wristband activity that solicited challenges that students had faced: "Think about this week's project, what was the biggest challenge? If you had no challenges, what tips do you have for people who may be struggling?" (160422 field notes). After students had some minutes to think and to write, she invited various students to share out what they had written. Numerous students shared advice such as "plan more," echoing Angela's lesson from above regarding the importance of creating clear circuit diagrams as blueprints in help in constructing a functional e-textile artifact. Students also mentioned other issues that spanned across the multiple domains of e-textiles work. For instance, another student brought up the polarity issue of "mixing up the positive and negative" when trying to create a working sewn circuit. In response, Angela invited students to share suggestions on how to avoid this issue; they suggested curling or twisting the positive and negative sides of the LED wires to look different, developing symbolic means to identify polarity.

Another way that the teachers modeled students' mistakes was through reflections between projects. For instance, after the wristband (project #3) was complete, Ben provided some constructive thoughts to his class

Ben: 'I want to talk about the [project] we just did. Because those bracelets look awesome, they look fantastic and you should be proud of yourselves. There were a couple of things that I saw that you could improve … I saw some sloppy stitching. Meaning that they were more than an inch. Some of them were not pulled completely tight. I know that some of you might not want to do the work of going in and out and in and out. But what might be a concern?'

Student: 'It would get caught in something.'

Ben: 'Yes, and concise means a little tighter.' (160418, field notes)

In this example, Ben went on to coach students about two other problems he saw in student work on the wristbands. Each time, he presented a problem then asked

students why they thought it might be an issue, allowing the students to share their expertise on these problems. In this way Ben also supported iterative design between projects, not just within a project. This suggests that having a *series* of projects provided more opportunities for iterative practices than just having a single project. Students could improve their techniques across projects by recalling this ongoing catalogue of mistakes and solutions, and essentially acting as problem-solving resources for one another. Beyond merely supporting *iterative activity*, the teaching strategies described above pushed students to explicitly consider how *iterative thinking* was an essential part of computational work. By asking students to continually reflect upon the challenges and issues they faced, both Angela and Ben were able to highlight the inherently adaptive, trial and error nature of creating a functional e-textile artifact. Students learned not only how to recognize potential problems, but how to continually address these through more effective planning and ongoing implementation. From this standpoint, students became more cognizant of the powerful role that iteration could play in developing their own increasingly complex projects over time.

16.4.3 Supporting Personalized Design to Facilitate Iterative Practice

What inspired these practices of revealing and sharing mistakes, thus reinforcing process over product, and supporting iteration as an activity and a perspective? One thing that both teachers noted in their interviews with us was that doing the projects themselves helped them understand what students were going through. As Angela expressed, "A lot of times, we don't do what we're asking kids to do. It was great doing the projects in the [professional development sessions] because it helped me anticipate questions that might come up in class and think of ways to address them" (160609, interview). Creating projects helped the teachers themselves to reflect on their own processes of making them (including their own experience of trial and error, revision, and iteration) and provided a base of stories and examples to share with students. Thus one of the most important underlying aspects of the classroom culture was the teachers' *emphasis on their own as well as students' personalized projects*.

How did the teachers' promotion of personalized projects promote iterative thinking and design? Students were tasked to create projects within predetermined constraints throughout the unit (e.g., a light-up wristband, an interactive felt banner). However, both Angela and Ben also actively encouraged students to develop their own personal ideas through these assignments. This can be illustrated, in particular, through the students' final human sensor project, where a fabric object was modified to include four to five LEDs, which could be triggered by readings from conductive foil patches into at least four customized light pattern functions. From the start, the teachers actively encouraged personalization by allowing students to either bring or

www.dbooks.org

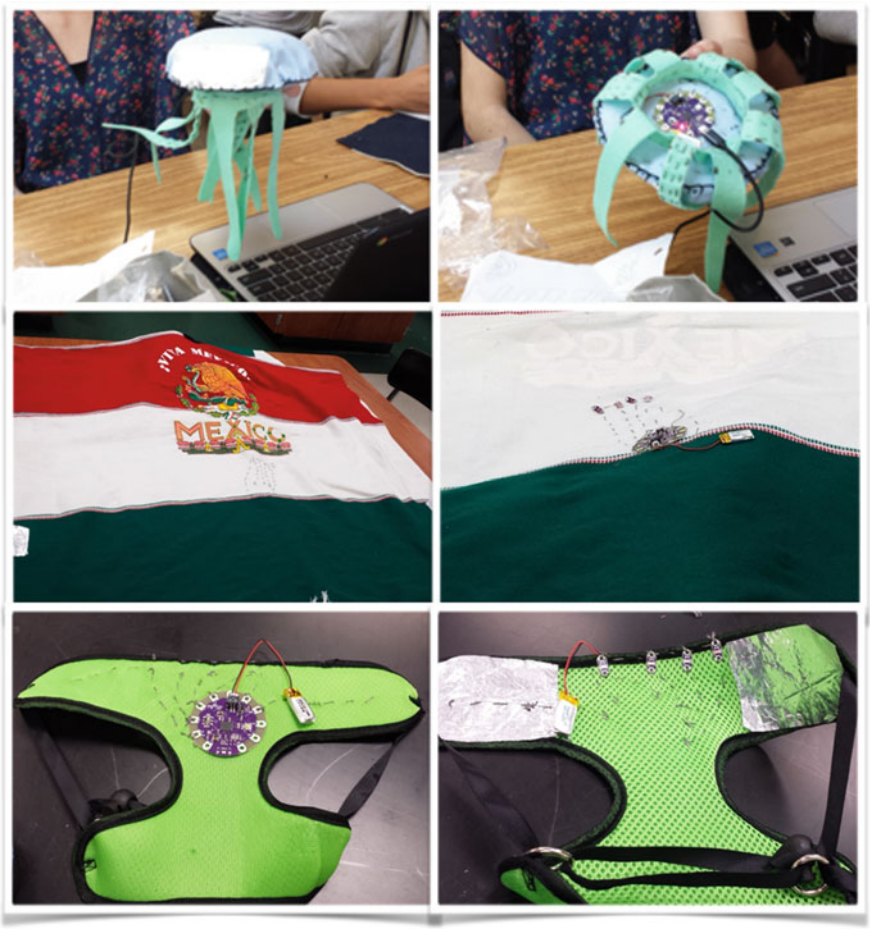


Fig. 16.1 Human sensor projects by students (top to bottom): Bridget's jellyfish (top and bottom views), Mateo's Viva Mexico poncho (front and back), and Peter's dog harness (top and bottom)

create their own personal objects for modification. This ranged from a "Viva Mexico" poncho, to a dog walking harness, to a stuffed jellyfish made of fabric (see Fig. 16.1). Further, teachers encouraged the students to customize the desired functionality based on their own interests and desires. For instance, some projects had a practical purpose (the walking harness that lit up when it was actually worn by a dog), while others' projects were more whimsical (the jellyfish, whose tentacles glowed when triggered through play and interaction).

By pushing this personalization within the context of the given project constraints, teachers inherently led students through an iterative process. One way this occurred was in developing unique circuit diagrams for the diverse objects. For instance, Angela had to assist students in thinking through the complex spatial dimensions

translating a two-dimensional blueprint to many different kinds of three-dimensional objects, for instance, a hat with an inside and outside surface or a stuffed animal with multiple overlapping surfaces. Another important context of iteration was in programming their objects. Students were all given basic "starter code" that included the functions for activating the conductive patches and a conditional statement with different outputs based on sensor readings. To match individual projects, students were required to make multiple changes. Ben, for instance, worked with his entire class to test how the different conductive patch sizes and users would shift the reading ranges that would need to be included within students' different programs.

Notably, this emphasis on original design ensured feelings of personal ownership over student projects, which in turn led to a willingness to persevere through mistakes and bugs. Because each project was distinct, students dealt with unique sets of challenges and issues. Troubleshooting therefore became a process of knowing how to isolate different problems, iteratively moving through different potential causes and solutions. While some students did become discouraged by this process, for others, the personal commitment to the project led them to push through the process, even beyond the teachers themselves, as demonstrated by the example from an observation below

> Ethan's project was finished but a glitch in the code caused two of the LEDs not to turn on in two of the four lighting pattern functions. The teacher and the researcher worked to help Ethan troubleshoot the project; Ethan expressed great frustration. Two other students came to try to assist. One said that she had a problem with delays in her code, so Ethan tried many different types of code fixes (added delays, copied and pasted code into a new function, re-typed the problematic function). No one could figure out the underlying problem. Unbothered at this point, Ethan finished by rewriting the code for the two problematic functions, designing simpler lighting functions, and it worked (160531, field notes).

From this standpoint, encouraging personalization not only creates natural opportunities for iterative design and troubleshooting, but also has the potential of motivating students to continually engage with and push through these processes. Considering that incremental and iterative work is so essential to computational thinking, the personalization promoted by the teachers and afforded by the medium of e-textiles therefore further enhances iteration.

16.5 Discussion

Our paper contributes to the emerging body of research on teaching practices of computational thinking that focused not just on programming but also on physical crafting and electronics. In our analysis we focused on a key aspect of computational thinking—iterative practices—that teachers addressed within the electronic textiles curriculum unit. The teachers illustrated how to support CT with personalized project creation involving large numbers of students (24 and 35), the restricted time constraints of class periods, and the temporary nature of classroom-based makerspaces where materials had to be put away every day. In the following sections, we dis-

www.dbooks.org

cuss further aspects of teaching computational thinking that we saw instantiated in our study with the goal to make teaching CT more culturally relevant and equitable (Goode et al., 2014).

First, we draw attention to *how* teachers engaged with iterative practices: by not only modeling publicly their own mistakes but also those of their students so that others could learn from them. These combined practices created a classroom culture where process—improving, fixing, iterating on designs—was situated in the context of students' personal projects addressing their individual concerns while also sharing them with others. This approach to teaching computational thinking made iterative practices public and part of the larger classroom community while also distributing responsibilities. Through this approach teachers validated "process" alongside "product", highlighting iteration not only as an activity, but as a perspective which can help organize students' engagement with computational problems. In more recent efforts we are working to further enhance this approach by having students report and reflect on their mistakes in portfolios (see Lui, Jayathirtha, Fields, Shaw, & Kafai, 2018; Lui et al., in press; Fields, Shaw, & Kafai, 2018). Our plan is to have students document the challenges and revisions that come up in their iterative design processes by taking pictures of mistakes, marking errors in versions of code, or showing changes in the development of their circuit diagrams. This will add a formal element of reflection on iterative design to the pedagogy that the teachers have already implemented in their classrooms.

Second, we point out that by making computational thinking public and part of the larger classroom culture, the teachers also addressed another critical element relevant to supporting equitable and inquiry-based teaching: creating an audience for e-textiles projects that highlights their usability. For instance, 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 well as their teachers' projects could have relevance outside the classroom. These features are important because audience and participation are key areas of supporting design though they have only recently been recognized as relevant in the area of computational thinking by highlighting computational participation (Kafai & Burke, 2014). Other valuable strategies such as validating student expertise and supporting personal designs during class discussion to engage students with computational thinking have been addressed elsewhere (see Fields et al., 2018a, b).

By themselves the strategies discussed above are nothing new in having been identified 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 develop this emerging field of pedagogy in computer science education. We see them as an example of computational pedagogical content knowledge, or the unique knowledge that teachers need to develop in order to embed computational thinking in their instructional practice to support student learning. This chapter provided salient classroom examples of what teaching iterative practices in the context of personal student projects can look like.

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.

Brennan, K., & Resnick, M. (2012, April). *New frameworks for studying and assessing the development of computational thinking*. Annual Meeting of the American Educational Research Association, Vancouver, BC, Canada.

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.). (2013a). *Textile messages: Dispatches from the world of e-textiles and education*. New York, NY: Peter Lang.

Charmaz, K. (2011). Grounded theory methods in social justice research. *The Sage Handbook of Qualitative Research, 4,* 359–380.

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.

Fields, D. A., Kafai, Y. B., Nakajima, T. M., Goode, J., & Margolis, J. (2018a). Putting making into high school computers science classrooms: Promoting equity in teaching and learning with electronic textiles in Exploring Computer Science. *Equity, Excellence, and Education, 51*(1), 21–35.

Fields, D. A., Shaw, M. S., & Kafai, Y. B. (2018b). Personal learning journeys: Reflective portfolios as "objects-to-learn-with" in an e-textiles high school class In V. Dagiene & E. Jastuė (Eds.), *Constructionism 2018: Constructionism, Computational Thinking and Educational Innovation: Conference Proceedings* (pp. 213–223). Vilnius, Lithuania. http://www.constructionism2018.fsf.vu.lt/proceedings.

Fields, D. A., Kafai, Y. B., & Searle, K. A. (2012). Functional aesthetics for learning: Creative tensions in youth e-textiles designs. In J. van Aalst, K. Thompson, M. J. Jacobson, & P. Reimann (Eds.), *The Future of Learning: Proceedings of the 10th International Conference of the Learning Sciences (ICLS 2012), Full Papers* (Vol. 1, pp. 196–203). Sydney, NSW, Australia: International Society of the Learning Sciences.

Fields, D. A., Lui, D., & Kafai, Y. B. (2017). Teaching computational thinking with electronic textiles: High school teachers' contextualizing strategies in *Exploring Computer Science*. In S. C. Kong, J. Sheldon, & R. K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education 2017* (pp. 67–72). Hong Kong: The Education University of Hong Kong.

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.

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.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

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.

Guzdial, M. (2016). Drumming up support for AP CS principles. *Communications of the ACM, 59*(2), 12–13.

www.dbooks.org

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. (2014a). 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. (2014b). A crafts-oriented approach to computing in high school. *ACM Transactions of Computing Education, 14*(1), 1–20.

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.

Lui, D., Jayathirtha, G., Fields, D. A., Shaw, M., & Kafai, Y. B. (2018). Design considerations for capturing computational thinking practices in high school students' electronic textile portfolios. In *Proceedings of the International Conference of the Learning Sciences.* London, UK.

Lui, D., Walker, J. T., Hanna, S., Kafai, Y. B., Fields, D. A., & Jayathirtha, G. (in press). Communicating computational concepts and practices within high school students' portfolios of making electronic textiles. *Interactive Learning Environments.*

Margolis, J., & Goode, J. (2016). Ten lessons for CS for all. *ACM Inroads, 7*(4), 58–66.

Margolis, J., Goode, J., & Ryoo, J. (2015). Democratizing computer science knowledge. *Educational Leadership, 72*(4), 48–53.

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.

Penuel, W. R., Fishman, B. J., Cheng, B., & Sabelli, N. (2011). Organizing research and development at the intersection of learning, implementation, and design. *Educational Researcher, 40*(7), 331–337.

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.

Soloway, E., & Spohrer, J. C. (Eds.). (1989). *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum.

Tayal, S. P. (2013). Engineering design process. *International Journal of Computer Science and Communication Engineering*, 1–5.

Tofel-Grehl, C., Fields, D. A., Searle, K., Maahs-Fladung, C., Feldon, D., Gu, G., & Sun, V. (2017). Electrifying engagement in middle school science class: Improving student interest through e-textiles. *Journal of Science Education and Technology, 26*(4), 406–417.

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). 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), 1–16.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 17
A Study of the Readiness of Implementing Computational Thinking in Compulsory Education in Taiwan



Ting-Chia Hsu

Abstract In recent years, Computational Thinking (CT) Education for K-12 students and undergraduates has become an important and hotly discussed issue. In Taiwan, starting from August 2019, all students in secondary schools will be required to be fostered with computational thinking competencies. This chapter discusses not only the preparation of students to learn CT but also the preparation required for teachers and principals. There will be six credits each for the compulsory education of information technology and living technology in junior high schools. Integrating CT into other courses, such as mathematics, is one of the approaches implemented at the primary school level. This chapter explores an initiative in which teachers integrated block-based programming into a mathematics course for the sixth-grade students, and further studied the self-efficacies and motivations of the students while they learn CT in the integrated course. The chapter also reports on investigations of the attitudes of educational leaders, such as the K-12 principals, towards teacher preparation for conducting CT education in their schools. The results of the study indicate that the weakest part of the object readiness (facilities) in 2017 in Taiwan was the availability of classrooms for maker activities from the perspectives of the K-12 principals. In terms of human resource readiness, instructional material resource readiness, and leadership support (management readiness), teachers and principals scored readiness degree at more than three points but less than four points on a five-point Likert scale, implying that there is still room in all these aspects to be enhanced. Many teacher training courses will need to be carried out in the next 1 to 2 years because the technological and pedagogical content knowledge of the teachers regarding CT education must continue to be strengthened.

Keywords Computational thinking · Self-efficacy · Learning motivation · Teacher education

T.-C. Hsu (✉)
National Taiwan Normal University, Taipei, Taiwan
e-mail: ckhsu@ntnu.edu.tw

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_17

17.1 Introduction

Computational thinking refers to the basic concepts and processes used for solving problems in the computer science domain. The term was officially proposed in 2006 (Wing, 2006), and was later simplified into four phases for the curriculum design of CT in the United States (e.g., https://code.org/curriculum/course3/1/Teacher; http://cspathshala.org/2017/10/25/computational-thinking-curriculum/), the United Kingdom (e.g., https://www.bbc.co.uk/education/guides/zp92mp3/revision), India (e.g., https://www.nextgurukul.in/KnowledgeWorld/computer-masti/what-is-computational-thinking/), and so on (Fig. 17.1).

As shown in Fig. 17.1, the first phase of the CT process is to decompose the problem so that it can be analyzed and divided into several smaller subproblems. This is called the "problem decomposition" phase. 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 methods, they can identify their similarities, regularities, or commonalities. Therefore, they do not need to spend time repeating work when they write out the solution steps. The third phase is to generalize or abstract the principles or factors to become a formula or rule. The students have to try to model the patterns they found in the previous step. After testing, they

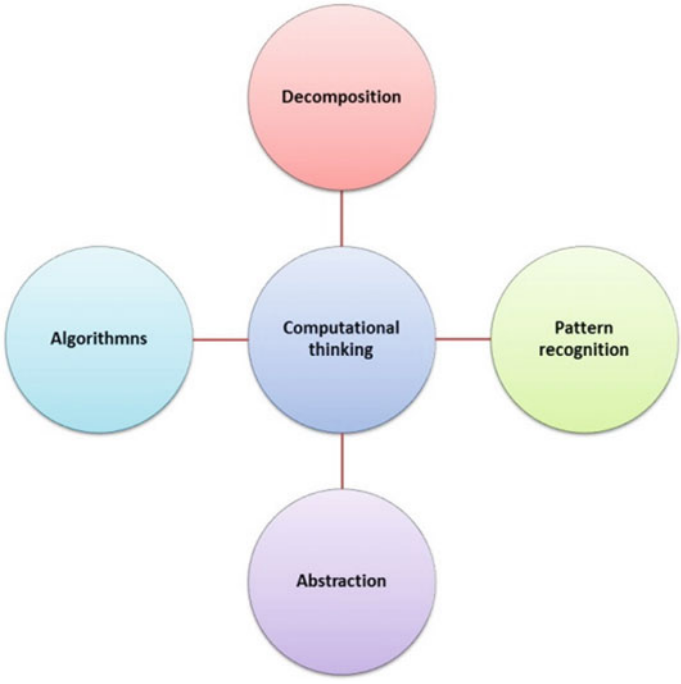


Fig. 17.1 CT process (cited from the BBC, UK)

www.dbooks.org

identify and abstract the key or critical factors presenting the model for solving the problem 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 CT is not equal to programming, block-based programming languages such as Scratch, Blockly, mBlock, App Inventor, and so on, are good tools for developing the capabilities of students' CT. CT has been defined 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). The current study not only employed Scratch to learn CT, but also used it to implement the solution to a problem that the students encountered in their mathematics course. Scratch or other visual programming tools are suitable to be used in different contexts such as games, science, music, and so on (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Armoni, Meerbaum-Salant, & Ben-Ari, 2015).

In a study by Maloney (2008), when Scratch was introduced to young students from 8 to 18 years old, the students were found to be highly motivated to write programs. Another study found that fifth and sixth graders perceived Scratch as being useful, and that they had high motivation and positive attitudes toward using it (Sáez-López, Román-González, & Vázquez-Cano, 2016). Ke (2014) applied Scratch for secondary school students to design mathematics games, and found that the integration of block-based programming and Mathematics game design could promote the potential of the students to learn Mathematics, and resulted in students' having significantly more positive attitudes toward the development of Mathematics. Furthermore, this method was beneficial for activating students' reflection on their daily-life mathematical experiences. The mathematics concepts and block-based programming were integrated when the students solved the problems or created the games. They not only took part in achieving the mathematics learning target, but also carried out CT, and transferred the reasoning process into an abstract program. It has been found that using block-based programming in computer science can promote the cognitive level and self-efficacy of students, but it does not result in high learning anxiety, and the students spend less time learning and creating new programs in comparison with line-based programming (Armoni et al., 2015).

The first study reported in this chapter integrated the block-based programming software, Scratch, into a mathematics course, and applied the four phases of CT 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 block-based programming. Therefore, the research questions are as follows:

(1) Was the students' learning effectiveness of mathematics significantly promoted after the treatment?

(2) Was there a significant correlation between the performances of block-based 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 treatment?

Apart from exploring the effectiveness of such CT courses for K-12 students, the degree of preparedness of the teachers in teaching CT is another important issue. When CT becomes a necessary form of literacy all around the world, it will not only be a kind of expertise that, stereotypically, only computer engineers use. On the contrary, everyone should have positive attitudes toward CT in order to understand and make use of it (Wing, 2006).

Based on a survey of 17 European countries, in a previous study (Balanskat & Engelhardt, 2014), it was found that most of the countries have tried to integrate CT courses into their K-12 curricula. In addition, elementary and secondary schools in Australia have introduced CT into courses for a period of time, and have placed CT literacy in the national education curricula (Falkner, Vivian, & Falkner, 2014). Therefore, many teachers are now trying to integrate CT into various courses (Heintz, Mannila, & Färnqvist, 2016). With the current development of digital technologies and the concerns about CT literacy, how teacher education should prepare teachers to teach CT is an important question to be studied.

Recently, Orvalho (2017) indicated that teachers should follow the methodology for pre-service teachers: Before teaching students how to do CT, the teachers themselves should first acquire the knowledge and abilities related to CT. Yadav also pointed out that introducing computer science into courses for pre-service teachers can efficiently enhance teachers' understanding of CT. Therefore, the teacher can not only learn how to incorporate CT in their courses, but can also help the students cultivate their problem-solving capabilities (Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). Mouza applied TPACK (i.e., Technology, Pedagogy, and Content Knowledge) instructional method to CT education, by having teachers designed CT courses associated with K-8 education during their teacher education. The results showed that the pre-service training not only had a positive influence on teachers, but could also help them to develop and practice instructional content embedded in CT (Mouza, Yang, Pan, Ozden, & Pollock, 2017).

As CT is applied to the training of not only teachers but also principals, they will all know what CT is, and how to integrate it into their courses, as well as the requirements of those courses. Many counties in Taiwan have asked newly appointed principals to enroll in training courses related to technology and leadership since 2011. It is expected that the principals know the requirements of the facilities and faculty in their schools for carrying out technology-related instruction, administration, and service. In the recent 2 years, they were made aware of the associated issues related to CT. Israel, Pearson, Tapia, Wherfel, and Reese (2015) applied CT to teacher education to overcome obstacles for teachers to achieve expertise in an Introduction to Computer Science course. K-12 faculty would realize what difficulties the students with deficient resources may encounter. Through the teacher education for pre-service teachers or newly appointed principals, they would benefit greatly and could know

www.dbooks.org

how to provide support and assistance to their teachers for enhancing CT education (Israel, Pearson, Tapia, Wherfel, & Reese, 2015).

In addition, when it comes to CT education, visual programming is a critical enabler. When the teachers design CT-related courses, they mostly use block-based programming tools for the basic level. Cetin (2016) considered CT to be the foundation, and applied Scratch to pre-service teachers' training. The results indicated that this did indeed help the teachers in arranging beginner courses, and the visual programming environment could help teachers better understand CT (Cetin, 2016).

The second study reported in this chapter applied the same approach in study one (i.e., visual programming for the mathematical learning unit) in the teacher training for newly appointed K-12 principals. After they experienced the demonstrations and training, we then investigated the readiness of their schools according to four dimensions: technology readiness, teacher readiness, instructional resource readiness, and leadership support. We also investigated the technology, pedagogy, and content knowledge (TPACK) and the overall TPACK related to CT education based on the real conditions the principals perceived. Therefore, the research questions for the second study are as follows:

(4) Concerning the principals who had experienced this course during their professional development training, how did they perceive the present readiness of their school for conducting such CT courses?
(5) Concerning the principals who had experienced this course during their professional development training, how did they perceive the present TPACK of the teachers in their school?

Overall, study one aimed to confirm the feasibility and effectiveness of conducting CT education in K-12 courses. Study two explored the readiness of the leadership in K-12 schools to implement and support CT education.

17.2 Method of Study One

As mentioned above, two studies are integrated into this chapter. The following section illustrates the research method including participant samples, measuring tools, and the experimental process, as well as the research results for study one.

17.2.1 Participants

For research questions one to three in study one, 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 10 years. The average age of the students was 12.

17.2.2 Measuring Tools

The learning performance of CT includes three aspects which are concepts, perspectives, and practices (Brennan & Resnick, 2012). In study one, the research tools 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 mathematics test sheets were developed by two experienced teachers. The pre-test consisted of 10 calculation questions about the prior knowledge of the course unit "equality axiom," with a perfect score of 100. The post-test consisted of 10 calculation questions for assessing the students' knowledge of the equality axiom unit, with a perfect score of 100. For instance, the following is an example for the elementary school students to practice mathematics and programming at the same time.

> On Sandy's birthday, her father, mother, and brother go to a theme park with her. They participate in a competition in which they have to guess the size of the facilities, which constitute a triangle. The host asks them to estimate the area of the triangle to get points by applying the block tools on a computer. If you were Sandy, how would you solve the problem using block-based programming to make automatic calculations?

Table 17.1 shows the answer of one student for the abovementioned problem to reveal an example of block-based programming and the CT process.

In the post-test of programming performance, there was a total of five situated problems for the students to solve using block-based programming according to the four phases of CT. Each programming problem was scored as 20 points, including five points for assessing whether the students employed proper blocks, five points for checking the usage of variances, five points for evaluating the formula transferred from the meaning of the problem by the students in the program, and five points for confirming if the output was correct or not. Consequently, the 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). It consisted of seven items (e.g., "It is

Table 17.1 Demonstration of the block-based programming and CT process

Example of block-based programming	Algorithmic thinking	CT process
	Starting the sequential steps	Define problem
		Pattern recognition
	Read the base from input	
	Read the height from the input	
	Use a formula to calculate	Abstraction
	Output the area of the triangle	

www.dbooks.org

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). It consists of eight items (e.g., "I'm confident I can understand the basic concepts taught in this course") with a five-point Likert rating scheme. The Cronbach's alpha value was 0.894.

The Creativity Assessment Packet (CAP) was revised from Williams (1991), and included the scales of imagination, curiosity, and so on. It consisted of 50 items (e.g., "I have a vivid imagination") with a 5-point rating scheme for the scales of overall creativity, curiosity, imagination, complexity, and risk taking.

17.2.3 Experimental Procedure

Before the experiment, the students were given time to get used to the block-based programming environment. Figure 17.2 shows the flow chart of the experiment. Each period in the mathematics class is 40 min in elementary school. At the beginning, the instructor spent 8 weeks (i.e., one period a week, and totally eight periods) teaching the students to become familiar with the block-based programming environment.

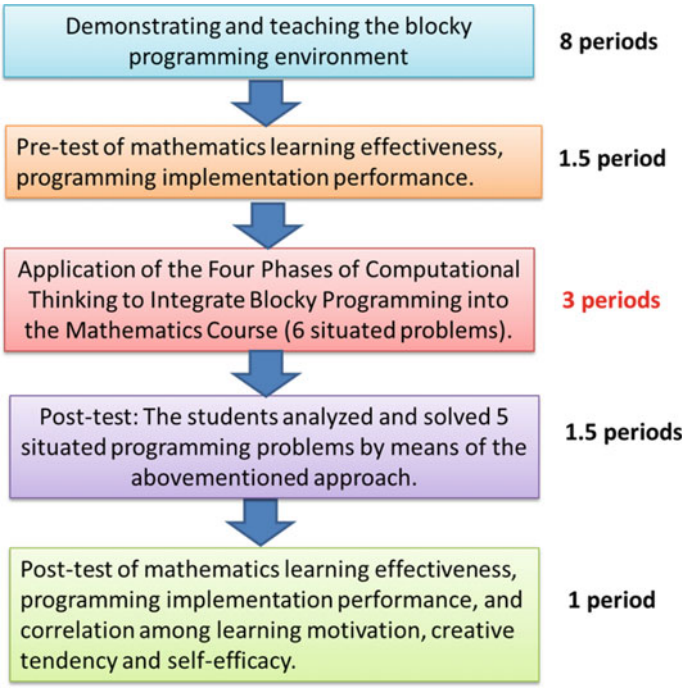


Fig. 17.2 Experimental procedure in study one

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

Thereafter, 3 weeks (i.e., one period per week, and totally three periods) was spent on the enhancement of applying the four phases of CT and the integration of block-based programming in a sixth-grade Mathematics course. After the effectiveness of involving the CT process with mathematics was confirmed in study one, this part (three periods) was later demonstrated in the teacher training course for the newly appointed principals to experience and observe the common sense of involving CT processes in learning. The students practiced this method six times, each time taking half 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 block-based programming problems through the four CT phases. After the learning activity, there were totally five programming problems for evaluating the students' block-based programming performance, with the four CT phases involved in both the block-based 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. There were totally 15 periods spent on the experiment, which lasted for a total of around three-fourth of a semester (i.e., 15 weeks). The experimental treatment after the pre-test was 5 weeks.

17.2.4 Data Analysis

In study one, the pre- and post-test were compared via a paired-sample *t* test to assess whether or not the students had made progress in the mathematics unit.

Their block-based programming performance was also assessed. Correlation analysis was performed to identify the relationship between the students' block-based 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 after the students learned mathematics from the application of the four phases of CT to integrate visual programming into the mathematics course.

Based on the above-mentioned data analysis methods, study one in this chapter reports the cognition and perspectives of the students involving CT processes in their mathematical learning.

www.dbooks.org

Table 17.2 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	

*$p < 0.05$

Table 17.3 Correlation between programming performance and the post-test (N = 20)

Pearson correlation coefficient	Block-based programming	Post-test
Block-based programming performance	1	0.673***
Post-test	0.673***	1

***$p < 0.001$

17.3 Results of Study One

17.3.1 Paired-Sample t-Test Analysis of the Mathematics Pre- and Post-test

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 CT were applied to integrate block-based programming into the mathematics course. Table 17.2 reveals that this approach did indeed contribute to the students' learning effectiveness. They made significant progress in the mathematics equality axiom unit after the experimental treatment ($t = 2.72$, $p < 0.05$).

17.3.2 The Correlation Between the Block-Based Programming Performance and the Mathematics Post-test

In this study, we attempted to verify the correlation between the performance of block-based 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 17.3. When the students had better performance on applying the four phases of CT 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.

Table 17.4 Correlations between self-efficacy, 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

**$p < 0.01$

17.3.3 Correlations Between Students' Self-efficacy, Creative Tendency, and Learning Motivation

The self-efficacy of the students applying the four phases of CT to integrate block-based 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, $p > 0.05$), as shown in Table 17.4. In sum, the learning motivation was positively correlated with the students' self-efficacy regarding CT processes in their learning.

17.4 Method of Study Two

As mentioned above, two studies are integrated into this chapter. The following sections illustrate the participant samples, measuring tools, and the experimental process for study two.

17.4.1 Participants

For study two, there were 24 newly appointed principals who participated in the teacher training course. They were taught by the same instructor as study one in the teacher training workshop.

17.4.2 Measuring Tools

In study two, the participants had to answer the questionnaire of readiness for CT education at their school, and express the situation they perceived in the TPACK (i.e., technological, pedagogical, and content knowledge) of their teachers. There are eight

www.dbooks.org

scales in the questionnaire. The first four were revised from the readiness questionnaire of mobile learning (Yu, Liu, & Huang, 2016) which referred to an eclectic e-learning readiness including object readiness, software readiness, and leadership support (Darab & Montazer, 2011), and referred to the higher education m-learning readiness model based on the theory of planned behavior (TPB; Cheon, Lee, Crooks, & Song, 2012). Accordingly, object readiness, instructor readiness, instructional resource readiness, and leadership support are important scales for evaluating the readiness for putting something into practice at school, such as e-learning, mobile learning, or CT. Therefore, this study employed the readiness questionnaire, and the Cronbach's reliability coefficient for each scale in the revised questionnaire was .701 for object readiness, .673 for instructor readiness, .646 for instructional resource readiness, and .835 for leadership support.

The relationship between teachers' technological, pedagogical, and content knowledge (TPACK) is clearly pointed out in the framework of the TPACK model (Mishra & Koehler, 2006). Numerous studies have therefore adopted this model to assess teachers' professionality or the effectiveness of teacher education (Chai, Koh, & Tsai, 2010; Koehler, Mishra, & Yahya, 2007). This model has also been introduced in another study for teachers to perform self-assessment (Schmidt et al., 2009). The current study also employed the TPACK model (Chai et al., 2010) for the principals to describe the school teachers in the technology domain. The Cronbach's reliability coefficient for each scale in the revised questionnaire was .840 for the knowledge of technology, .884 for the knowledge of pedagogy, .943 for the knowledge of content, and .908 for the overall TPACK.

17.4.3 Experimental Procedure

After study one, the same instructor taught the CT course in the teacher training workshop for the newly appointed principals. The teacher training workshop consisted of 18 h. There were 9 h spent experiencing the CT process integrated with the mathematics unit through the tool of visual programming. During the remaining 9 h, they had to visit a school or institute where the infrastructure has been well established, and attend the training course introducing the requirements for conducting the 12-year compulsory education in the technology domain.

After they experienced the CT course and completed the teacher training, the principals filled out the questionnaires to assess their schools. One questionnaire was revised from the readiness for mobile learning, and the other one was the TPACK (i.e., Technological, Pedagogical, and Content Knowledge) model.

Table 17.5 Descriptive information for the first four scales: readiness

Scale name	Description	Sample item
Object readiness	For the current situation of equipment in the school, please answer the following questions	There is enough information equipment such as computers for learning in the school, providing resources for technological courses
Human resources readiness (instructor readiness)	For the current situation of teachers in your school, please answer the following questions	There are full-time information technology teachers in my school
Instructional resource readiness	For the arrangement of teaching materials for the technology domain, please answer the following questions	The teachers in my school have the capabilities to employ the official textbooks in the information technology courses
Leadership support (management readiness)	For the attitude of school management, please answer the following questions	School management proposes visions, policies, or plans that support and encourage the teaching as well as learning in the technological domain

17.4.4 Data Analysis

In study two, the descriptive information for the first four scales of readiness is shown in Table 17.5, which is abstracted from the first questionnaire (i.e., readiness).

The reliability data suggest that the refined version of each scale for readiness and TPACK has acceptable internal consistency. The investigation results show the descriptive statistics for each item, including the mean scores and the standard deviation.

17.5 Results of Study Two

17.5.1 The Present Readiness for CT Education as Perceived by the Principals

From the investigation results in Table 17.6, the average scores of object readiness are quite low. From each item shown in the questionnaire in Appendix 17.1, it could be found that the information equipment such as computers for learning in the school has been available for a period of time (Mean = 4.17, SD = 0.87). Therefore, the prin-

www.dbooks.org

Table 17.6 School readiness for conducting compulsory education in the technology domain for each dimension (N = 24)

Scale	Mean	SD
Object readiness	3.00	0.82
Human resources readiness (instructor readiness)	3.28	0.93
Instructional material resource readiness	3.18	0.80
Leadership support (management readiness)	3.77	0.74

cipals expressed higher scores for the computer hardware in the first item. However, if the instruction requires equipment for hands-on activities, the maker classrooms are relatively lacking at the present time (Mean = 1.88; SD = 1.36). This is the main reason why the object readiness was reduced. In sum, the overall technology hardware and software for conducting compulsory education in the technology domain has not yet been well prepared as it is still 2 years before compulsory education in the technology domain begins.

As for instructor readiness, there are not enough full-time faculty in the technology domain according to the results of the human resources readiness scale, as shown in Table 17.6. The teacher education institutes must speed up the cultivation of new teachers, and the K-12 schools should open recruitment for teachers in the technology domain as their top priority.

In terms of instructional material resource readiness, the teachers tended not to take part in the teaching plan competitions. Therefore, holding teaching plan contests may not be the best strategy to produce adequate instructional material in the technology domain.

Finally, the leadership support was also taken into consideration for the readiness of conducting compulsory education in the technology domain. The scale of leadership support has the highest mean score among the four scales (Mean = 3.77; SD = 0.74). There is a strong tendency for the school leaders to put greater emphasis on students participating in DIY activities. In other words, school leadership tends to accept that teachers can design problem-solving tasks integrating different disciplines and hands-on activities in the future.

17.5.2 The Teachers' TPACK for CT Education as Perceived by the Principals

From the survey of TPACK for CT teachers, as shown in Table 17.7, it was found that the teachers are partially prepared at present (see also Appendix 17.2). The expertise of the present teachers was acceptable in terms of their knowledge of technology from the principals' point of view. However, the pedagogy of CT still has room for improvement. From the investigation results, the teacher education institutes have to put more effort into pedagogical research and training.

Table 17.7 TPACK for CT teachers for each scale (N = 24)

Scales	Mean	SD
Knowledge of technology	3.95	0.69
Knowledge of pedagogy	3.83	0.87
Knowledge of content	3.74	0.99
TPACK	3.58	0.97

17.6 Discussion and Conclusion

Study one not only put the four phases of CT into practice, but also applied them to solve mathematics problems with the block-based programming language. The results indicate that the implementation of programming activities 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 programming implementation had a significantly positive correlation with the learning effectiveness of mathematics, implying that the students who had better block-based programming scores outperformed the other students in the mathematics concepts post-test.

In addition to the CT concepts and practices, the perspectives of the students were also assessed. Few studies have explored the relationships among self-efficacy, learning motivation, and creative tendency. The results of study one found that 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. This result was similar to that of a previous study which pointed out that information literacy self-efficacy is associated with both intrinsic and extrinsic motivation (Ross, Perkins, & Bodey, 2016). Another study indicated that the creative tendency, such as curiosity as well as imagination, and domain-specific knowledge are critical for students' creative science problem-finding ability (Liu, Hu, Adey, Cheng, & Zhang, 2013). An earlier study reported that self-efficacy was closely related to creativity with intrinsic motivation completely mediating this relationship (Prabhu, Sutton, & Sauser, 2008). From study one, it has been confirmed that the students could learn CT and mathematics at the same time. In future studies, to promote the motivations of the students, the teachers could try to ask the students to design mathematics game programs with the block-based programming tools so that the students can also learn CT and mathematics at the same time. Future studies could also integrate CT into different subjects so that students can learn CT, programming, and subject knowledge (e.g., physics, mathematics) at the same time.

www.dbooks.org

After confirming the feasibility and benefits of conducting CT in study one, study two further explored the readiness of the K-12 schools. Based on the results of this investigation on the K-12 principals in study two, some suggestions to enhance the preparation for involving CT education in the 12-year compulsory education are given as follows.

It appears that object readiness, which refers to the educational hardware of the technology domain at school, is the easiest part if the government is willing to devote sufficient resources to the K-12 schools. However, the teachers have to be trained so they know how to operate the new equipment, regardless of whether it is the maker environment or computer technology products; otherwise, the money spent on the hardware will be wasted. The perfect environment which is expected to be constructed within the next 2 years will not work without professional teachers. Therefore, future studies could further analyze the regression between the readiness of the teachers in the technology domain and the readiness of the hardware, and find direct evidence for this inference.

Unfortunately, the participants perceived that the leadership and management levels have not provided enough support for conducting CT education. In other words, many people agree that CT education is important; nevertheless, people in leadership roles have not put enough emphasis on it. We inferred that the reason for this unexpected situation is that the literacy of CT is not included as part of the senior high school or college entrance examinations. It is important that schools should not just pay attention to the subjects related to senior high school or college entrance examinations; liberal education should also be encouraged.

Study one was conducted in 2016, and study two was carried out in 2017, while the compulsory education of CT will be put into practice in August 2019. Accordingly, in the next 2 years, the related institutes have a large amount of work to do. The report of this chapter provides some findings, references, and suggestions for both K-12 school faculty and the Ministry of Education. The most important aspect is teacher education. The teachers in the technology domain should be trained in the requirements of instruction in the technology domain.

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 and MOST 107-2511-H-003-031.

Appendix 17.1

School readiness for conducting compulsory education in the technology domain (N = 24)

Scale	Item	Mean	SD
Object readiness	1. There is enough information equipment such as computers for learning in the school, providing resources for technological courses	4.17	0.87
	2. Living Technology classrooms are set up, enabling complete living technology tutoring activities	2.46	1.38
	3. Maker classrooms are set up, giving students a proper environment to do maker activities	1.88	1.36
	4. There is availability of managers familiar with hardware and software, either full-time or outsourcing service	3.22	1.38
	5. I have confidence that the infrastructure in my school could put the 12-year compulsory education for the technological domain into practice	3.33	0.92
Human resources readiness (instructor readiness)	6. There are full-time information technology teachers in my school	3.29	1.71
	7. There are full-time living technology teachers in my school	2.54	1.56
	8. Our school provides workshops for teachers in the domain of technology to enhance their specialty	3.67	1.24
	9. The second specialty classes are available for other specialist teachers who want to be teachers in the technology domain	3.08	1.44
	10. The school is willing to arrange intramural and interschool activities for technology teachers to improve their capability	3.79	0.98
Instructional material resource readiness	11. The teachers in my school have the capabilities to employ the official textbooks in the information technology courses	3.29	1.33
	12. The teachers in my school have the capabilities to employ the official textbooks in the living technology courses	3.17	1.30
	13. The teachers in my school can develop instructional materials for school-based curricula on their own	3.17	1.17
	14. The teachers in my school are willing to take part in contests of making teaching plans for curricula	2.96	1.27

(continued)

www.dbooks.org

(continued)

Scale	Item	Mean	SD
	15. Currently, the school teachers do not have to worry about the teaching materials for the technology domain	3.29	1.08
Leadership support (management readiness)	16. School management proposes visions, policies, or plans that support and encourage the teaching as well as learning in the technological domain	3.71	0.81
	17. The school has established a reward system for those who have outstanding teaching performance in technology	3.58	0.88
	18. School management is gradually putting greater emphasis on students participating in DIY activities and contests	3.92	0.83
	19. School management will encourage teachers and students to engage in a robot or programming competition if there is one	3.79	1.10
	20. School management will encourage teachers and students to engage in a living technology contest if there is one	3.83	1.09

Appendix 17.2

TPACK for Computational thinking teachers (N = 24)

Scales	Questionnaire items	Mean	SD
Knowledge of technology	TK1-Our teachers know how to solve their own technical problems	4.13	0.90
	TK2-Our teachers can learn new technology easily	4.04	0.81
	TK3-Our teachers have the technical skills and use the technologies appropriately	3.92	0.78
	TK4-Our teachers are able to use computational thinking tools or software to do problem-solving	3.71	0.86

(continued)

(continued)

Scales	Questionnaire items	Mean	SD
Knowledge of pedagogy	PK1-Our teachers can adapt their teaching style to different learners	3.83	1.09
	PK2-Our teachers can adapt their teaching based upon what students currently do or do not understand	3.79	1.02
	PK3-Our teachers can use a wide range of teaching approaches in a classroom setting (collaborative learning, direct instruction, inquiry learning, problem/project based learning, etc.)	3.75	1.03
	PK4-Our teachers know how to assess student performance in a classroom	3.96	0.91
Knowledge of content	CK1-Our teachers have various ways and strategies of developing their understanding of computational thinking	3.67	1.05
	CK2-Our teachers can think about the subject matter like an expert who specializes in computational thinking	3.79	1.06
	CK3-Our teachers have sufficient knowledge of computational thinking	3.75	1.03
TPACK	TPACK1-Our teachers can teach lessons that appropriately combine computational thinking, technologies, and teaching approaches	3.75	1.11
	TPACK2-Our teachers can use strategies that combine content, technologies, and teaching approaches	3.46	1.14
	TPACK3-Our teachers can select technologies to use in the classroom that enhance what they teach, how they teach, and what students learn	3.58	1.18
	TPACK4-Our teachers can provide leadership in helping others to coordinate the use of content, technologies, and teaching approaches at my school	3.54	0.93

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE), 14*(4), 25.

Balanskat, A., & Engelhardt, K. (2014). *Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe*: European Schoolnet.

www.dbooks.org

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* (pp. 1–25). Vancouver, Canada.

Cetin, I. (2016). Preservice teachers' introduction to computing: exploring utilization of scratch. *Journal of Educational Computing Research, 54*(7), 997–1021.

Chai, C. S., Koh, J. H. L., & Tsai, C. C. (2010). Facilitating preservice teachers' development of technological, pedagogical, and content knowledge (TPACK). *Educational Technology & Society, 13*(4), 63–73.

Cheon, J., Lee, S., Crooks, S. M., & Song, J. (2012). An investigation of mobile learning readiness in higher education based on the theory of planned behavior. *Computers & Education, 59*(3), 1054–1064. https://doi.org/10.1016/j.compedu.2012.04.015.

Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying computational thinking for noncomputer scientists*. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Darab, B., & Montazer, G. (2011). An eclectic model for assessing e-learning readiness in the Iranian universities. *Computers & Education, 56*(3), 900–910. https://doi.org/10.1016/j.compedu.2010.11.002.

Falkner, K., Vivian, R., & Falkner, N. (2014). The Australian digital technologies curriculum: Challenge and opportunity. In: *Paper presented at the Proceedings of the Sixteenth Australasian Computing Education Conference* (Vol. 148).

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. In: *Paper Presented at the Frontiers in Education Conference (FIE)*: IEEE.

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.

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education, 82,* 263–279.

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.

Koehler, M. J., Mishra, P., & Yahya, K. (2007). Tracing the development of teacher knowledge in a design seminar: Integrating content, pedagogy and technology. *Computers & Education, 49*(3), 740–762.

Liu, M., Hu, W., Adey, P., Cheng, L., & Zhang, X. (2013). The impact of creative tendency, academic performance, and self-concept on creative science problem-finding. *PsyCh Journal, 2*(1), 39–47.

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.

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.

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record, 108*(6), 1017–1054.

Mouza, C., Yang, H., Pan, Y.-C., Ozden, S. Y., & Pollock, L. (2017). Resetting educational technology coursework for pre-service teachers: A computational thinking approach to the development of technological pedagogical content knowledge (TPACK). *Australasian Journal of Educational Technology, 33*(3).

Orvalho, J. (2017). *Computational Thinking for Teacher Education.* Paper presented at the Scratch2017BDX: Opening, Inspiring, Connecting.

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).

Prabhu, V., Sutton, C., & Sauser, W. (2008). Creativity and certain personality traits: Understanding the mediating effect of intrinsic motivation. *Creativity Research Journal, 20*(1), 53–66.

Ross, M., Perkins, H., & Bodey, K. (2016). Academic motivation and information literacy self-efficacy: The importance of a simple desire to know. *Library & Information Science Research, 38*(1), 2–9.

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.

Schmidt, D. A., Baran, E., Thompson, A. D., Mishra, P., Koehler, M. J., & Shin, T. S. (2009). Technological pedagogical content knowledge (TPCK): The development and validation of an assessment instrument for preservice teachers. *Journal of Research on Technology in Education, 42*(2), 27.

Williams, F. E. (1991). *Creativity assessment packet: Test manual*. Austin, TX: Pro-Ed.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 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 (TOCE), 14*(1), 5.

Yu, Y.-T., Liu, Y.-C., & Huang, T.-H. (2016). Support-object-personnel mobile-learning readiness model for primary and secondary schools. *Journal of Research in Education Sciences, 61*(4), 89–120.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 18
Self-development Through Service-Oriented Stress-Adaption-Growth (SOSAG) Process in the Engagement of Computational Thinking Co-teaching Education

Mani M. Y. Wong, Ron C. W. Kwok, Ray C. C. Cheung, Robert K. Y. Li and Matthew K. O. Lee

Abstract In this chapter, we propose the service-oriented stress-adaption-growth (SOSAG) process based on the existing intercultural transformation theory (ITT), through the engagement of cross-institutional tertiary students in computational thinking (CT) education at primary schools. Students from tertiary education institutions in Hong Kong are recruited, trained, and assessed to become qualified teaching assistants (TAs) for providing in-school co-teaching support in CT education in 32 primary schools. TAs are monitored and dispatched to different years and different classrooms in multiple regions in Hong Kong. Through service engagement, and the proposed service-oriented stress-adaption-growth (SOSAG) process, each TA has to undergo self-development in multiple stages, including pre-assessment, training, teaching practice, and in-class co-teaching. We summarize the ongoing challenges and future directions of SOSAG in this chapter.

Keywords Computational thinking · Teaching assistant · Co-teaching · Stress-adaption-growth · Service-oriented · Service learning · Intercultural transformation theory

M. M. Y. Wong · R. C. W. Kwok (✉) · R. C. C. Cheung · R. K. Y. Li · M. K. O. Lee
City University of Hong Kong, Hong Kong, China
e-mail: isron@cityu.edu.hk

M. M. Y. Wong
e-mail: mani.w@cityu.edu.hk

R. C. C. Cheung
e-mail: r.cheung@cityu.edu.hk

R. K. Y. Li
e-mail: Robert.Li@cityu.edu.hk

M. K. O. Lee
e-mail: ismatlee@cityu.edu.hk

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_18

18.1 Introduction

Programming and computing-related skills are vital in the information age both for personal and social development. In CoolThink@JC, City University of Hong Kong (CityU) aims to provide professional education support to enhance programming literacy among Hong Kong citizens through a series of elaborative teaching and learning activities, in particular targeting the primary school student group in the Hong Kong population.

Programming/coding has now become a global initiative in multiple countries, such as the "Hour of Code" campaign was 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 Australia, programming 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 the public to learn to code, and officiated the first "hour of code" in the territory. Over 2,000 h of programming has been achieved in the previous "Hour of Code HK" workshops, and we, at CityU of Hong Kong, have offered over 10,000 h of programming 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 to postgraduate level are getting increasingly involved in understanding the fundamentals of computer programs and programming skills. In the UK, a new version of the relevant curriculum was established a year earlier on July 8, 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 programming 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 is a plan being fermented by the 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 programs in classroom settings employing the teaching and educational resources, which are available to other fundamental curricula. 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 programs. It is one of the first countries to have a government that was fully enabled. The ProgeTiger initiative was started in January 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.

www.dbooks.org

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 programming. It is also important to know that when running the "Hour of Code HK" Campaign, we observe that youth group can achieve the programming 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 the students in the participating schools three years of in-class training and out-of-class mentoring support from junior, intermediate, up to advanced level. For the in-class training at each level, there are 8−14 lessons with each lasting around 35–45 min. The out-of-class mentoring support is provided by our university student mentors on a group basis (around two student mentors take care of a class of 40 students). The student mentors take part in this project through our established campus internship and other cocurricular experiential learning schemes.

In CoolThink@JC, a sustainable learning environment was created for a period of 3 years for the participating primary students to learn the skill to program and keep up the learning attitude. The main role of the CityU team is to provide in-class manpower support and 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 student members, is able to provide such 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 other Hong Kong higher education institutions who are passionate about computational thinking (CT) and programming education are also recruited to join this project.

Yet, one challenge identified by the research group is the diverse cultures and backgrounds of the recruited students, which the cultural difference is expected to be overcome in the stress-adaption-growth (SOSAG) process in the recruitment.

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.

18.2 Roles and Responsibilities of TAs

In CoolThink@JC, 100 and 500 teaching assistants (TA) were recruited by CityU from over 10 tertiary institutions of Hong Kong in the academic years of 2016/17 and 2017/18 to serve 32 pilot primary schools that participate in the computational thinking education in Hong Kong.

The main roles of TAs are to assist teachers in dealing with classroom teaching, e.g., co-teaching CT and answering students' enquiries in class in the pilot primary schools. They also help in creating a joyful and innovative learning environment, and act as a role model in the classroom (e.g., by providing a passionate and responsive presence).

Another major responsibility of TAs is to provide professional support to teachers in relation to teaching and learning. They have to motivate students' learning and encourage them to interact with others, for example, by praising students who have successfully completed the class exercises with creative ideas and are behaving well, and encouraging them to assist other classmates. Also, they take the role of inspiring students to generate creative ideas by encouraging students to finish their tasks by themselves with appropriate guidance. They have to be aware of student progress and achievements, and report any concerns regarding student matters to their supervisors, namely teaching leads (TLs).

18.3 Service-Oriented Stress-Adaption-Growth (SOSAG) Process

18.3.1 Assessment and Stress

TAs take the main role in providing support in CT lessons and act as an ultimate executor of co-teaching in primary schools. Before being assigned to serve in primary schools, potential candidates are trained and assessed based on their performances on a series of tests and teaching practices to become "qualified TAs".

Unlike other subjects or skill set training, teaching CT is not easy as learners are required to have thorough understanding of both concepts and mechanisms to acquire the thinking skills needed for asking questions, understanding problems, and identifying solutions. Training TAs to be qualified to provide support to CT teaching at a large scale is even more a challenging task. Instead of "spoon-feeding" candidates for the essential soft and hard skills as the knowledge for co-teaching CT, candidates are expected to have a good attitude and high motivation, especially in Stage 1 where there are frequent interactions between candidates and the assessor for examining candidates' understanding on CT concepts, CT practices, and CT perspectives. This ensures the training and assessment at a large scale can be conducted smoothly while maintaining quality.

www.dbooks.org

Among hundreds of TAs recruited from various academic background and experience, some do not have relevant education background while some lack relevant experiences, e.g., teaching or interacting with children. To overcome the cultural difference of a large group of TAs, assessments are crucial to evaluate and maintain the standard of TAs via various kinds of assessment methodologies. Potential candidates are exposed to stress in four stages of assessments, which include a test via electronic submission and interview screening (Stage 1), training assessment (Stage 2), teaching practice assessment (Stage 3), and probation assessment to be qualified TAs (Stage 4). The assessment stages are summarized in Fig. 18.1.

In various stages of assessments, stress is caused by the intercultural differences including but not limited to (i) education background (e.g., education, programming) and (ii) work experiences (e.g., teaching experience or experience with children). The potential challenges that new recruits for this program may face are highlighted and emphasized with an increasing extent in each stage to enlarge their stress, and therefore the adaption and growth eventually.

Many candidates expressed that they suffered from varying degree of anxiety, sorrow, and even pain in different stages due to the various reasons: some candidates are not confident to learn CT when being given a test related to Scratch in Stage 1; some are distressed when being asked to handle student issues during interview while some feel worried to handle a big class of students and answer all enquiries from students properly in the lesson.

To resolve the stress, adaptation takes place to promote qualitative transformation toward growth (Kim, 2001). We consider the "cultural shock" as a "catalyst" for potential TAs to adapt quickly and grow to make them fit in the roles in the service engagement.

18.3.2 Adaptation

Although stress may be considered as a negative emotion, an appropriate level of stress can be beneficial (Spencer-Oatey & Franklin 2009). Studies show that people under higher frequency of stress have higher level of adaption (Kim, 2001). Adaptation to "intercultural" differences marks a change in terms of behavior and attitude. In the adaption process, we offer TAs debriefing and reflection in different assessment stages to allow for self-review and lesson observation for self-improvement; for example, some candidates showed stress when failing in teaching practice caused by lateness or unsatisfactory performance. They reflected on the importance of time management and preparation.

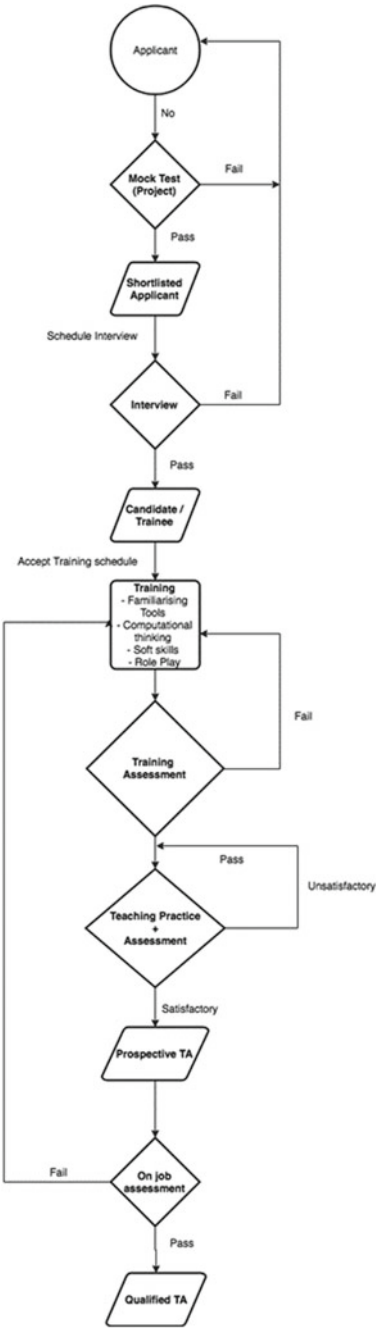


Fig. 18.1 The stages of becoming a qualified TA in CoolThink@JC of Hong Kong

www.dbooks.org

Table 18.1 ITT factors in the stages to become qualified TAs in CoolThink@JC of Hong Kong

ITT factors	SOSAG process			
	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching (at least 2 sessions)
Stress	✓	✓	✓	✓
Adaption		✓	✓	✓
Growth		✓	✓	✓

18.3.3 Growth

Adaptation also leads to psychological growth and better understanding of who we are, what we value, and where we might want to go (Shi, 2006). The journey of becoming qualified TAs of CoolThink@JC in the form of service learning helps to achieve a balance between service outcomes and learning goals (Furco, 1996). Service and learning goals are of equal weight and each enhances the other for all participants in service-learning (Sigmon, 1994). In the case of TAs in this project, service refers to the co-teaching support provided to teachers and students at primary schools while learning refers to the growth of TAs.

During the service, TAs are required to identify a problem, propose a solution, and learn from the experience (Crutsinger, Pookulangara, Tran, & Kim, 2004). During the lessons, TAs observe the students' learning progress to identify their problems, then think of a way to present the problem and to inspire the students to solve it. Experienced TAs usually learn from past experience to respond to students of different levels and characteristics for a better learning outcome.

We consider it a stress-adaption-growth process which eventually leads to the self-development of TAs. As an extension to the existing intercultural transformation theory (ITT), we propose that this multiple-stage service-oriented process leads to TAs' self-development as the "service-oriented stress-adaption-growth (SOSAG) process" highlights the growth via engaging candidates in service. The self-development of TAs is examined in Sect. 18.4—Evidences (Table 18.1).

18.4 Evidence

In this study, data was being collected and presented in the form of in-depth reflective summary submitted by TAs. The extracted content of the reflective summary was mapped against the corresponding factors of the stress-adaption-growth process of the intercultural transformation theory (ITT) (Kim & Ruben, 1988). Five cases were examined in light of ITT factors through the different stages of the SOSAG process.

The TA subjects in this study were invited to reflect on a number of attributes (listed in Sect. 18.4.1–18.4.2) which are expected of a qualified TA in CoolThink@JC.

18.4.1 Attitude

- Understanding the importance of the sense of responsibility (e.g., punctuality, being well prepared).
- Stepping out the comfort zone to accept new challenges which may be out of their profession.

18.4.2 Soft Skills

18.4.2.1 Communication Skills

- Communicating with various stakeholders including supervisors, school teachers, students, and partner TAs (e.g., observing students' learning progress and problems, communicating with students to give them a helping hand, handling student behavioral and disciplinary issues occasionally, and resolve conflicts between students).

18.4.2.2 Time and Stress Management

- For example, handling a certain number of students inquiries and issues within the limited lesson time.

18.4.2.3 Knowledge on CT and Programming

- For example, using CT concerts to solve problems in daily life.

Table 18.2 summarizes the TAs' self-review comments through the SOSAG in the service engagement in CT education. The comments that correspond to the respective ITT factors are highlighted for further analysis.

18.5 Discussion

18.5.1 Add-On Training

For the enhancement of knowledge in computational thinking, supplementary training will be offered to the existing TAs. Based on train-the-trainer model, experienced TAs form a taskforce to be trainers to design and offer add-on training for junior TAs.

www.dbooks.org

Table 18.2 Excerpts from reflective summaries submitted by teaching assistants in CoolThink@JC

(a) Stage 1—Pre-assessment

ITT factors	Stage 1—Pre-assessment
Stress	"With knowledge on different computer languages such as C++, how come the educational tools for primary school students will beat me? However, **I recognized that I might be too arrogant** after pre-assessment as my background and knowledge could not support answering. I felt nervous during interview. I actually have no idea nor experience on CT questions and handling problems in given scenarios. **There was a lot of stress in the pre-assessment so I did not perform well enough**." (Student 1) "As a linguistics student, several programming languages are required to be learnt for linguistic purposes. Yet, **I am not confident of CT and programming**. **Though it was undoubtedly stressful in the preparation**, I successfully went through the stages of online test and interview with sufficient self-learning and online research." (Student 2) "My journey in CoolThink@JC began with pre-assessment. Without any relevant background as majoring in Finance, **I encountered many difficulties** in the test which were solved by searching information online and even asking advice from friends. However, as I sowed, so I reaped. The pre-assessment helped me understand some CT knowledge before moving to next stage. Then, I was pretty stressful during interview as there are lots of CT mentioned on the question paper such as parallelism and data manipulation. I tried to understand by relating each concept to the events happened in my daily life." (Student 3) "Since I am a student studying Computer Science, almost all questions in the quiz could be answered with confident. However, if I wanted to get all correct, I still need to put some effort on revise some materials before start. **This made me feel a little worried about the difficulty of learning materials**. After that, **I was very nervous** in interview because I was weak in interview and also afraid of questions out of preparation. During the interview, there were questions asking to deal with some sudden problems in lesson. **This made me start to have some stress on the required soft skills of this job**, such as how could I handle a class of students and cooperate with teachers and other ambassadors." (Student 4) "It(CoolThink@JC) aroused my interest. However, **there is a big problem** as I am not studying a subject related to CT and programming, and I do not have any idea of CT concepts." (Student 5)
Adaption	
Growth	

(continued)

Table 18.2 (continued)

(b) Stage 2—Training	
ITT factors	Stage 2—Training
Stress	"I just remember that I was quite embarrassed to "perform" in front of the class. Even worse, **our demonstration was not comprehensive enough**." (Student 1) "Apart from the soft skill training, the hard skill coaching of Scratch and AI2 in the afternoon definitely enabled me to learn some solid skills. **It was quite challenging and stressful for me in the beginning**. The learning process generally was pleasurable thought **I felt a bit left behind and stressful in digesting the excessive new information in a limited time**. Luckily, TLs and a friend of mine provided guidance throughout the training." (Student 2) "**Sometimes I found I would lose myself in some steps and did not know where can find some codes or forget what was the next step**." (Student 4)
Adaption	"Luckily, CoolThink@JC is a family; **the professors gave us precious and useful feedbacks and suggestions from their expert aspect**, also **other groups provided supplementary methods from their experience**. Besides, we enjoyed the excellent performance of other groups and learnt to handle other situations." (Student 1) "Coming up with an instruction sheet for other groups to fold a paper plane was also fun-filled. It was a good chance for us to experience and recognize how to clearly deliver a message to our audiences, allowing us to realize if our instructions are clear and understandable for the receivers." (Student 2) "I was no longer stressful about Scratch because previous knowledge learned from pre-assessment allowed me to adapt new knowledge in CT faster." (Student 3) "Even though everything seemed complicated to me, I received help from supervisors. They were willing to help me when I had any problems encountered during the training." (Student 3) "Hence, I took myself as a student that I might help later on and remember what problems I would face and observed how the teacher helped and solved those problems." (Student 4) "At the beginning, I just think that the CT concepts can only apply in programming and computing. I never thought that CT concepts can exist in everywhere." (Student 5)
Growth	"Hence, I realized self-study and practice are also important apart from training provided." (Student 2) "This improved my performance on how could I observe the students who might want to ask some questions and assist them before they ask." (Student 4)

(continued)

www.dbooks.org

Table 18.2 (continued)

(c) Stage 3—Teaching practice	
ITT factors	Stage 3—Teaching practice
Stress	"Before I went to the teaching practice or co-teaching, **I was a little bit worried**. In spite of the unknown situation that I might be faced in co-teaching, but I was also worried that programming might be too difficult for the primary school students. When I was in primary school, I just learned some background knowledge about the computer hardware and history, plus some basic controls of computer only, instead of coding or programing. Besides, I am afraid if CT or programming might be too difficult for the immature kids to learn." (Student 1) "Before getting involved in co-teaching, we were sent to gain practical experience via teaching practice. **I thought I was ready to overcome any difficulty. Nevertheless, the thought was proven wrong**. Since I was not aware about the school location and arrival time, it took me long time to arrive. As result, I was late for my first ever teaching practice. **I felt guilty and ashamed about myself as I was not executing my duty**—Always on time. **Also, I was stressful and worried about the impact**." (Student 3) "**However, I was a bit sad that my first teaching practice was failed due to lateness**. Although I was just late for a few minutes, I understood it should not happen since every lesson at primary school is short and valuable. Being late is not permitted and may affect the lesson quality like insufficient co-teaching support to assist lesson delivery and cause extra troubles for the teachers like entering the room suddenly and interrupting the lesson." (Student 4) "In the second teaching practice, there were more TAs than usual situation in co-teaching. With the help of the currently working ambassadors, the class could be maintained in a good way. There were not many cases that some students were lagging behind a lot. However, TAs and teacher were so busy with the teaching, **making me a bit worried** of co-teaching and class management to solve students' problem in the future." (Student 4) "I still remember that **I was very nervous** when I had teaching practice in the primary school because there were many things to pay attention, such as assisting quality, classroom discipline and other uncertainties." (Student 5)
Adaption	"With preparations before the class, I could provide suggestions to help them." (Student 1) "Since it was my first teaching practice, the supervisor is willing to give me another chance. When participating in another teaching practice, I understood that planning should be always done before moving, which allows me to have sufficient time and effort to complete the duty." (Student 3) "I had more pressure in the second attempt and prepared more time than the last time for taking transportation, finding ways and walking to prevent being late again, therefore I arrived on time and could try to help the students with their needs." (Student 4) "Experience really helped a lot. The more problems we handled, the more we experience we had to understand how to handle different cases. Therefore, I always notice how other ambassadors and teacher supported different students, remind myself how can I do in the similar condition." (Student 4) "In the classroom, there were other experienced TAs. I took them as a reference and I started to imitate the way they are." (Student 5)

(continued)

Table 18.2 (continued)

Growth	"The teaching practice helped me a lot, as I gained the experience and knowledge to control the whole class and teach the kids, also it strengthens my CT ability." (Student 1) "Through the teaching practice, I also noticed the duties of a teacher are not only delivering the solid knowledge to students in the lesson, but also paying time and effort to prepare teaching materials beforehand. **It is very crucial to ignite students' passion in the subject and hence motivate them to learn and further develop their skills**." (Student 2) "I was satisfied to prove myself being able to handle the teaching practice." (Student 3) "I realized the importance of time management" (Student 4) "Gradually, I build up my own set of practice. I am confident to have co-teaching and I can be more mature to handle any difficulties." (Student 5)
(d) Stage 4—Co-teaching	
ITT factors	Stage 4—Co-teaching
Stress	"**However, a certain level of stress came out as** I was assigned to be the TA for specific classes cooperating with the same teacher continuously. As I was no longer a trainee conducting teaching practice in one random lesson, I had much more responsibility to help the teacher while taking care the students as a regular TA of the class." (Student 1) "It is true that the students' characters from different schools are highly diverse, regarding learning motivation, concentration as well as conduct in class. In some schools, **I always found the students are more creative and willing to learn**. They understand most of the content and able to work on their own with a little assistance. **In contrast, I feel stressed at some schools whose students show no interest in Scratch and hence they are unwilling to listen and work on the assigned tasks. I was frustrated and worried for students being reluctant to learn. It is comparatively difficult to ignite their passion in Scratch**." (Student 2) "The most challenging part was going to different schools for co-teaching as the variations include teachers, partner TAs, lesson venue, and teaching progress." (Student 4) "**Here comes another challenge**. Some students were very naught, who did not stay at theit seat properly and kept talking with others. They acted obediently for a short time after my warning and resume their naughty behavior again. **I was a bit confused as I did not know what I can do**."

(continued)

www.dbooks.org

Table 18.2 (continued)

Adaption	"Before the lesson, I went through the teaching material with full understanding, and tried the block and app before to confirm the steps." (Student 1) "During the co-teaching, I had to follow the teaching progress of the assigned class. Although they were just lessons for primary school students, I had to cooperate with the teacher well to make sure that the lesson could continue fluently. Despite the class time was short, normally 30 min to 1 h, there were a lot of uncertainties during the class. Beside of the content delivering to the students, I had to make sure that the students paid attention during the lesson to prevent them missing teacher's instructions." (Student 1) "I realized that the attitude and behavior of teachers can actually have direct influences to their students. I believe every student can feel whether his or her teachers are teaching and guiding them from heart or just simply demonstrating the procedures of Scratch to them. Through observation and experience, I deeply understand being teacher is not an easy job. It requires a lot of time, energy and effort in order to be a responsible teacher and even TAs." (Student 2) "Fortunately, my partner was experienced with the situation. Thus, he told me to report it to the teacher after the teacher finished important slide. Eventually, everyone sat still and worked hard on the scratch together." (Student 3) "**Every co-teaching session may be boring to others, but not to me because I got to know something new every time**." (Student 3) "Since different classes might have different progress, I would read the learning materials and my co-teaching notes in the preparation time so to have a brief idea of what was taught and would be taught in the last and following lesson. Hence, no matter what the teacher was going to teach, I still could pick up quickly and know what and how should I help the students in the lesson." (Student 4) "During the lesson, apart from students, it was important for ambassadors to pay attention to what the teacher was doing, in order to provide immediate support and response to the teacher to ensure a smooth lesson delivery." (Student 4) "Between ambassadors, we would have a labor division for looking after different students. If students in this group were in good progress, then we might observe other students." (Student 4)
Growth	"**Not only do I step out my comfort zone to learn a discipline that I was not familiar with, but I also get a taste of being a TA in primary school**. Even though being a teacher might not be the first priority in my future career path, various training in this program enable me to understand the importance of the sense of responsibility and having correct attitude at work. I have also realized nothing is impossible if we are willing to learn from our mistakes and get along with stress, and hopefully turn it into our motivation to become a better individual." (Student 2) "For myself, I learned CT, which encouraged me to solve problem by searching solution from different aspects. When solving problem, I would like to find out what was the root to tackle the problem effectively and efficiently. Besides, it is important to be responsible to perform or complete a task assigned satisfactorily and on time. I am grateful to CoolThink@JC which gives me chances to learn about the computational thinking and correct personality that I need to have for my career. **Everyone I met here had always given me direction and feedback to push me one step forward on my career path**." (Student 3) "Up till now, I have joined the CoolThink@JC programme for more than a year. I have participated in different posts such as co-teaching, backup duties and training programme. Meanwhile, **I am fortunate to become one of the supervisory TA. I am delighted that I can involve in supervisor duties. Each post provides different experiences for me. It makes me growth and it broadens my horizon. In CoolThink@JC, I do not only learn CT concept, but also equip myself with different knowledge and skills**." (Student 5)

Under sufficient guidance by supervisors, they can become a good trainer to incorporate interactivity and foster thought-provoking conversations among peers.

18.5.2 Promotion

Based on enhanced intercultural transformation theory (ITT) (Sivakumar & Kwok, 2017), promotional exercises can be considered as "motivation" while add-on training and advice from supervisors are "support" in the TAs' SOSAG process.

After the completion of at least one semester of service, TAs are eligible to apply for promotion via self-application or supervisors' nomination. They may be promoted to "Senior TAs" or even "Supervisory TAs" based on the following factors:

- Personal motivation and willingness to take up additional duties.
- Experience.
- Performance review.
- Evaluation conducted during individual and group interview for promotion.

Promoted TAs are expected to take up additional duties including to act as the role model for TAs, to help new TAs get acquainted with work environment, and to assist supervisors in managing TAs by conducting a quality inspection in relation to co-teaching support in various schools irregularly.

Some senior and supervisory TAs also involve in designing and offering add-on training to other TAs under the guidance of supervisors. They are trained to be the trainers. This train-the-trainer model allows outstanding TAs to maximize their potential ability.

18.5.3 Challenges

One of the biggest challenges in engaging hundreds of tertiary students in the CT education in primary schools is to maintain manpower bank. Due to the complexity of matching school lesson schedule and TA preferences according to availability and school location, a pool of well-trained TAs is needed to serve more than thirty primary schools.

www.dbooks.org



Fig. 18.2 TAs from CoolThink@JC supported CT-related workshops in InnoTech 2017

If a similar model of providing co-teaching support were to be continually adopted in the CT education in primary schools, TA recruitment may need a wider and stronger support from all higher education institutions. Recognition of the importance of CT education and its impact by various parties in the society could be a significant motivator in the collaboration with higher education institutions. This mobilizes a bigger pool of potential TA candidates ready for screening process, and eventually allowing more suitable candidates can be identified. This is actually a win-win approach for both the service receivers (teachers and primary students) and givers (TAs).

18.5.4 Future Directions

Besides the duties of co-teaching in primary schools, TAs are also actively participating in the support of CT workshops for public, for example, some were sent to support programming workshops for the public in InnoTech Expo 2017, a large-scale innovation and technology event held in Hong Kong Convention and Exhibition Centre. Through co-organizing the events with volunteers from different backgrounds, TAs learned new knowledge like using programs to control drones and gained new exposure and insights to the application of programming in real life. The pictures in Fig. 18.2 show our TAs' engagement in the InnoTech Expo 2017.

In the long run, TAs with the experience in CT education will have a higher chance of engagement in CT education and related industries. Besides CT education, TAs are more likely to serve the community continually in different aspects and be more prepared to become future pillars in the society.

The potential development of qualified TAs is summarized and illustrated in Fig. 18.3.

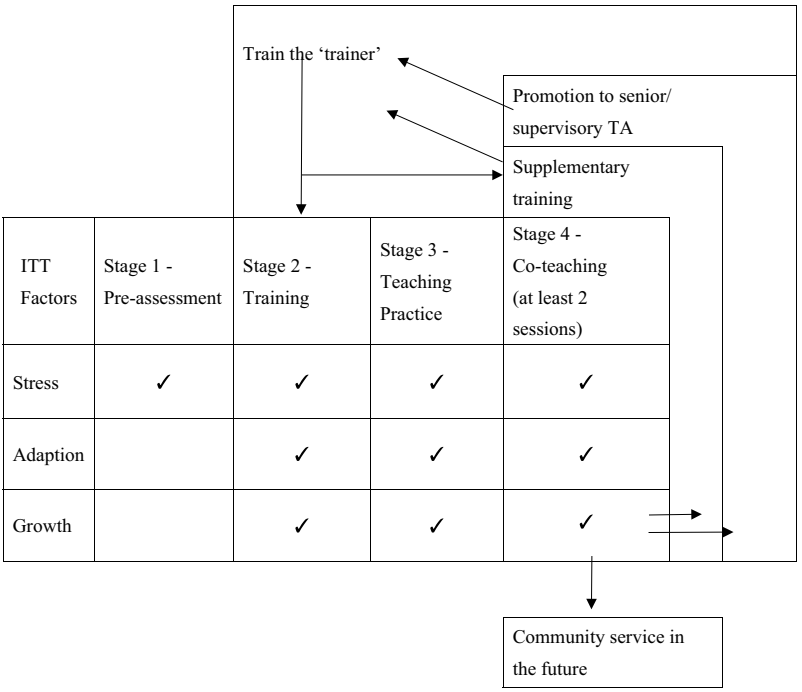


Fig. 18.3 Potential development of qualified TAs in CoolThink@JC of Hong Kong

18.6 Conclusion

This chapter extends the existing intercultural transformation theory (ITT) and proposes the service-oriented stress-adaption-growth (SOSAG) process in the engagement of computational thinking co-teaching education. Through service engagement in CT education at primary schools, service-oriented stress-adaption-growth process took place and allowed TAs to undergo self-development in multiple stages.

www.dbooks.org

Appendix

Case 1: Ordinary TAs

Student 1 from Electronic Engineering (Years of Service: 2016/17–2017/18)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Stress	With knowledge on different computer languages such as C++, how come the educational tools for primary school students will beat me? However, **I recognized that I might be too arrogant** after pre-assessment as my background and knowledge could not support answering. I felt nervous during interview. I actually have no idea nor experience on CT questions and handling problems in given scenarios. **There was a lot of stress in the pre-assessment so I did not perform well enough**	Some real situations that will be faced in co-teaching were given, such as how we can handle and help the teacher to control students during the lesson. Each group was required to demonstrate how to solve potential problems in front of other groups. Our group topic was what should ambassador do if students do not follow the instructions of the teacher and harass other students. I did not remember what we played in front of the classroom; I just remember that I was quite embarrassed to "perform" in front of the class. Even worse, **our demonstration was not comprehensive enough**	Before I went to the teaching practice or co-teaching, **I was a little bit worried**. In spite of the unknown situation that I might be faced in co-teaching, but I was also worried that programming might be too difficult for the primary school students. When I was in primary school, I just learned some background knowledge about the computer hardware and history, plus some basic controls of computer only, instead of coding or programing. Also, in my generation, computer science or programming was not a core subject or even not important. Besides, I am afraid if CT or programming might be too difficult for the immature kids to learn	After the teaching practice, I started my co-teaching works. Similar with the teaching practice, I found my position to support the teachers while answering inquiries from students. **However, a certain level of stress came out as** I was assigned to be the TA for specific classes cooperating with the same teacher continuously. As I was no longer a trainee conducting teaching practice in one random lesson, I had much more responsibility to help the teacher while taking care the students as a regular TA of the class

(continued)

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Adaption		Luckily, CoolThink@JC is a family; **the professors gave us precious and useful feedbacks and suggestions from their expert aspect**, also **other groups provided supplementary methods from their experience**. Besides, we enjoyed the excellent performance of other groups and learnt to handle other situations. These role-play games gave me a very good experience	But after the first teaching practice, I found that my worries were unnecessary. During the teaching practice, teachers were well trained with Scratch and App Inventor 2. They controlled the whole class and indicated what ambassadors should do. Also, I found that nowadays primary school students are very talented and enthusiastic. They would like to help their classmates to solve the problems together. Most of them could finish their own work timely or even earlier. Then they would start to observe their classmates' works and see whether they could improve their own project or not. Therefore, I could give them some higher level task and strengthen their computational thinking. Even better, they could use different categories and blocks to optimize and yield their own project. Actually I was quite astonished that students had the ability to use other command block and make the block more concise and efficient. Also, sometimes they could fix the mistakes or change the parameters in the interface by observation. But as they were still primary school students, there might be something that was too difficult for them. With preparations before the class, I could provide suggestions to help them	Before the lesson, I went through the teaching material with full understanding, and tried the block and app before to confirm the steps. During the co-teaching, I had to follow the teaching progress of the assigned class. Although they were just lessons for primary school students, I had to cooperate with the teacher well to make sure that the lesson could continue fluently. Despite the class time was short, normally 30 min to 1 h, there were a lot of uncertainties during the class. Beside of the content delivering to the students, I had to make sure that the students paid attention during the lesson to prevent them missing teacher's instructions. For example, we had to prevent and advice the student to surf another websites, such as Facebook and YouTube, that is not related to CoolThink@JC or teaching material. Also, I was responsible to pay extra attention to some students with special educational needs to make sure that they could follow the teaching material. Sometimes there were some naughty students who did not follow the instructions of the class; we had to help the teacher to maintain the good order of the classroom. Even more, as the teaching materials became more and more difficult, we had to practise ourselves well daily to train our ability and familiarity of the Scratch and App Invertor 2, otherwise we could not follow the lesson too
Growth		**Not only does it help me to solve common problems we face in co-teaching, but it also gains my experience in co-teaching and makes me calm down with no disarray**. These helped me a lot	The teaching practice helped me a lot, as I gained the experience and knowledge to control the whole class and teach the kids, also it strengthens my CT ability	Until now, I am still just a qualified TA but not a good TA, as **I believe a well-prepared TA has to practise and practise more to gain the experience**

www.dbooks.org

Student 2 from Linguistics (Year of Service: 2017/18)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Stress	As a linguistics student, several programming languages are required to be learnt for linguistic purposes. Yet, **I am not confident of CT and programming**. **Though it was undoubtedly stressful in the preparation**, I successfully went through the stages of online test and interview with sufficient self-learning and online research	A series of training and briefing were arranged to all of the ambassadors after the selection. I think the most remarkable part would be the 1-day training workshop on both soft and hard skills. With role-playing training in the morning, we are able to understand how to handle different circumstances in classroom, such as resolving the quarrels between students and dealing with emergencies in class. Coming up with an instruction sheet for other groups to fold a paper plane was also fun-filled. It was a good chance for us to experience and recognize how to clearly deliver a message to our audiences, allowing us to realize if our instructions are clear and understandable for the receivers. Apart from the soft skill training, the hard skill coaching of Scratch and AI2 in the afternoon definitely enabled me to learn some solid skills. **It was quite challenging and stressful for me in the beginning**. The learning process generally was pleasurable thought **I felt a bit left behind and stressful in digesting the excessive new information in a limited time**. Luckily, TLs and a friend of mine provided guidance throughout the training	Nil	In the current time, I am co-teaching at two primary schools. It is true that the students' characters from different schools are highly diverse, regarding learning motivation, concentration as well as conduct in class. In some schools, **I always found the students are more creative and willing to learn**. They understand most of the content and able to work on their own with a little assistance. **In contrast, I feel stressed at some schools whose students show no interest in Scratch and hence they are unwilling to listen and work on the assigned tasks**. I also observed that most of them rather chitchat with their classmates instead of listening to the teacher. I usually have to guide them from the very first step until the last step, given that they are willing and patient to listen. **I was frustrated and worried for students being reluctant to learn. It is comparatively difficult to ignite their passion in Scratch**

(continued)

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Adaption		Hence, I realized self-study and practice are also important apart from training provided	During teaching practice, hands-on experience was the most valuable reward for me. It was my first time being a TA in school and I was glad that the whole teaching process was smooth and manageable	Furthermore, the teaching styles are very distinct in different schools. Through multiple duties in various schools, I have learned that I should always try my best to assist the students to catch up with their teacher. I realized that the attitude and behavior of teachers can actually have direct influences to their students. I believe every student can feel whether his or her teachers are teaching and guiding them from heart or just simply demonstrating the procedures of Scratch to them. Through observation and experience, I deeply understand being teacher is not an easy job. It requires a lot of time, energy and effort in order to be a responsible teacher and even TAs
Growth		Generally, we all think **we have stepped out our comfort zone and accepted new challenges beyond our school professions**	Through the teaching practice, I also noticed the duties of a teacher are not only delivering the solid knowledge to students in the lesson, but also paying time and effort to prepare teaching materials beforehand. **It is very crucial to ignite students' passion in the subject and hence motivate them to learn and further develop their skills**	To conclude, it is true to say that I have learned so much since the first day of this program. After going through different stages, I have become a qualified Teaching Assistant in CoolThink@JC. It has certainly been a precious experience in my University life. **Not only do I step out my comfort zone to learn a discipline that I was not familiar with, but I also get a taste of being a TA in primary school**. Even though being a teacher might not be the first priority in my future career path, various training in this program enable me to understand the importance of the sense of responsibility and having correct attitude at work. I have also realized nothing is impossible if we are willing to learn from our mistakes and get along with stress, and hopefully turn it into our motivation to become a better individual

www.dbooks.org

Case 2: TAs who are required to redo Teaching Practice

Student 3 from business (Year of Service: 2017/18)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Stress	My journey in CoolThink@JC began with pre-assessment. Without any relevant background as majoring in Finance, **I encountered many difficulties** in the test which were solved by searching information online and even asking advice from friends. However, as I sowed, so I reaped. The pre-assessment helped me understand some CT knowledge before moving to next stage. Then, I was pretty stressful during interview as there are lots of CT mentioned on the question paper such as parallelism and data manipulation. I tried to understand by relating each concept to the events happened in my daily life	After successfully entered group of TAs in CoolThink@JC, I must complete the training before I start my duty. In training, I learned more about computational thinking and making my own application such as piano application	Before getting involved in co-teaching, we were sent to gain practical experience via teaching practice. **I thought I was ready to overcome any difficulty. Nevertheless, the thought was proven wrong**. Since I was not aware about the school location and arrival time, it took me long time to arrive. As result, I was late for my first ever teaching practice. **I felt guilty and ashamed about myself as I was not executing my duty**—Always on time. **Also, I was stressful and worried about the impact**	Finally, getting into co-teaching session, as I have planned the route to the primary school before hand, I got to there on time. **Here comes another challenge**. Some students were very naught, who did not stay at their seat properly and kept talking with others. They acted obediently for a short time after my warning and resume their naughty behavior again. **I was a bit confused as I did not know what I can do**

(continued)

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Adaption		I was no longer stressful about handling the Scratch because previous knowledge learned from pre-assessment allowed me to adapt new knowledge in CT faster. Besides, I learned about how to execute co-teaching practice in next academic year. What I should do and what should be avoided during co-teaching duty. Even though everything seemed complicated to me, I received help from supervisors. They were willing to help me when I had any problems encountered during the training	Since it was my first teaching practice, the supervisor is willing to give me another chance. When participating in another teaching practice, I understood that planning should be always done before moving, which allows me to have sufficient time and effort to complete the duty. Eventually, I came earlier because I did preparation before the practice	Fortunately, my partner was experienced with the situation. Thus, he told me to report it to the teacher after the teacher finished important slide. Eventually, everyone sat still and worked hard on the scratch together. **Every co-teaching session may be boring to others, but not to me because I got to know something new every time**. The method of creating an application was revised for me every time when I have co-teaching session. Besides, students' creativity can always surprise me. For example, in doing the maze run on scratch, some students made it as Halloween version, while some added elements that were not mentioned. I believe in the future, some of them can become great programmer with lots of creativity and create more useful applications for us

(continued)

www.dbooks.org

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Growth		Nil	I was satisfied to prove myself being able to handle the teaching practice	For myself, I learned CT, which encouraged me to solve problem by searching solution from different aspects. When solving problem, I would like to find out what was the root to tackle the problem effectively and efficiently. Besides, it is important to be responsible to perform or complete a task assigned satisfactorily and on time. If I don't perform the task satisfactorily and on time, students and teacher may suffer as teacher cannot handle all the students at once and chaos may occur. I am grateful to CoolThink@JC which gives me chances to learn about the computational thinking and correct personality that I need to have for my career. **Everyone I met here had always given me direction and feedback to push me one step forward on my career path**

Student 4 from Computer Science (Year of Service: 2017/18)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Stress	Since I am a student studying Computer Science, almost all questions in the quiz could be answered with confident. However, if I wanted to get all correct, I still need to put some effort on revise some materials before start. **This made me feel a little worried about the difficulty of learning materials**. After that, **I was very nervous** in interview because I was weak in interview and also afraid of questions out of preparation. During the interview, there were questions asking to deal with some sudden problems in lesson. **This made me start to have some stress on the required soft skills of this job**, such as how could I handle a class of students and cooperate with teachers and other ambassadors	After being shortlisted as an ambassador, a 1-day training was conducted to reinforce the knowledge of the programming tools which are the learning materials for students. **Sometimes I found I would lose myself in some steps and did not know where can find some codes or forget what was the next step**	Before I go co-teaching in the schools, a teaching practice was needed for giving me a preview of what co-teaching is doing in real situation. I think I can learn some soft skills like how to work with other ambassadors and teachers so to make me more confident to do this job. **However, I was a bit sad that my first teaching practice was failed due to lateness**. Although I was just late for a few minutes, I understood it should not happen since every lesson at primary school is short and valuable. Being late is not permitted and may affect the lesson quality like insufficient co-teaching support to assist lesson delivery and cause extra troubles for the teachers like entering the room suddenly and interrupting the lesson. In the second teaching practice, there were more TAs than usual situation in co-teaching. With the help of the currently working ambassadors, the class could be maintained in a good way. There were not many cases that some students were lagging behind a lot. However, TAs and teacher were so busy with the teaching, making me a bit worried of co-teaching and class management to solve students' problem in the future	The most challenging part was going to different schools for co-teaching as the variations include teachers, partner TAs, lesson venue, and teaching progress

(continued)

www.dbooks.org

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Adaption		Hence, I took myself as a student that I might help later on and remember what problems I would face and observed how the teacher helped and solved those problems	I had more pressure in the second attempt and prepared more time than the last time for taking transportation, finding ways and walking to prevent being late again, therefore I arrived on time and could try to help the students with their needs. Experience really helped a lot. The more problems we handled, the more we experience we had to understand how to handle different cases. Therefore, I always notice how other ambassadors and teacher supported different students, remind myself how can I do in the similar condition	Before the lesson, there was some time for ambassadors to prepare. Since different classes might have different progress, I would read the learning materials and my co-teaching notes in the preparation time so to have a brief idea of what was taught and would be taught in the last and following lesson. Hence, no matter what the teacher was going to teach, I still could pick up quickly and know what and how should I help the students in the lesson. During the lesson, apart from students, it was important for ambassadors to pay attention to what the teacher was doing, in order to provide immediate support and response to the teacher to ensure a smooth lesson delivery. For example, when the teacher was mentioning some worksheets and we might help teacher to distribute the materials so to save time or when the teacher asked the students to follow to do the tasks, then we might start checking the progress of the students and provide help when necessary. Between ambassadors, we would have a labor division for looking after different students. If students in this group were in good progress, then we might observe other students
Growth		This improved my performance on how could I observe the students who might want to ask some questions and assist them before they ask	I realized the importance of time management	Nil

Case 3: Outstanding TAs

Student 5 from Public Policy (Years of Service: 2016/17–2017/18) who was promoted to supervisory TA after a year of service

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Stress	CoolThink@JC invited university's student to be a teaching assistant. It aroused my interest. However, **there is a big problem** as I am not studying a subject related to CT and programming, and I do not have any idea of CT concepts. On the other hand, I realized that it is a rare chance for me to be a teaching assistant. In view of this, I apply to the application. To be honest, it is quite frustrating. It is because there were a quiz and an interview which determine whether I can be a TA. Without any related knowledge, I may not have a good performance	At the beginning, I just think that the CT concepts can only apply in programming and computing. I never thought that CT concepts can exist in everywhere	Before having constant co-teaching, all the TAs must attend teaching practice that TL will assess the performance of TAs. I still remember that **I was very nervous** when I had teaching practice in the primary school because there were many things to pay attention, such as assisting quality, classroom discipline and other uncertainties	Nil

(continued)

www.dbooks.org

(continued)

ITT factors	Stage 1—Pre-assessment	Stage 2—Training	Stage 3—Teaching practice	Stage 4—Co-teaching
Adaption		Meanwhile, I was excited because I can edit an application of the smartphone myself. Also, it was a first experience for me to learn programming. At the end, it had a quiz which tests TAs understanding toward the Scratch and the App inventor. It was not that difficult after listening to the instruction of the teaching leader	In the classroom, there were other experienced TAs. I took them as a reference and I started to imitate the way they are	Nil
Growth		After accomplishing a set of assessment, I have finally become a TA	Gradually, I build up my own set of practice. I am confident to have co-teaching and I can be more mature to handle any difficulties	Up till now, I have joined the CoolThink@JC program for more than a year. I have participated in different posts such as co-teaching, backup duties and training program. Meanwhile, **I am fortunate to become one of the supervisory TA. I am delighted that I can involve in supervisor duties. Each post provides different experiences for me. It makes me growth and it broadens my horizon. In CoolThink@JC, I do not only learn CT concept, but also equip myself with different knowledge and skills**

References

Crutsinger, C., Pookulangara, S., Tran, G., & Kim, D. (2004). Collaborative service learning: A winning proposition for industry and education. *Journal of Family and Consumer Sciences*, *93*(3), 46–52.

Furco, A. (1996). Service-learning: A balanced approach to experiential education. In *Introduction to service-learning toolkit* (pp. 9–13).

Kim, Y. (2001). *Becoming intercultural: an integrative theory of communication and cross-cultural adaptation*. Google Books.

Kim, Y., & Ruben, B. D. (1988). Intercultural transformation: A systems theory. In Y. Kim & W. Gudykunst (Eds.), *Theories in intercultural communication* (pp. 299–321). Newbury Park: Sage, cop.

Shi, X. (2006). Intercultural transformation and second language socialization. *Journal of Intercultural Communication, 11*.

Sigmon, R. (1994). *Serving to learn, learning to serve. linking service with learning* (Council for Independent Colleges Report).

Sivakumar, C., & Kwok, C. W. (2017). Course design based on enhanced intercultural transformation theory (EI): Transforming information systems (IS) students into inventors during academic exchange. (A. 19, Ed.) *Communications of the Association for Information Systems, 40*.

Spencer-Oatey, H., & Franklin, P. (2009). *Intercultural interaction: a multidisciplinary approach to intercultural communication*. Basingstoke: Palgrave Macmillan Ltd.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Part VI
Computational Thinking in Educational Policy and Implementation

Chapter 19
Educational Policy and Implementation of Computational Thinking and Programming: Case Study of Singapore



Peter Seow, Chee-Kit Looi, Meng-Leong How, Bimlesh Wadhwa and Long-Kai Wu

Abstract Many countries that recognise the importance of Computational Thinking (CT) skills are implementing curriculum changes to integrate the development of these skills and to introduce programming into formal school education. In countries such as the United Kingdom, Lithuania, Finland, Korea and Japan, initiatives and policies are made to introduce the development of CT skills and programming in the schools. This chapter provides an in-depth analysis of policies of CT in the education of one particular country, namely Singapore. We review Singapore's approach to its implementation of CT education by first describing various initiatives in Singapore for Preschool, Primary and Secondary schools. Unlike several countries that have 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 of relying on an ecosystem is characterised by allowing schools the choice to opt-in, nurturing students' interest in computing, upskilling teachers in computing and a multi-agency approach.

Keywords Singapore programming initiatives · Computational thinking · Ecosystem in learning computing · Physical computing

P. Seow (✉) · C.-K. Looi · M.-L. How · L.-K. Wu
National Institute of Education, Singapore, Singapore
e-mail: peter.seow@nie.edu.sg

C.-K. Looi
e-mail: cheekit.looi@nie.edu.sg

M.-L. How
e-mail: mengleong.how@nie.edu.sg

L.-K. Wu
e-mail: longkai.wu@nie.edu.sg

B. Wadhwa
National University of Singapore, Singapore, Singapore
e-mail: bimlesh@nus.edu.sg

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_19

www.dbooks.org

19.1 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 a great interest from academia, industry and government in Computational Thinking (CT) and programming. Sites such as code.org, which is sponsored by industry giants like Google, provide free resources on learning programming to anyone who is interested. Though research in programming and computing education has been around for a few decades going back to the introduction of Logo in the 1970s, there is a renewed interest in learning programming and how it develops CT skills. National governments in addressing the shift from a knowledge/information economy to an economy driven by computation, are introducing educational policies that would prepare their citizens to be future ready. Reflecting 10 years after her seminal publication on CT, Wing (2017) said she never dreamt that Computer Science education, which was once available only at university level, would be introduced in K-12 at such a large scale today. Governments, educational authorities and schools are introducing Computer Science education at different levels of education. In countries such as the United Kingdom, Lithuania, Finland, Korea and Japan, initiatives and policies are made to introduce Computational Thinking skills and programming in the schools. This paper describes Singapore's effort in introducing the CT and programming in the education from preschool to secondary schools.

19.2 International Initiatives

England is one of the first countries to implement Computational Thinking in its K-12 curriculum (Bocconi, Chioccariello, & Earp, 2018). In 2014, the curriculum was reformed to infuse CT into the curriculum, and it is organised in four key stages over the span of formal K-12 education in the UK. For each stage, students are expected to develop aspects of Computational Thinking skills progressively. For example, in Key Stage 1 (age 5–7 years old), students create and debug simple programmes, in Key Stage 2 (age 7–11 years old), they can design, write and debug programmes to accomplish specific goals, in Key Stage 3 (age 11–14 years old), students can design, use and evaluate computational abstractions that model the behaviour of problems and at Key Stage 4 (age 14–16), students develop and apply their analytical, problem-solving, design and computational thinking skills (DfE, 2017). The development of CT skills is integrated in the curriculum, paving the way for a reform in how subjects such as language, mathematics and science are taught.

In Europe, Finland has introduced algorithmic thinking and programming as a compulsory cross-curricular activity from grade 1. The development of a new National Core curriculum for primary and lower secondary schools in 2014 included

learning objectives that relate to aspects of CT and programming, as well as developing problem-solving skills in the context of real-world problems.

In Asia, countries like Japan and Korea are planning to make programming a compulsory component of education in schools. Japan recently announced plans to make computer programming compulsory for all primary school children by 2020, followed by middle school students in 2021 and high school students by 2022 (*Japan Times*, 2017). South Korea prepares students for its Creative Economy with strategies such as its Software Education initiative. The changes in the curriculum is focused on developing skills, CT and creative expression through programming, which will be implemented at all levels from primary to university education. The new programme will become mandatory in 2018 for the primary and lower secondary levels (APFC, 2017).

19.3 Computing Programmes in K-10

In 2014, Singapore launched the Smart Nation initiative, a nationwide effort to harness technology in the sectors of business, government and home to improve urban living, build stronger communities, grow the economy and create opportunities for all residents to address the ever-changing global challenges (Smart Nation, 2017). One of the key enablers, to support the above initiative, is to develop the nation's computational capabilities. Programmes are implemented to introduce and develop CT skills and programming capabilities from preschool children to adults. We survey the landscape of K-10 CT and programming-related curricula in Singapore, which are implemented by various government organisations. We present these programmes, organised according to three groups: Preschool, Primary and Secondary.

19.3.1 Preschool

In Singapore, children aged from 3 to 6 years old attend preschools 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 Preschools in Singapore (IMDA, 2017). There are over 3000 preschools in Singapore. The initial phase involved piloting the programme in 160 preschools. IMDAs approach to introducing CT is to use electronic, robotic or programmable toys that would engage young children in play while developing CT skills such as algorithmic thinking. IMDA provided a set of the toys to pilot centres for use in the classroom by the teachers.

The toys selected by IMDA for playful exploration of technology are: (1) Beebot; (2) Circuit Stickers; and (3) Kibo. The Beebot is a toy with simple programmable steps to control its movement. Children can program the toy to move it along a path by logically sequencing the number of steps to move and control its direction.

www.dbooks.org

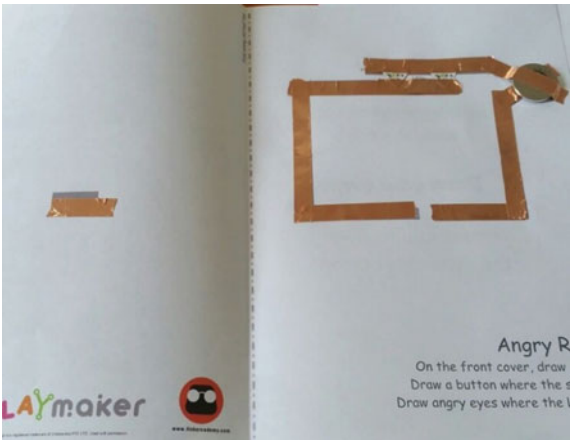


Fig. 19.1 Circuit stickers

Playing Beebot can help young children to develop problem-solving skills and logical thinking as they plan and program the movement of the toy. Kibo, developed by researchers in Tuft University, allowed children to create a sequence of instructions by arranging Kibo wooden blocks. The blocks can be scanned in a sequence with the instructions passed to the robot to execute the steps. Circuit sticker is a toolkit comprised of peel-and-stick electronic components such as LEDs and conductive copper tapes. With this 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 Fig. 19.1). Children can creatively 'make' while learning and applying basic electricity concepts.

Preschool 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 typically have apprehensions or concerns in using technology in their lessons. To address teachers' lack of experience and concerns, IMDA organises teacher seminars and workshops for teachers to experience the use of the Beebot, Kibo and Circuit Stickers. The hands-on sessions are facilitated by the instructors to introduce teachers to the tech toys and work on simple projects. These experiences enable teachers to understand the potential learning opportunities for their students by first learning the technology for themselves. Hands-on sessions 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 preschool teachers, IMDA works with a local polytechnic which offers preschool training for teachers. At the Preschool Learning Academy, preschool lecturers and trainers, together with technologists work together to formulate the use of the various tech toys in the preschool classroom. The learning experiences were shared among the teachers. Such a collaboration created an opportunity to understand how the tools could be used in the classroom and it also built the capacity among the trainers to work with the teachers on how these tools can be used to develop the students' learn-

ing potential. The academy was able to provide ongoing professional development to the current and new teachers.

Marina Bers and Amanda Sullivan were engaged by the IMDA to study the effectiveness of the KIBO program in the preschools. They studied the preschools' implementation of a curriculum called 'Dances from Around the World'. They found that the children were successful in mastering the foundational concepts of programming and that the teachers were successful in promoting a creative environment (Sullivan & Bers, 2017).

19.3.2 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. As of March 2016, the programme has been implemented in 117 schools with about 34,000 participating students. The goals of the programme are to expose a large base of students to CT concepts and programming, develop a generation of workforce equipped with basic programming and CT skills. To scale-up this enrichment programme, IMDA invited technology training partners to propose 10 h programmes that would include programming activities using a visual programming language, such as Scratch, combined with a robotic kit such as the MoWay or microcontrollers such as the micro:bit. The programme aims to make students appreciate programming and develop CT skills in problem-solving and logical thinking. Schools 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 provides 70% of the funding for each student with the rest of the funds provided by the school, on the condition that a certain number of students will be attending the programme. Teachers are also required to attend a training course about the programme conducted by the technology vendors. IMDA aims that the programme would be taught by the trained teachers in the future. Currently, each 10-h session is conducted by the technology trainers in the respective school lab. In each session, students are introduced to computing concepts such as the use of variables and conditional statements, and on how to use a visual programming tool such as Scratch. Students also use robotic tools such as the Lego WeDo kits, or the MoWay robot based on the proposal by the different training partners. Schools can choose different tools offered by the various trainers based on their students' interest and budget.

The Code for Fun enrichment and Playmaker programme are part of the Code@SG movement initiated by the government to teach CT and programming to students from an early age. Driven by the IMDA, the initiative is important for building Singapore's national capacity of 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 competitions. Previously, students in the school Infocomm clubs were responsible for operating the school's audiovisual and computer

www.dbooks.org

equipment during events. The clubs are now organised such that students now have more opportunities to apply their creativity with computers through programming and digital media. IMDA and the Information Technology Standards (ITSC) committee also organises an annual programming competition CodeXtreme for students, supported by educational institutions such as the universities, polytechnics, the Singapore Science Centre and technology industry partners such as RedHat and PayPal. Students are encouraged to participate in CodeXtreme in which Primary school students can work in teams to complete a given project with Scratch. Each team has an adult as a supervisor and mentor. Prior to the competition, the students must attend a workshop to equip themselves with the necessary skills for the challenge of the hackathon.

19.3.3 Secondary Schools

In 2017, the Ministry of Education introduced a new Computing subject which will be offered to students as an 'O' Level subject. It replaced the existing Computer Studies subject (MOE, 2017). Students taking the subject would learn to code in Python, a programming language which was previously taught only at 'A' Level Computing. In the new syllabus design, students will develop CT and programming 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.

The new Computing syllabus is built on the framework shown in Fig. 19.2: (1) Computer as a Science; (2) Computer as a Tool; and (3) Computer in Society.

The dimension of Computer as a Science is comprised 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 programming. Using both CT skills and systems thinking, students are required to work on a project of their own interest. This is, however, a non-assessment component of the programme. It is to encourage students to take more ownership by identifying a problem that they have an interest and develop ideas to solve the problem using programming tools. The purpose of a non-assessed project work is to encourage the students to be more creative in designing solutions without the pressure of assessment. 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 twenty-first 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.

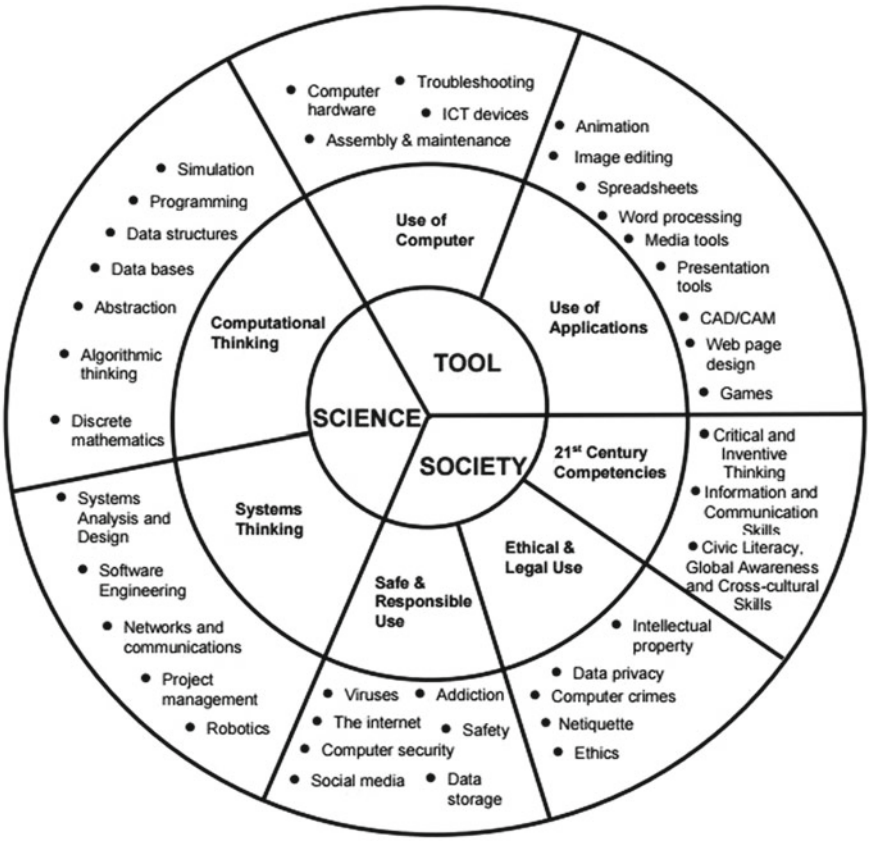


Fig. 19.2 Computing education framework

Currently, 22 out of about 150 secondary schools (15%) are offering Computing as an O level subject. One of the reasons for the low number of schools is the lack of teachers who can teach computing. There are relatively few teachers who have Computing or Computer Science background who can teach programming. Teachers who are interested in teaching Computing and programming attend a year-long conversion course taught by Computer Science faculty from a university. They were given time by their schools to attend the course during school hours. In the course, they upgraded their Computer Science content knowledge such as data structures and programming in Python. 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 workshops, teachers also experienced the use of the tools for teaching such

www.dbooks.org

as the Raspberry Pi. The workshop also served as a platform for teachers to raise their concerns about teaching the subject, for example about how the project work for students can be organised and implemented in the school.

In most schools, the typical number of students offered Computing ranges from 15 to 30 students. Computing is offered as an elective subject for the O levels and student can opt-in for the subject at the end of Secondary 2. A student is typically offered 7–8 'O' Level subjects in their Secondary 3 year. Subjects like Additional Mathematics and the Sciences are more favoured by students as these subjects are better considered for admission for pre-university junior colleges and polytechnics. Hence, students who are initially interested in taking up Computing may instead choose to take Additional Mathematics, because of the limit in the number of subjects they are offered and for pragmatic reasons for entry to pre-university courses.

19.4 Singapore's Approach in Computing

Unlike countries like Finland, England and Korea, Singapore is not including Computing or CT as compulsory education. Instead, Singapore' approach is to provide opportunities for students to develop their interests in programming and computing skills through touchpoint activities at various ages, as shown in Fig. 19.3. Computing and CT skills, which are introduced to the children are age-appropriate and can engage them in learning. Children can then progressively develop interest and skills, leading them to select Computing as a subject in the 'O' levels. The following sections describe the characteristics of the approach.

19.4.1 Opt-in by Schools

The decision to teach Computational Thinking skills and computing is decided by individual schools. An opt-in model respects the choice of each school to select programmes according to the needs of their students and the readiness of the teachers. Schools are encouraged to develop a niche in skills or sports, creating a diverse learning landscape to meet various educational needs and interest of students in Singapore. 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 a pivotal role in implementing the programmes, there must be buy-in from the teachers who can see the importance of the programmes for the students. For the schools to opt-into adopt computing, there must be teachers within the school to be ready to learn, experiment, and implement the programme. One such example of opt-in is Bukit View Secondary School, Singapore. The school has formulated a programme that enables all students to learn computational thinking. The school addressed the issue of a packed curriculum by structuring time for learning computing in the time table and integrating with school subjects. Training partners are engaged to work

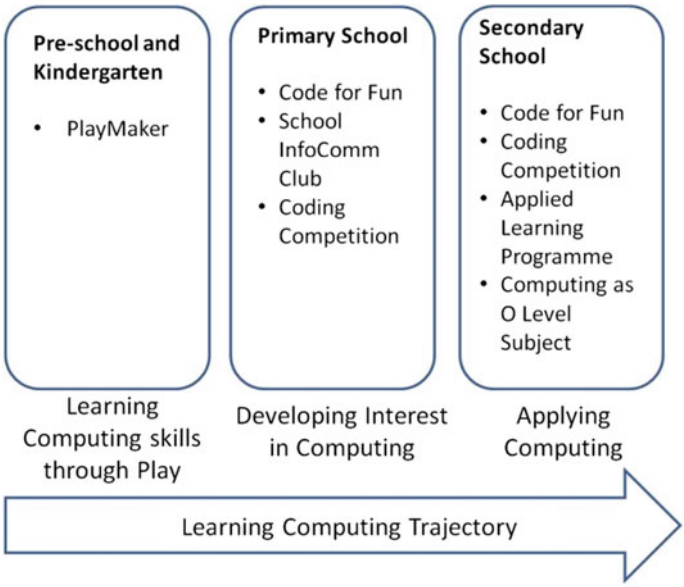


Fig. 19.3 Learning computing in Singapore

with teachers in designing and integrating computing with the subjects to address the lack of teachers' experience. Teachers' capacity is continuously developed as they gain competence in computing. The school iteratively improve the programme for sustainability and richer learning experience for the students (Hameed et al., 2018)

19.4.2 Nurturing Interest in Computing

Instead of making computing part of compulsory education, an approach is to nurture interest at an early age. Preschool children are developing problem-solving and logical thinking skills through play. Toys like the Beebot and Circuit Stickers are age-appropriate. Young children are engaged through 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 to have fun and engagement while learning programming and developing logical thinking skills. As students progress up to the O Levels, they can select Computing as a subject based on their interest and choice. Starting from preschool, a pipeline is created for students to develop interest and computational thinking skills; enabling them to choose Computing rather than making its learning compulsory.

www.dbooks.org

19.4.3 Upskilling Teachers in Teaching Computing

To prepare teachers to develop logical thinking, algorithmic thinking, problem-solving and programming skills, professional development and support must be given. Professional development should be appropriate to the needs of the teachers; to prepare them to think about how they can teach their students. For the preschool teachers, a form of learning is for them to experience playing with the toys and in turn, understand how their students 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 schools, computing teachers undergo an intensive computing course learning in computer science concepts and programming. Most of these teachers are non-computer science graduates but they volunteer for the conversion course based on their own interests. Even after the course, these teachers hold regular meetups to continue improving their knowledge in teaching computing.

In the National Institute of Education, researchers are working with teachers to develop and design pedagogies for Computation Thinking and Computing Education. One such effort is the development of unplugged approaches for teaching computational thinking in the schools (Looi, How, Wu, Seow, & Liu, 2018). Researchers conduct workshops for computing teachers to experience various unplugged activities such as sorting with a balance scale, binary cards and deadlock avoidance (Fig. 19.4). Through the workshop experiences, teachers appreciate the role of kinaesthetic activities and concrete materials play in helping students to understand some of the key concepts in Computing. Appropriate unplugged computing activities that can be mapped to the computing curriculum are identified and designed. These unplugged computing activities are enacted in the classroom in collaboration with the teachers. Data regarding student learning outcomes from the unplugged activities are collected, analysed, and shared with the teachers. Using a collaborative design-based research approach, the researchers aim to bridge the gap of research and practice that can improve teaching practices in computing.

19.4.4 Multi-agency and Many Hands Approach

The task of building CT and Computing skills requires the combined effort of multiple agencies working together. They include the government agencies like Infocomm Media Development Authority (IMDA), Ministry of Education and the Ministry of Social and Family Development, education centres like the Singapore Science Centre, universities and educational training providers. These agencies have been working together, sometimes also independently, to organise opportunities for students to learn computational thinking skills by providing them with varied experiences. These agencies can pool resources such as funding and support for initiating, implementing and sustaining the programmes.



Fig. 19.4 Unplugged computational thinking activity with pre-service teachers

IMDA started the Digital Maker programme to develop a peoples' passion to create with technology and grow a community of digital creators. In 2017, IMDA introduced the micro:bit board, a pocket-sized programmable device, to the community including schools, teachers and students. Programmes and workshops are organised with training partners to explore the possibilities of digital making with the micro:bit boards. In schools, students can use the micro:bit boards to develop various real-world physical applications such as creating games, controlling a car, and measuring the height of the flag pole. In workshops for teachers, they learn the basics of using the micro:bit and how it can be used as a learning tool in the classroom. If teachers see value in the use of the boards and would like to receive a set of micro:bit boards for their school, they can submit a lesson plan to IMDA to provide details about how they plan to integrate the micro:bit boards in their lessons. Schools can engage training partners to run the additional programmes or train more teachers in the use of the micro:bit boards. The above example shows how different partners can work together in an ecosystem to teaching Computing in the schools. In the next section, we will elaborate on this ecosystem in computing education.

19.5 Towards an Ecosystem of Learning Computing

A learning ecosystem is defined as a network of relationships among learning agents, learners, resources, and assets in a specific social, economic and geographic context (Prince, Saveri, & Swanson, 2015). Learning for students is not limited by the boundaries of the classroom and lessons taught by teachers but facilitated through a network

www.dbooks.org

of resources and agents to develop learning pathways based on the interest of learners. Students have opportunities to informal and formal resources in the learning ecosystem (Basham, Israel, & Maynard, 2010). They can participate in enrichment programmes, special-interest clubs and after-school activities; have access to a myriad of digital based media such as videos, social media and Internet. In the learning ecosystem, each agent plays an important role in the ecosystem in initiating and sustaining the learning interest of learners. Compared to the traditional classroom where tightly bound relationships and resources are the nexus; the flow to deliver instruction, develop curriculum, perform assessment, teaching, and learning can now be available, through and enhanced by a vibrant learning ecosystem of resources for students.

Bell, Curzon, Cutts, Dagiene, and Haberman (2011) explain that informal outreach programmes which downplay syntax-based software programming as a prerequisite skill for engaging with Computer Science ideas can effectively make CT concepts accessible to the students in short bursts without formal curriculum support. These outreach programmes can operate outside, or in conjunction with the formal education system to expose students to computer science concepts, so that they can make informed decisions about their career paths. Kafai and Burke (2013) observe that developments in K-12 computer software programming education can be characterised by a 'social turn', a shift in the field in which learning to code has shifted from being a predominantly individualistic and tool-oriented approach to now one that is decidedly sociologically and culturally grounded in the creation and sharing of digital media. The three dimensions of this social turn are: (1) from writing code to creating applications, (2) from composing 'from scratch' to remixing the work of others and (3) from designing tools to facilitating communities. These three shifts illustrate how the development of artefacts, tools and communities of programming could catalyse the move from computational thinking to computational participation, and hence broaden the students' participation in computing. One of the most active proponents of this social turn is the Maker Movement (Martin, 2015), a community of hobbyists, tinkerers, engineers, hackers, and artists who creatively design and build projects for both playful and useful ends. There is a growing interest among educators in bringing 'making' into K-12 education to enhance opportunities to utilise CT concepts to engage in the practices of engineering, specifically, and STEM more broadly. CT is considered the best practice for teaching computing and more broadly to solve problems and design systems; however, as computing extends beyond the desktop, Rode et al. (2015) argue we could transition from CT to computational making as an educational framework via the Maker Movement. Besides using micro:bit, many people in the Maker Movement also use the LilyPad Arduino, a fabric-based construction kit that enables novices to design and build their own soft wearables and other textile artefacts. (Buechley, Eisenberg, Catchen, & Crockett, 2008; Kafai et al., 2013; Kafai, Lee, Searle, & Fields, 2014).

In Singapore, students can choose to participate in the Learning Ecosystem for Computing, which is comprised of informal education programmes such as Code for Fun, computing-related Core Curricular Activities (CCA) such as the Computer and Robotics club, and in computer-based Applied Learning Programmes (ALP). Nevertheless, there is a dearth in the extant literature about how these informal learning educational programmes could help to sustain the students' interests in the development of CT after they have progressed through them. Low (2014) notes that while ALP holds promise in achieving authentic achievements as a long term and sustainable form of learning that can positively influence student learning, equip students with twenty-first century competencies and ensure positive future outcomes, there is a need for greater scholarship discourse in areas of authentic assessment and authentic achievement and their roles in the enactment of ALP.

The various programmes provide varied experiences for the students to develop their own interests and apply computing to solving problems. The programmes are facilitated, developed and implemented by stakeholders in the ecosystem as shown in Fig. 19.5. The agents in the ecosystem directly involved with students include teachers, school leaders and parents. External agents include the government agencies such as IMDA, Ministry of Education, the Singapore Science Centre and the computing trainers. The range of programmes for Computing includes formal Computing education in the O and A level computing; informal programmes in schools such as Code for Fun, Play/Digital Maker, Computing-based CCAs and ALP; and the outside-school enrichment programmes such as Computing Educational Centres or the Singapore Science Centre (Fig. 19.5).

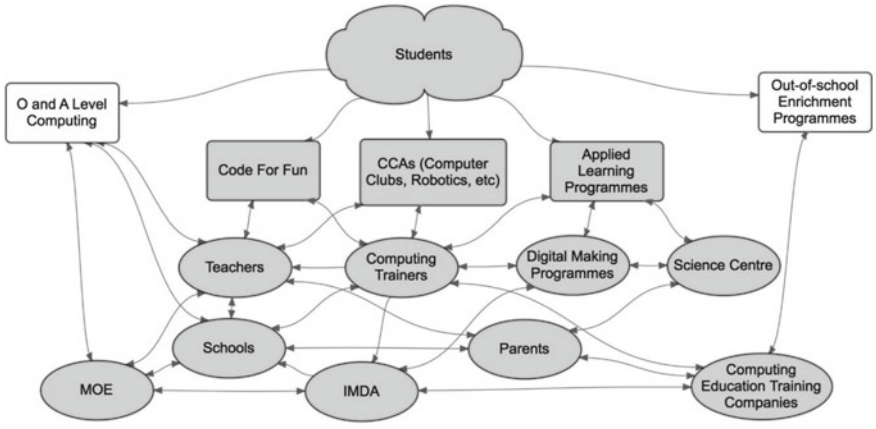


Fig. 19.5 Ecosystem for learning computing in Singapore

www.dbooks.org

Currently, computing in formal education is offered at 'O' Level and 'A' Level curriculum. Pixel labs under Code@SG movement (Pixel Labs, 2017) aims to see programming and CT as Singapore's national capability. Accordingly, programming and CT are being considered to teach from an early age to students. Children with special needs and the underprivileged are also encouraged to cultivate their interest in programming, programming and technology development. The Infocomm Clubs (Infocomm Club, 2017), a co-curricular activity programme for school children not only excites students about Infocomm in a fun and meaningful way but also cultivates leadership and entrepreneurship capabilities along with providing opportunities for project work and community service projects at an early age. Many initiatives (Code in Community, 2017) offers free programming lessons to underprivileged children in Singapore. Thus, the main approach is to enthuse a broad base of students in computing and expose them to possibilities of technology through enrichment programmes and co-curricular activities. Learning to code is part of the ALP in 41 secondary schools. In addition, MOE also partners IMDA to provide enrichment programmes like the 'Code for Fun' and 'Lab on Wheels', which have been well received by schools. There are secondary schools (33 schools in 2016) with Infocomm clubs, which tap on the support of IMDA to provide learning in areas involving programming such as app development and robotics.

In the National Institute of Education, efforts have been made to bring together researchers, teachers, computing education trainers, and students together to share ideas and practices in the learning of Computing and Computational Thinking. In 2017, a half-day symposium and workshop was organised within a major international education conference with representatives from Infocomm Media Development Authority (IMDA), National Institute of Education (NIE), National University of Singapore (NUS), school teachers and students presented national-level initiatives on Computing Education, research findings, teaching and learning experiences in Computing. A workshop provided attendees to participate in various Computing learning activities such as unplugged activities, board game playing and physical computing with micro:bit and Arduino (Fig. 19.6). We plan to continually organise more of these events to draw participants in the ecosystem together in the sharing, learning and building of a vibrant community for Computing Education in Singapore.

19.6 Summary

Singapore has taken a pragmatic approach in the implementation of CT skills and computer science education. Taking such an approach provides children with the 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 schools, students learn through fun and are given opportunities to extend their interest in programming through clubs and programming competition. At the secondary school, students can choose whether they wish to pursue Computing as a subject. Schools can opt-into offer programmes based on the students'



Fig. 19.6 Workshop with computing learning activities such as physical computing with micro:bit, card games and unplugged activities to learn computing concepts

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 computing learning. Thus, Singapore's approach to CT education is promising but different from various countries that have made CT part of compulsory education.

References

APFC. (2017). *Preparing students for South Korea's creative economy: The successes and challenges of educational reform.* Retrieved February 13, 2017, from http://www.asiapacific.ca/research-report/preparing-students-south-koreas-creative-economy-successes.

Basham, J. D., Israel, M., & Maynard, K. (2010). An ecological model of STEM education: Operationalizing STEM for all. *Journal of Special Education Technology, 25*(3), 9–19.

www.dbooks.org

Bell, T., Curzon, P., Cutts, Q., Dagiene, V., & Haberman, B. (2011). Overcoming obstacles to CS education by using non-programming outreach programmes. In *Informatics in schools: Contributing to 21st century education* (pp. 71–81). Springer. https://doi.org/10.1007/978-3-642-24722-4.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). *The Nordic approach to introducing computational thinking and programming in compulsory education*. Report prepared for the Nordic@BETT2018 Steering Group. https://doi.org/10.17471/54007.

Buechley, L., Eisenberg, M., Catchen, J., & Crockett, A. (2008). The LilyPad Arduino. In *Proceeding of the Twenty-Sixth Annual CHI Conference on Human Factors in Computing Systems—CHI '08* (p. 423). https://doi.org/10.1145/1357054.1357123.

Code in Community. (2017). Google's free coding classes are proving to be a major success—Sees first 500 S'pore kids graduate. Retrieved May 22, 2017, from https://vulcanpost.com/611220/code-community-google/.

DfE. (2017). *National curriculum in England: Computing programmes of study*. Retrieved February 13, 2017, from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study.

Hameed, S., Low, C.-W., Lee, P.-T., Mohamed, N., Ng, W.-B., Seow, P., & Wadhwa, B. (2018). *A school-wide approach to infusing Coding in the Curriculum* (pp. 33–36). Paper presented at the Proceedings of 2nd Computational Thinking Education Conference 2018, Hong Kong SAR, The Education University of Hong Kong.

IMDA. (2017). *PlayMaker changing the game.* Retrieved February 13, 2017, from https://www.imda.gov.sg/infocomm-and-media-news/buzz-central/2015/10/playmaker-changing-the-game.

InfoComm Club. (2017). *Infocomm Clubs programme*. Retrieved February 13, 2017, from https://www.imda.gov.sg/imtalent/student-programmes/infocomm-clubs.

Japan Times. (2017). *Computer programming seen as key to Japan's place in 'fourth industrial revolution'*. Retrieved February 13, 2017, from http://www.japantimes.co.jp/news/2016/06/10/business/tech/computer-programming-industry-seen-key-japans-place-fourth-industrial-revolution/#.WKG2P_l97b0.

Kafai, Y. B., & Burke, Q. (2013). The social turn in K-12 programming. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education—SIGCSE '13* (p. 603). https://doi.org/10.1145/2445196.2445373.

Kafai, Y. B., Searle, K., Kaplan, E., Fields, D., Lee, E., & Lui, D. (2013). Cupcake cushions, scooby doo shirts, and soft boomboxes: e-textiles in high school to promote computational concepts, practices, and perceptions. In *Proceedings of the 44th ACM technical symposium on Computer Science Education* (pp. 311–316). ACM.

Kafai, Y. B., Lee, E., Searle, K., Fields, D., Kaplan, E., & Lui, D. (2014). A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Transactions on Computing Education (TOCE), 14*(1), 1.

Looi, C. K., How, M. L., Wu, L., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Journal of Computer Science Education* (JCSE). https://doi.org/10.1080/08993408.2018.1533297.

Low, J. M. (2014). Applied learning programme (ALP): A possible enactment of achieving authentic learning in Singapore schools. In *40th Annual Conference (2014)*. Singapore: International Association for Educational Assessment. Retrieved from http://www.iaea.info/documents/paper_371f25129.pdf.

Martin, L. (2015). The promise of the maker movement for education. *Journal of Pre-College Engineering Education Research (J-PEER), 5*(1). https://doi.org/10.7771/2157-9288.1099.

MOE. (2017). *O level computing teaching and learning syllabus*. Retrieved February 13, 2017, from https://www.moe.gov.sg/docs/default-source/document/education/syllabuses/sciences/files/o-level-computing-teaching-and-learning-syllabus.pdf.

Pixel Labs. (2017). *CODE@SG movement—Developing computational thinking as a national capability*. Retrieved October 10, 2017, from https://www.imda.gov.sg/industry-development/

programmes-and-grants/startups/programmes/code-sg-movement-developing-computational-thinking-as-a-national-capability.
Prince, K., Saveri, A., & Swanson, J. (2015). *Cultivating interconnections for vibrant and equitable learning systems*. KnowledgeWorks: Cincinnati, OH, USA.
Rode, J. A., Weibert, A., Marshall, A., Aal, K., von Rekowski, T., El Mimouni, H., & Booker, J. (2015, September). From computational thinking to computational making. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 239–250). ACM.
Smart Nation. (2017). *Why smart nation*. Retrieved February 13, 2017, from https://www.smartnation.sg/about-smart-nation.
Sullivan, A., & Bers, M. U. (2017). Dancing robots: Integrating art, music, and robotics in Singapore's early childhood centers. *International Journal of Technology and Design Education,* 1–22.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.
Wing, J. M. (2017). *Computational thinking, 10 years*. Retrieved December 13, 2017, from https://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later/.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Chapter 20
Teaching-Learning of Computational Thinking in K-12 Schools in India



Sridhar Iyer

Abstract Schools in India have been offering computer studies as a subject for the past two decades. However, there is wide variation in the choice of topics, as well as treatment of a given topic. Typically, the focus is on usage- and skill-based content for specific applications. There is very little emphasis on important thinking skills of broad applicability, such as computational thinking and 21st century skills. This chapter describes a 10-year long project, called Computer Masti, to integrate thinking skills into computer studies. The project includes: curriculum across K-12 grades, textbooks that contain explicit chapters on thinking skills in each grade, and teacher training so that teachers gain proficiency in teaching this content. This chapter provides an overview of the rationale and content of the curriculum, examples of how computational thinking skills are addressed in the textbooks and learning activities, summary of its implementation in schools in India, and some results of evaluation studies.

Keywords Thinking skills · Computational thinking · Computer studies · K-12

20.1 Introduction

20.1.1 Computational Thinking Perspectives

There are diverse perspectives of computational thinking (Papert, 1980; Wing, 2006; diSessa, 2000; Barr & Stephenson, 2011; Brennan & Resnick, 2012). The International Society for Technology in Education and Computer Science Teachers Association (ISTE & CSTA, 2011) provides an operational definition of computational thinking for K-12 education as

S. Iyer (✉)
Department of Computer Science and Engineering, & Interdisciplinary Programme in Educational Technology, Indian Institute of Technology Bombay, Mumbai, India
e-mail: sri@iitb.ac.in

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_20

"Computational Thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem-solving process to a wide variety of problems."

An important goal of developing computational thinking is to use the above elements to solve problems in various disciplines and contexts, including but not limited to programming.

20.1.2 Computer Studies in K-12 in India

Schools in India are affiliated to one of many Boards (CBSE; ICSE; IB; IGCSE; State Board). Each Board prescribes curricula and conducts standardized examinations for grades 10 and 12. Schools have been offering computer studies as a subject to their students for the past two decades. However, there is wide variation in the choice of topics, as well as treatment of a given topic. Typically, the focus is on usage- and skill-based content for specific applications. There is very little emphasis on important thinking skills of broad applicability, such as computational thinking and twenty-first-century skills.

20.1.3 Computer Masti Project

To address the above issues, the Computer Masti project was initiated at IIT Bombay in 2007.

The project has three aspects: (i) Defining the curriculum, called CMC, (ii) Developing the textbooks, called Computer Masti, and (iii) Supporting schools to implement the curriculum. Developing thinking skills is a key focus across all these aspects.

CMC (Iyer et al., 2013) has explicit emphasis on thinking skills, i.e., the basic procedures and methods used in making sense of complex situations and solving problems. Thinking skills, such as algorithmic thinking, problem-solving, systematic information gathering, analysis and synthesis, and multiple representations, are mapped to various grades. Computer applications and usage skills are introduced only after motivating the need for developing the corresponding thinking skill.

www.dbooks.org

Computer Masti textbooks (Iyer, Baru, Chitta, Khan, & Vishwanathan, 2012) have explicit chapters for developing thinking skills across a variety of real-life contexts. For example, students are introduced to computational thinking in grade 3 through activities such as planning a picnic, and then go on to programming in Scratch (Scratch). The teacher training support provided to schools has explicit emphasis on the need for thinking skills, their applicability across subjects and grades, and how they may be developed. Teachers are trained to go beyond teaching computer usage skills and focus on thinking skills. In 2018, Computer Masti is being implemented in 700+ schools, across India. Over the years, Computer Masti has been used by more than 1.5 million students in India. The next section in this chapter provides an overview of CMC curriculum. It is followed by examples of how thinking skills are addressed in Computer Masti textbooks. The subsequent sections provide a summary of its implementation in schools, and results of evaluation studies.

20.2 CMC Curriculum

20.2.1 Underlying Philosophy

The underlying philosophy of CMC (Iyer et al., 2013) is as follows:

- *Develop computer fluency, not just computer literacy.*

While computer literacy is defined as the knowledge and ability to use computers and related technology efficiently, computer fluency means a robust understanding of what is needed to use information technology effectively across a range of applications (CITL, 1999). The goal of computer fluency is to enable students to adapt to the changes in digital world, rather than get them to be only users of specific computer applications. Hence CMC approaches the learning of computer usage skills in the context of learning about fundamental concepts and thinking process skills. CMC has emphasis on understanding of the concepts behind various computer-based activities, rather than usage of specific tools. The goal of such a concept-oriented approach is to equip students to be self-learners and enable them to cope with the inevitable advent of new versions, tools, and technologies of the future.

- *Develop thinking process skills, not just content mastery.*

While content mastery is important, the need to develop thinking process skills is well established (Padilla, 1990; Big6). There exist recommendations from professional bodies and accreditation boards such as ACM and ABET (ACM; ABET) that thinking skills are required by college graduates. Students do not automatically acquire these skills, while learning content (Marzano et al., 1988). Hence, CMC

has explicit emphasis on teaching-learning of thinking process skills, which are the basic procedures and methods used in making sense of complex situations, solving problems, conducting investigations and communicating ideas. CMC explicitly addresses the thinking skills of: Algorithmic thinking, problem-solving, systematic information gathering, brainstorming, analysis and synthesis of information, multiple representation and divergent thinking. Computer literacy skills are introduced only after motivating the need for developing the corresponding thinking skill.

- *Highlight the interconnectedness of knowledge, not just address a topic in isolation.*

While mastery of a topic is important, recognizing the interconnectedness of various topics and ideas leads students to construct a more expert-like knowledge structure (Ellis & Stuen, 1998). Hence, CMC has emphasis on: (i) thematic integration, i.e., the integration of knowledge from various subjects into computer studies, and the use of computer-based activities to strengthen knowledge in other subjects, and (ii) spiral curriculum, i.e., the content of the curriculum is organized such that themes and topics are revisited with increasing depth in each successive grade.

20.2.2 Computational Thinking Skills in CMC Across the Grades

A subset of the thinking skills from CMC (Iyer et al., 2013), which correspond to items from the operational definition of computational thinking (ISTE & CSTA, 2011), is provided in Table 20.1. Each row shows how a particular thinking skill is gradually developed and advanced across grades 3–8.

For example, students begin to learn algorithmic thinking in grade 3, where they identify and sequence the steps involved in carrying out simple activities at that grade, such as walking like a robot. In grade 4, they learn to identify constraints, use of branching and iteration, to design solutions that meet the constraints. In grades 5 and 6, they learn to gather requirements, synthesize information, use of representations, decision-making, and apply their learning to tasks such as digital story-telling. In grades 7 and 8, they apply their learning to more complex tasks such as designing an app for senior citizens.

Algorithmic thinking skills are strengthened through Scratch programming in grades 3–6, where students apply concepts of if-else, loops, event handling, input, variables and lists. In grades 6–8, they move to writing flowcharts and programming using a procedural language.

www.dbooks.org

Table 20.1 Thinking skills in CMC across grades

Grade 3	Grade 4	Grade 5	Grade 6	Grade 7	Grade 8
***1. Problem-solving**: algorithmic thinking, logical reasoning and decision making*					
• Break up a non computer-based activity into steps • Identify the main steps or sub-tasks in order to carry out an activity • Identify the sequence of detailed steps required to carry out each sub-task	• Identify goals and constraints of given problem • Apply algorithmic thinking to solve • Write steps to perform given task	• Identify known and unknown information to solve a given problem • Identify problem constraints • Identify solutions that meet the constraints	• Identify and choose between multiple solutions • Identify necessary and sufficient conditions that a solution should satisfy • Apply reasoning, decision making to make choices in performing real-life tasks	• Systematically divide tasks into sub-tasks • Solve increasingly complex problems • Evaluate pros and cons of multiple solutions and decide among the various options • State assumptions under which a solution works	• Apply all aspects of thinking skills from previous grades
***2. Logical data handling**: gathering, organizing, analyzing and synthesizing information*					
• Grouping related objects together • Create and organize folders	• Organize given information into hierarchy • Use folders and sub folders	• Identify resources that can provide information to accomplish a task • Gather information for given task • Organize and synthesize information to accomplish task	• Synthesize given information • Use of multiple classification schemes • Compile and present reports	• Conduct surveys of users to gather data for research • Present gathered information in a coherent manner • Use of multiple representations	• Apply multiple thinking skills together to solve a given problem

(continued)

Table 20.1 (continued)

Grade 3	Grade 4	Grade 5	Grade 6	Grade 7	Grade 8
3. Multiple representations of data					
		• Represent given information is Lists or Tables, based on the requirement	• Represent given information using lists, tables, charts, graphic organizers	• Use multiple representations to analyze data and convey results	• Integrate different data forms (table, charts, picture) from different applications

20.3 Computer Masti Books

Computer Masti books (Iyer et al., 2012), bridge the gap between prescriptive curriculum and the teacher's need to transact it in the class. The books are labeled as Level I–VIII, which can be mapped onto grades 1–8. They are sequenced in such a way that students who have studied them from grades 1–8 would be able to meet the requirements of various education boards of the country (CBSE, ICSE, IB, State boards) for grade 9 onwards. These books are released under Creative Commons license (CC) and can be freely downloaded (Iyer et al., 2012).

20.3.1 Pedagogical Approach

The Computer Masti books provide learning activities to address the thinking skills in CMC. These books use real-world context, analogies and a narrative structure to show the broad applicability of thinking skills not just in other subjects but also in "real life".

The books use pedagogical approaches that are established to be effective for learning of thinking skills. The approaches used are:

- *Inquiry-based learning*: Inquiry-based learning (Barrett, Mac Labhrainn, & Fallon, 2005; Olson & Loucks, 2000) is an approach in which students are actively engaged in the learning process by asking questions, interacting with the real world, and devising multiple methods to address the questions. For example, the teacher may ask students to prepare a presentation on an unfamiliar topic, such as maintaining an aquarium or embedded applications of computers. Students first identify what they already know about the topic and what they need to find out. Then they gather the relevant information from various sources, and synthesize it to create the presentation. The teacher oversees the process, provides feedback and intervenes wherever necessary. Detailed examples are given in Sect. 20.3.3.

www.dbooks.org

- *Learning* via *Real-world Context*: Learning is effective when it is situated in a real-life context (Bransford, Sherwood, Hasselbring, Kinzer, & Williams, 1992). Making real-world connections to the content, teaching abstract principles by establishing the need for them in a real-life context, and using analogies from students' everyday lives, are practical methods to situate learning. For example, the teacher may ask students to conduct a market survey, before introducing the concepts and skills in the use of Spreadsheets. Students learn data analysis, interpretation and multiple representations, using real data. More details are given in Sect. 20.3.3.
- *Collaborative learning*: Students learn more of what is taught, retain knowledge longer, are able to solve more complex problems, and are more satisfied with the process when they learn in groups (Johnson & Johnson, 1998; Totten, Sills, Digby, & Russ, 1991). For example, the teacher may ask students to work in pairs for programming activities, or in larger groups to create multimedia reports on field trips.
- *Playfulness*: Play has been found to be a developmentally appropriate way for children to learn since it facilitates problem-solving, perspective taking, social skills, and development of the mind (Bailey, 2002). The Computer Masti books use playfulness as an explicit basis for designing learning activities. For example, lessons are structured as conversations between two students and a facilitator, which a teacher may enact in class.

20.3.2 Structure

Each Computer Masti book contains lessons, worksheets and activities based on the themes in the CMC curriculum recommended for that particular grade. Each book has a "Concept before Skill" approach focusing on conceptual understanding before learning the usage skills associated with specific applications.

Each lesson has a narrative style, consisting of two children who learn about computers while they engage in their daily activities. A third character in the narrative, a mouse look-alike, plays the role of a facilitator for inquiry-based learning. Students make real-world connections and integrate knowledge via the context in the narrative. Worksheets, activities and projects in each lesson are geared towards exploration, collaborative learning, and reflection. Each lesson also has a 'Teacher's Corner' that provides guidelines for transacting the lesson as per the pedagogy recommended.

There are explicit lessons on stepwise thinking in grade 3, on logical thinking in grade 4, gathering and synthesizing information in grade 5, and using multiple representations in grade 6. Students are introduced to computer applications only after they learn the corresponding thinking skills in a familiar context. For example, they are introduced to Internet search only after they learn the thinking skill of systematically gathering information and concepts related to organizing information. They are formally introduced to programming only after they apply algorithmic processes to daily life situations, such as planning a school play.

20.3.3 Example Learning Activities

This section provides some illustrative examples of learning activities in Computer Masti for addressing thinking skills. These correspond to the computational thinking aspects shown in Table 20.1, viz., problem-solving, logical data handling, and multiple representations of data.

20.3.3.1 Problem-Solving: Algorithmic Thinking

Figure 20.1 shows an example of algorithmic thinking from grades 3 and 4. The facilitator (Moz) poses a problem of planning a picnic. The students (Tejas and Jyoti) perform stepwise thinking to solve the problem by first listing the main sub-tasks, then working out the details in each sub-task, and the sequencing of the steps. The broad applicability of such thinking is reinforced by enclosing them in a "Concept box" as shown. The corresponding "Teacher's Corner" makes recommendations such as "*Ask them to give other examples where they follow steps to do the activity. For instance, ask them to plan a birthday party for their classmate. What steps are involved? Ask them to arrange all the steps they need to do, in proper sequence, so that the party turns out*

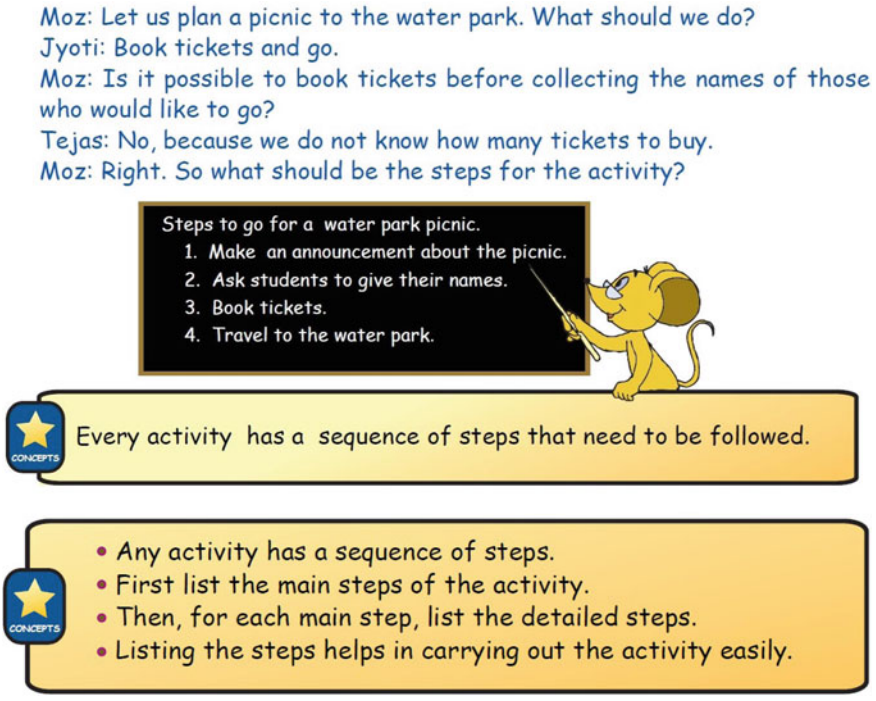


Fig. 20.1 Introduction of stepwise thinking in grade 3

www.dbooks.org

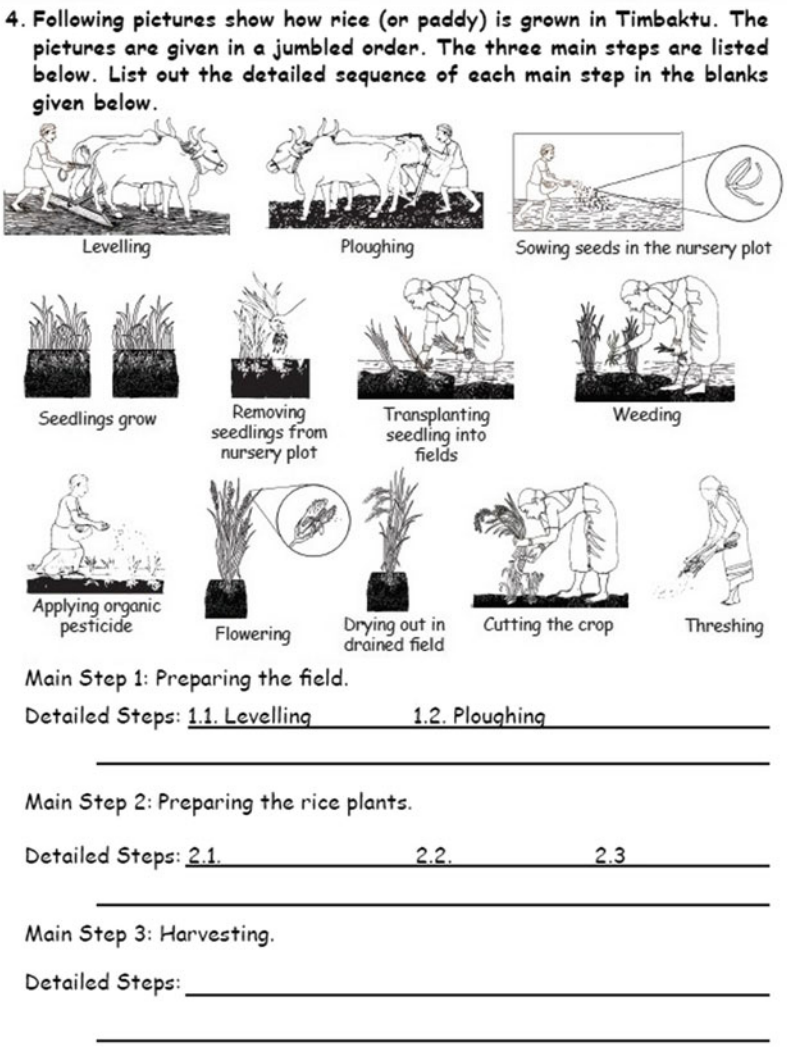
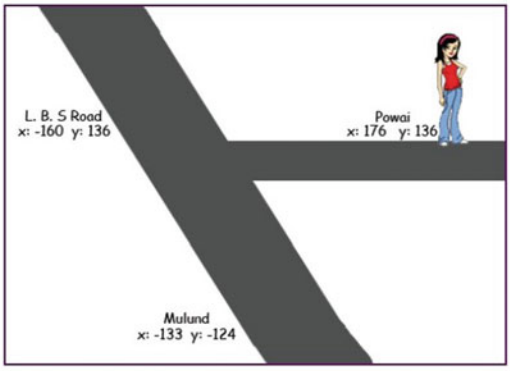


Fig. 20.2 Worksheet for applying stepwise thinking

to be well-organized. Ask them to identify the main steps and list the detailed steps." The students are given practice to apply the thinking through worksheets having real-life scenarios, as shown in Fig. 20.2. This is followed by stepwise thinking with conditions and branching, and connecting it to Scratch programming in grade 4, as shown in Fig. 20.3.

In subsequent grades students apply their learning from programming activities to real-life problem-solving. For example, in grade 7, they learn to use flowcharts while programming, through activities such as "Draw a flowchart to find the tallest

1. Shilpa is at Powai. She has to go via L.B.S Road to Mulund. The x-value and y-value shows positions of Powai, L.B.S Road, Mulund are given below. Write a program in Scratch for Shilpa to go from Powai to Mulund via L.B.S Road.



Hint: The starting instructions are given below. Complete the program and run it in Scratch.

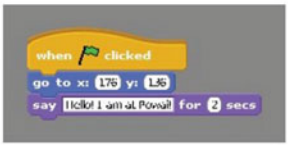


Fig. 20.3 Activity for applying stepwise thinking and programming a scenario

student in a group of 5 students". Then, they are asked to apply their knowledge of flowcharts to solve a real-life problem such as "You have to design the control mechanism for the crossing of a railway track and a road. Draw a flowchart for the functioning of the crossing gate to ensure that there are no accidents."

20.3.3.2 Logical Data Handling: Gathering and Synthesis

Figure 20.4 shows an example of gathering and synthesis of information from grade 5. The facilitator (Moz) poses a problem of setting up a home aquarium. The students (Tejas and Jyoti) apply thinking skills to solve the problem by analyzing the goal, identifying the requirements of sub-tasks, gathering the information, and categorizing it. This is followed by identifying constraints and decision-making, as shown in Fig. 20.5. Then, they create a Scratch program for a virtual aquarium, as shown in Fig. 20.6. The learning of the thinking skills is reinforced through worksheets and activities. The "Teacher's Corner" makes recommendations such as "*Ask them to plan an outdoor summer camp for the class using the thinking skills from this lesson.*

www.dbooks.org

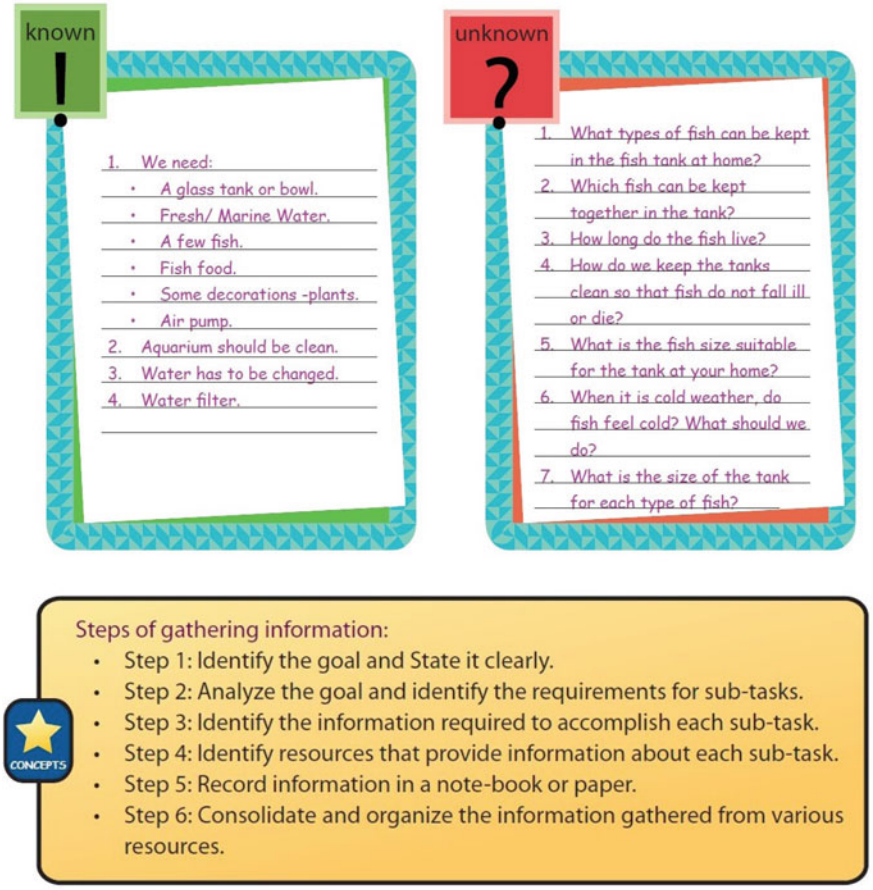


Fig. 20.4 Gathering information systematically for a problem-solving scenario

Ensure that they explicitly identify the goal and analyze the requirements, gather and consolidate information, and identify constraints and do decision-making."

In subsequent grades, integration of multiple concepts is explicitly targeted. Students apply their learning to complex problem-solving. For example, in Grade 8, they do activities such as "Design a mobile app for senior citizens". First, they create surveys for identifying needs. To do this, they may interview senior citizens in their neighborhood to identify events that need alerts, such as taking medicines on time. Then, they identify the features that are required by most of the participants. To do this, they may use spreadsheets for recording data, counting frequencies, drawing graphs and so on. Finally, they design their product. To do this, they may create mind-maps for categorizing features to be included, flowcharts to depict the functioning of their app, and a presentation to market their product.

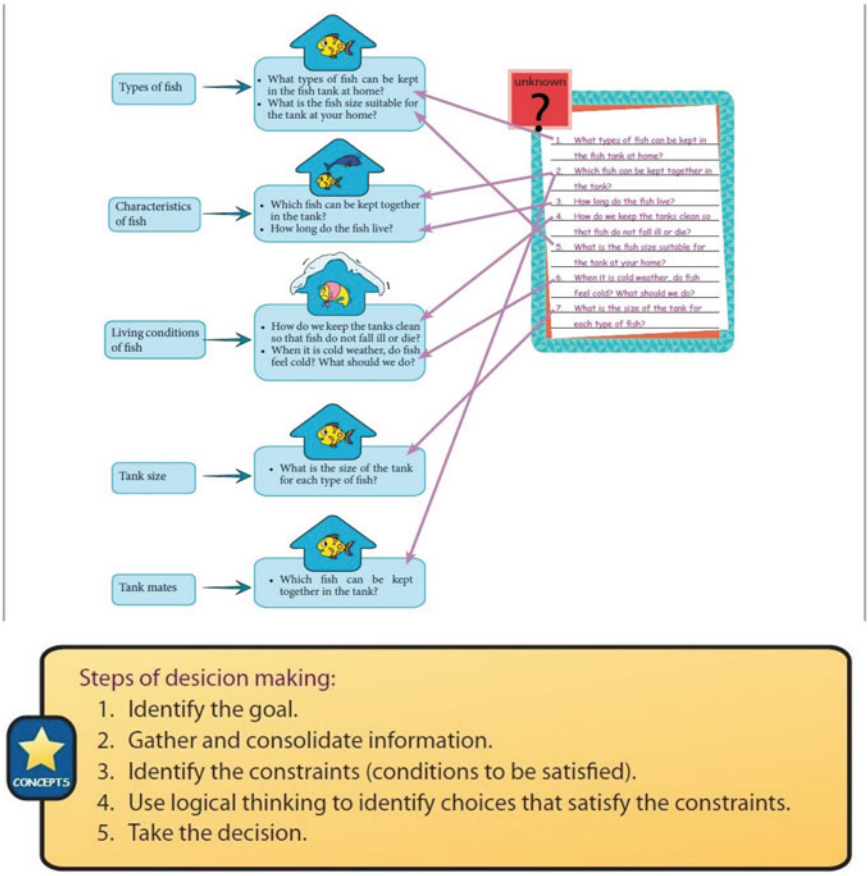


Fig. 20.5 Synthesis of information and decision-making for a problem-solving scenario

20.3.3.3 Multiple Representations of Data

Figure 20.7 shows an example of handling multiple representations of data from grade 7. The students learn to use tables, charts and graphs, and convert data from one representation to another in order to aid problem-solving.

The activities and "Teacher's Corner" emphasize the difference between the concepts in data handling versus the skill of using a spreadsheet application. Students learn to identify suitable representations for a given purpose, when to use which representation, and generate them.

The Computer Masti books develop the computational thinking skills of students by making them repeatedly apply the thinking skills, starting from familiar scenarios, to more and more complex scenarios across multiple grades.

www.dbooks.org

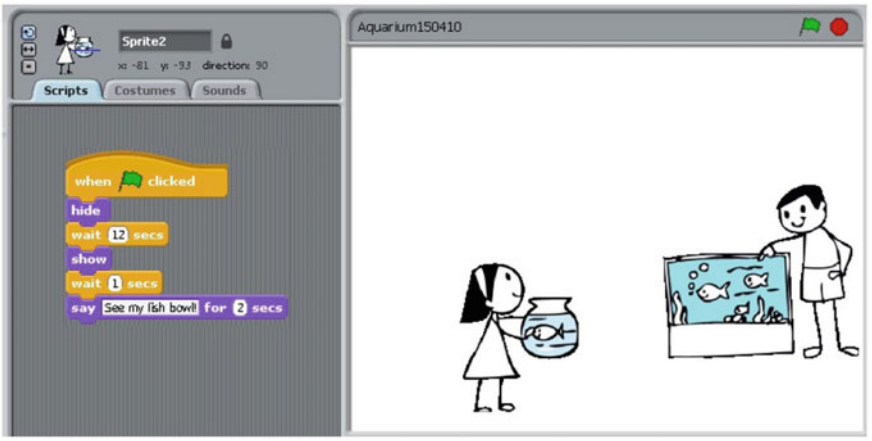
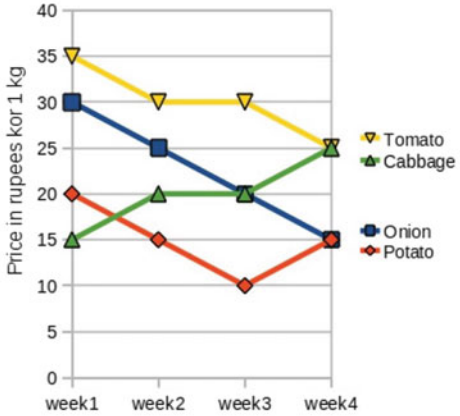


Fig. 20.6 Activity snippet for connecting problem-solving to programming a scenario

The price variation for 1 kg of vegetables in the month of March is given in the graph below. Study the graph and answer the following questions.



a. Using the data given in the graph, fill in the table given below.

Price in the month of March					
Sl. No	Name of the vegetable	week1	week2	week3	week4

g. Riju is buying vegetables in week 3. He has got 80 rupees with him. The shop has all the above vegetables. He has to buy three types of vegetables. The minimum quantity he can buy of each vegetable is 1 kilogram. But he has to follow the conditions below:

If he buys Onion, then he has to buy Potato also.

If he buys Tomato he can not buy Cabbage.

Find out the various options available to him.

Fig. 20.7 Worksheet for handling multiple representations of data

20.4 Implementation

20.4.1 Reach

Over the years, Computer Masti has been used by more than 1.5 million students in India. In 2018, Computer Masti is being implemented in 700+ schools, across India; 26 out of 29 states and 3 of 7 union territories. Table 20.2 below shows year-wise data, in terms of number of schools implementing Computer Masti and the approximate number of students across all grades.

In addition, the Computer Masti books may be freely downloaded (Iyer et al. 2012). The website's Google analytics data for 2009–2018, shows over 100,000 pageviews and 35,000 users, across 150 countries.

20.4.2 Teacher Training

In order to support schools to implement the Computer Masti curriculum, a company called InOpen Technologies was incubated at IIT Bombay in 2009. The company provided books, teacher training, assessment and other support to teachers, for effective implementation. In 2016, the Computer Masti work was transferred to Next Education India Pvt. Ltd. (Next).

The training program includes elements of technology (for example, programming), pedagogy (teaching for active learning, classroom management), as well content (for example, designing ephemera using Word processor). Some examples of the modules relevant to thinking skills

- *CMC philosophy*—The teachers are introduced to the goals of Computer Masti. This enables them to become active partners in achieving these goals for their students.
- *Thinking skills*—Teachers discuss why thinking skills need to be taught explicitly, which thinking skills, and methods of teaching them. Teachers discuss how thinking skills are connected to application usage skills, as well as to topics in other subjects.
- *Assessment*—Teachers create assessment questions for thinking skills at higher cognitive levels, such as apply and evaluate (Anderson & Krathwohl, 2001). These are similar to worksheet questions, such as systematically listing the steps involved in creating a website, evaluating pros and cons of different solutions, and so on.

At the end of training, teachers are provided with detailed lesson plans, activities, assessments and reference materials, to enable them to effectively implement the curriculum.

In order to provide sustained support to teachers, field teams visit the schools periodically. Besides conducting refresher workshops, the team observes classroom

www.dbooks.org

Table 20.2 Data of Computer Masti implementation

	2009–2010	2010–2011	2011–2012	2012–2013	2013–2014	2014–2015	2015–2016	2016–2017	2017–2018
No. of schools	10	45	145	213	292	368	397	543	710
No. of students	4000	20,250	72,500	117,150	175,200	239,200	258,050	352,950	461,500

teaching and provides constructive feedback to teachers. In addition, student workshops on specific topics, such as Scratch programming, are conducted.

20.4.3 MOOCs

In 2017, to scale up the teacher training, two MOOCs (Massive Open Online Courses) specific to teaching-learning of computer studies in schools were created. One MOOC dealt with computer fluency and thinking skills, while the other dealt with programming, as per the CMC curriculum. The MOOCs were offered through the IITBombayX platform (IITBx), in May–June 2017, and again Nov–Dec 2017. The first offering had 553 registered participants, of which 266 completed the course. The second offering had 1644 registered participants, of which 219 completed the course. The plan is to continue offering these MOOCs periodically, until the total number of participants reaches a significant fraction of K-12 teachers in India.

20.5 Evaluation

20.5.1 Student Perceptions

In order to identify student perceptions of learning with Computer Masti, a study was conducted in 2013–2014. Data was gathered through multiple focus group interviews of 24 students in grade 6, in two urban schools. The key findings were:

- Students often mentioned the use of thinking skills in real-life contexts. They gave examples of how step wise thinking helped in drawing pictures, applying logical thinking while solving math problems, and gathering and synthesis of information while working on projects.
- Students reported that they enjoyed Scratch programming and used it as a tool in everyday life. They gave examples of how they create animations and games.
- Students enjoyed solving worksheets that required them to apply thinking skills in different contexts.

In addition, 49 (22 females, 27 males) students of grade 6 answered a survey to determine their perceptions of Computer Masti. 97.9% of them reported that they enjoyed doing the worksheets and activities, 93.8% perceived the learning of thinking skills as useful, and 77.1% perceived that Computer Masti helped them in learning of other subjects.

Although the number of respondents is small, the results indicate that students overall have a positive perception about learning with Computer Masti.

www.dbooks.org

A preliminary study to determine the effectiveness of Computer Masti was carried out 2 two years. It showed an average of 18% improvement in student posttest scores. More rigorous studies are yet to be carried out.

20.5.2 Teacher Perceptions

There is anecdotal evidence that teachers perceive Computer Masti to be useful for teaching- learning of thinking skills, as seen from the quotes below

- "Computer Masti promotes out of the box thinking. Very innovative."
- "CM can really help a student in improving all aspects of studies as well as life."
- "Excellent learning experience for students. Develops analytical abilities."
- "I have seen my son invoke his lessons in the general world. For example, when he is getting ready for the day, he might refer to it as 'changing my costume' (Scratch), or he might narrate as he moves to a room 'move 10 steps and then turn right'. Similarly, I see students implement Computer Masti knowledge in other subjects also."
- "Students come up with lots of questions. As a result, the projects they carry out are better planned and mature."
- "Content such as Computer Masti will surely succeed in enhancing the core skills in students, not just computing but also creative, critical thinking and problem-solving."

Teachers also perceive the training through MOOCs to be useful, as seen by these quotes from the end-of-course surveys:

- "I liked the real-life examples involved at each step"
- "The course is very useful. It gave me confidence to teach computers to my students"
- "It helped me to rethink some strategy that I believed was correct"
- "This course helps me to understand how teachers need to upgrade themselves and try to be creative, rather than teaching in traditional way of teaching, as this generation students need to go along with computer learning through which peer learning, critical thinking can develop and become a responsible citizen too"
- "This course changed my way of teaching programming to my students. I started implementing the approaches learnt in the course. Feeling enriched after the completion of course. Please keep organizing courses like these for the development of teachers."

More systematic studies with teachers are yet to be carried out.

20.6 Conclusion

Many schools in India offer some form of computer studies as a subject to their students. This subject is suitable for learning thinking skills of broad applicability, such as computational thinking and twenty-first-century skills, in addition to learning computer application usage skills. One effort in this direction was the Computer Masti project at IIT Bombay. The project defined the curriculum, developed the textbooks, and provided teacher training support to schools. The curriculum and textbooks have explicit emphasis on computational thinking skills, such as algorithmic thinking and applying this problem-solving process to a wide variety of contexts. A growing number of schools have adopted this curriculum over the past decade, and now the teacher training has been scaled up through MOOCs on teaching-learning of the curriculum. All these resources are released in Creative Commons, so they may benefit schools, teachers, and students across the world.

Acknowledgements The author worked with several teams, at various stages of this project, and acknowledges their contributions. The co-authors of the current edition of the CMC curriculum are: Farida Khan, Sahana Murthy, Vijayalakshmi Chitta, Malathy Baru and Usha Vishwanathan. The co-authors of the first edition of the Computer Masti textbooks are: Vijayalakshmi Chitta, Farida Khan, Usha Vishwanathan and Malathy Baru, with illustrations by Kaumudi Sahasrabudhe, and design by Sameer Sahasrabudhe. The co-instructor for the teacher training MOOCs is Sahana Murthy, supported by TAs: Shitanshu Mishra, Kavya Alse, Gargi Banerjee and Jayakrishnan M. The implementation in schools (2009–2016) was done by InOpen Technologies, co-founded by the author and Rupesh Shah. Since 2016, Computer Masti work is being taken forward by Next Education India Pvt. Ltd.

References

ABET. *ABET accreditation organization*. http://www.abet.org/.

ACM. *Association of computing machinery*. http://www.acm.org/.

Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman.

Bailey, R. (2002). Playing social chess: Children's play and social intelligence. *Early Years, 22,* 163–173.

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,* 48–54.

Barrett, T., Mac Labhrainn, I., & Fallon, H. (2005). *Handbook of enquiry & problem based learning*. Galway: Center for Excellence in Learning and Teaching, NUI.

Big6. *Information and technology skills*. http://big6.com/pages/about/big6-skills-overview.php.

Bransford, J. D., Sherwood, R. D., Hasselbring, T. S., Kinzer, C. K., & Williams, S. M. (1992). Anchored instruction: Why we need it and how technology can help. In D. Nix & R. Spiro (Eds.), *Cognition, education, and multimedia* (pp. 115–141). Hillsdale, NJ: Erlbaum.

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* (pp. 1–25). Vancouver, Canada.

CBSE. *CBSE board*. http://cbseacademic.nic.in/curriculum.html.

www.dbooks.org

CITL. (1999). *Being fluent with information technology*. Washington, DC: Committee on Information Technology Literacy, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council. National Academy Press. http://www.nap.edu/catalog.php?record_id=6482.

CC. *Creative commons*. https://creativecommons.org/.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge: MIT Press.

Ellis, A. K., & Stuen, C. J. (1998). *The interdisciplinary curriculum*. Larchmont, NY: Eye on Education Inc.

IB. *IB board*. http://www.ibo.org/diploma/curriculum/.

ICSE. *ICSE board*. http://www.cisce.org/publications.aspx.

IGCSE. *IGCSE board*. http://www.cambridgeinternational.org/programmes-and-qualifications/cambridge-secondary-2/cambridge-igcse/.

IITBx. *IITBombayX Hybrid MOOCs platform*. https://iitbombayx.in/.

ISTE, & CSTA. (2011). *Operational definition of computational thinking for K-12 education*. International Society for Technology in Education and Computer Science Teachers Association. http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf.

Iyer, S., Baru, M., Chitta, V., Khan, F., & Vishwanathan, U. (2012). *Computer Masti series of books*. InOpen Technologies. http://www.cse.iitb.ac.in/~sri/ssrvm/.

Iyer, S., Khan, F., Murthy, S., Chitta, V., Baru, M., & Vishwanathan, U. (2013). *CMC: A model computer science curriculum for K-12 schools*. Technical Report: TR-CSE-2013-52. Department of Computer Science and Engineering, Indian Institute of Technology Bombay. http://www.cse.iitb.ac.in/~sri/ssrvm/.

Johnson, R. T., & Johnson, D. W. (1998). Action research: Cooperative learning in the science classroom. *Science and Children, 24,* 31–32.

Marzano, R., Brandt, R., Hughes, C., Jones, B., Presselsen, B., Rankin, S., & Suhor, C. (1988). *Dimensions of thinking: A framework for curriculum and instruction*. Association for Supervision and Curriculum Development.

Next. *Next Education India Pvt. Ltd*. https://www.nexteducation.in/.

Olson, S., &Loucks-Horsley, S. (Eds.). (2000). *Inquiry and the national science education standards: A guide for teaching and learning*. Committee on the Development of an Addendum to National Science Education Standards on Scientific Inquiry, National Research Council. http://www.nap.edu/openbook.php?isbn=0309064767.

Padilla, M. J. (1990). *The science process skills.* Research matters—To the science teacher no. 9004. NARST publications. https://www.narst.org/publications/research/skill.cfm.

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

Scratch. *Scratch website*. https://scratch.mit.edu/.

State Boards. *List of boards in India*. https://en.wikipedia.org/wiki/Boards_of_Education_in_India.

Totten, S., Sills, T., Digby, A., & Russ, P. (1991). *Cooperative learning: A guide to research*. New York: Garland.

Wing, J. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–36.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org