Raspberry Pi

Understanding computing education

Volume 1

Raspberry Pi Foundation Research Seminars

Understanding computing education

Volume 1

Raspberry Pi Foundation Research Seminars

Published in April 2021
by the Raspberry Pi Foundation

www.raspberrypi.org

ISSN 2514-586X (16)

Table of contents

Foreword

In May 2020, the Raspberry Pi Foundation held its first online research seminar on computing education. The format was simple: presentation from a researcher, breakout groups, then a whole-group question and answer session. It was a great success! Seventy-six people from nine different countries attended. From that point onwards, we have continued to host Tuesday seminars, first fortnightly and then monthly. By the end of 2021, we will have hosted 22 seminars on a range of different topics in the broad area of computing education.

To accompany the seminars, we plan to publish the proceedings for each set of seminars. This first volume covers nine seminars from May to December 2020. The variety of topics you will find here indicates the wealth of perspectives being brought to this research area.

In the first section, we look at computing education in all its breadth. Professor Celia Hoyles gives her personal perspective on how computing education interfaces with mathematics education, drawing on decades of experience. Rebecca Vivian and her colleagues share their work on the way that computing education is emerging throughout the world. Katharine Childs draws on the literature to summarise factors that have impacted on gender balance in our field.

The second section covers some specific aspects of the teaching of computing in school. Jane Waite describes how we can teach concepts better by understanding semantic wave theory, unpacking and repacking concepts to help learners' understanding. There are two chapters relating to assessment: Shuchi Grover argues for more attention to formative assessment research and practice in computing education. María Zapata Cáceres describes work on developing a test instrument for young children to measure their computational thinking. The last section moves us on to specific computing topics, including programming. David Weintrop describes his doctoral work looking at the relative benefits of block-based and text-based programming. Juan David Rodríguez describes a tool that helps us understand how we can teach new developments in machine learning artificial intelligence to young people, and I am pleased to be able to give an update on some recent work relating to the PRIMM approach, focusing on the role of classroom talk in the programming classroom.

The volume is an eclectic mix of current research and researchers' positions across the field of computing education for young people. One thing is clear from all these chapters: there is still much more research needed to understand the learning of computing more fully. Our ultimate goal is to ensure that the next generation has the skills they need to confidently participate in our increasingly technological society. To do this, we need research to support educators to teach computing effectively and confidently, research around conceptual understanding and progression in the broad area of computer science and digital literacy, and research to understand and address the significant barriers that inhibit diversity in computing education. At the Raspberry Pi Foundation, we are committed both to conducting research in this area and to bringing together researchers from around the world to form a global community.

I am very grateful to all those within the Raspberry Pi Foundation team who made the seminars possible, including Diana Kirby, Jan Ander and Jonnie Howard, as well as many others! We hope you enjoy reading these chapters as much as we enjoyed putting this volume together. Do let us know your feedback, and we look forward to bringing you Volume 2!

Sue Sentance, Chief Learning Officer
Raspberry Pi Foundation,
February 2021

Information about the authors (in alphabetical order)

Katharine Childs
(Raspberry Pi Foundation, UK)

Katharine Childs works at the Raspberry Pi Foundation and coordinates the Gender Balance in Computing project. Her background spans both computer science and learning theory, via her first-class honours degree in IT and Computing and masters degree in computing education. Following 15 years of professional experience working in the IT sector, she went on to teach computing in primary (K-5) schools and deliver professional development activities for other primary teachers. Katharine writes, blogs, and speaks about computer science education research, with a particular focus on gender equity, inclusivity, and physical computing.

Elizabeth Cole
(University of Glasgow, UK)

Elizabeth Cole is a part-time Ph.D. student at the University of Glasgow and an active member of the Centre for Computing Science Education. Elizabeth is currently working on computing science pedagogy in the early years of formal education for children aged 3 to 11. Elizabeth studied computer science at the University of Glasgow as part of her B.Ed in Primary Education and educational computing at Strathclyde University. She worked as a primary teacher followed by various leadership roles in promoted posts before becoming a head teacher. She now works at a national level in her role as HM Inspector of Education evaluating learning and teaching across a range of school sectors and early learning settings.

Shuchi Grover
(Looking Glass Ventures, USA)

Dr. Shuchi Grover is a senior research scientist at Looking Glass Ventures. Her work in computer science (CS) and STEM education since 2000 has spanned both formal and informal settings in the US, Europe, and Asia. Her current research centres on computational thinking (CT), CS education, and STEM+CT integration mainly in formal K-12 settings. Dr. Grover has authored over 100 well-cited scholarly and mainstream articles. She has advised the K-12 CS Framework as well as several K-12 school districts on CS implementation/integration. She served as a member of the ACM Education Advisory Committee and is on the editorial board of ACM Transactions on Computing Education. She has a PhD in Learning Sciences and Technology Design from Stanford University (with a focus on computer science education).

Dame Celia Hoyles
(University College London, UK)

Professor Dame Celia Hoyles was awarded a first-class honours degree in mathematics from the University of Manchester and holds a masters and doctorate in mathematics education. She taught mathematics in London schools before moving into higher education. She became a professor at the Institute of Education, University of London in 1984. Celia has received many awards: first recipient of the International Commission of Mathematics Instruction (ICMI) Hans Freudenthal medal in 2004, and of the Royal Society Kavli Education Medal in 2011. She has received Hon Doctorates from the Open University, Loughborough University, Sheffield Hallam University, and University of Bath. Celia was made an Officer of the Order of the British Empire in 2004 and a Dame Commander in 2014.

Estefanía Martín-Barroso
(Universidad Rey Juan Carlos, Spain)

Dr. Estefanía Martín-Barroso earned her Computer Science B.Sc. in 2002 and M.Eng. in 2004. She obtained her Ph.D. in computer science and telecommunications in 2008 at Universidad Autónoma de Madrid (UAM), with a thesis focused on mobile adaptive learning systems for collaborative contexts. Since 2008, she has been an associate professor with the computer science department at the URJC. Her research interests include learning systems, HCI, and disabilities.

Monica McGill
(CSEdResearch.org, USA)

Monica McGill, Ed.D. is CEO and founder of CSEdResearch.org. Monica earned her B.S. in Computer Science and Mathematics from University of Illinois-Urbana Champaign, M.S. in Computer Science from George Washington University, and Ed.D. in Curriculum and Instruction from Illinois State University. She worked for several years in industry as a computer scientist, and as a professor of computer science and game design/development for over 15 years. Monica has been conducting computing education research for over a decade, with her research work currently focused on supporting K-12 computing education researchers and evaluators via the K-12 Computing Education Research Resource Center. She also currently serves as inaugural chair for the ACM-W North America committee, as a member of the CSTA Board, and as an associate editor of the ACM Transactions on Computing Education.

Keith Quille
(Technological University Dublin, Ireland)

Keith Quille, Ph.D. is a lecturer at the Technological University Dublin, Tallaght Campus, Ireland, teaching Computing with Artificial Intelligence and Machine Learning, Applied Machine Learning and Applied AI and Deep Learning. Keith is a project lead at CSinc.ie where the research group specialises in CS education research (at primary, second, and third-level), K-12 outreach, and K-12 teacher professional development. He was also a second-level teacher for several years. Keith is the Working Group co-chair for the Innovation and Technology in Computer Science Education (ITiCSE) conference and the founding vice-chair for the SIGCSEire (the Irish chapter of the Association of Computing Machinery (ACM) Special Interest Group on Computer Science Education (SIGCSE)).

Juan David Rodríguez
(Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado (INTEF), Spain)

Juan David Rodríguez is a secondary education teacher and software developer. He works at Spain's National Institute of Educational Technologies and Teacher Training (INTEF), a unit of the Spanish Ministry of Education and Vocational Training which has responsibility for the integration of ICT and teacher training in non-university educational stages. Juan is currently working on computational thinking skills development through practical artificial intelligence activities. He has started exploring how machine learning (ML), one of the most used techniques in current AI applications, can be taught at school. To do this, he is developing the educational tool LearningML which is designed to easily build ML models that can be used in Scratch programs.

Marcos Román-González
(Universidad Nacional de Educación a Distancia, Spain)

Dr. Marcos Román-González was born in Madrid, Spain in 1977. He earned his B.S. in Educational Psychology in 2006 and M.S. in Educational Research in 2013. He obtained his Ph.D. in Education in 2016 at Universidad Nacional de Educación a Distancia (UNED), where he now works as an Associate Professor. His thesis was focused on code-literacy and computational thinking development in primary and secondary school. He is the first author of the Computational Thinking Test (CTt), which has been translated and is being used worldwide for assessing computational thinking in primary and secondary.

Sue Sentance
(Raspberry Pi Foundation, UK)

Dr. Sue Sentance is the Chief Learning Officer at the Raspberry Pi Foundation and Visiting Fellow at King's College London, UK. She researches the teaching of programming in school, teacher professional development, and physical computing. Her academic background is in computer science, artificial intelligence, and education, and she is a qualified teacher and teacher educator. At the Raspberry Pi Foundation, Sue's role is to advise on teaching and learning, and lead on research on computing education for young people. She plays a leading role in the government-funded National Centre for Computing Education. In 2020, Sue was awarded a Suffrage Science award for Maths and Computing and in 2017 the BERA Public Engagement and Impact Award for her services to computing education.

Rebecca Vivian
(University of Adelaide, Australia)

Dr. Rebecca Vivian is a qualified teacher and Senior Research Fellow in the Computer Science Education Research Group (CSER) at the University of Adelaide. She is lead designer for CSER's national K-12 Digital Technologies Education teacher training program and has extended this work to lead an Australian teacher training program in mathematics. Rebecca is a learning scientist and interdisciplinary researcher who undertakes research in STEM engagement, K-12 CS education, and teacher professional learning, with experience working with industry, government, and education stakeholders to influence policy and produce tangible benefits for the Australian education community.

Jane Waite
(Queen Mary University of London, UK)

Jane Waite works and studies at Queen Mary University of London. She is undertaking a part-time Ph.D. studying the teaching of design in K-5 (primary) programming activities. Jane also organises and runs teacher professional development and undergraduate modules on computer science education. Working with Sue Sentance she has researched PRIMM, the micro:bit, and pedagogy in general. With Paul Curzon she is investigating the use of semantic waves in the teaching of computer science. Jane is the Computing At School Research and University Working Group Chair.

David Weintrop
(University of Maryland, USA)

Dr. David Weintrop is an assistant professor in the Department of Teaching & Learning, Policy & Leadership in the College of Education with a joint appointment in the College of Information Studies at the University of Maryland. His research focuses on the design, implementation, and evaluation of accessible, engaging, and equitable computational learning experiences. He is also interested in the use of technological tools in supporting exploration and expression across diverse contexts including STEM classrooms and informal spaces.
His work lies at the intersection of design, computational thinking education, and the learning sciences. David has a Ph.D. in the Learning Sciences from Northwestern University and a B.S. in Computer Science from the University of Michigan.

María Zapata Cáceres
(Universidad Rey Juan Carlos, Spain)

María Zapata Cáceres qualified as an architect at the Universidad Politécnica de Madrid and graduated in Computer Science Engineering at the Universidad Nacional de Educación a Distancia in Spain. She also holds masters degrees in Virtual Environments (CSA) and Video Games Design and Production (UEM). She is currently pursuing her doctorate in Information Technology and Communications at the Universidad Rey Juan Carlos in Madrid, where she is a researcher and visiting professor. Her main area of research includes video games as learning instruments for computer science both in individual and collaborative environments. She has more than 15 years of professional experience as an entrepreneur and independent professional with activities related to 3D design, video games, technology, and teaching.

Section 1: Computing in context

Section 1: Computing in context

Programming and mathematics: insights from research in England

Dame Celia Hoyles (University College London, UK)

I start this short piece by providing a glimpse of the way computing was introduced in schools in England, culminating in 2014 with the introduction of a new statutory primary national computing curriculum for students aged 6 to 16 years. One key landmark on the way was the influential report from the Royal Society, *Shut Down or Restart* (The Royal Society, 2012). Then came the second Royal Society report, *After the Reboot: Computing Education in Schools* (The Royal Society, 2017), which recognised the importance of the teacher for the success of the curriculum initiative and led to the setting up of the National Centre for Computing Education, NCCE, specifically to support the teaching of computing.[1]

The computing curriculum included, as a key aspect, that students should design, build, and debug programs. My main personal concern was how programming, as well as being part of computing, might also fit with the rest of the curriculum, with particular reference, given my background, for mathematics. Could this curriculum innovation of computing be harnessed for the benefit of all subjects?

To address this question, I want to provide an outline of the history of programming and mathematics, which for me had its roots in innovations from MIT in the 1980s, and the vision of Seymour Papert for the development of Logo, as a language for learning. This is when I became personally convinced of the potential for programming in mathematics teaching and learning; as a teacher I experienced the 'buzz' of a classroom where the learners were actively engaged in exploring and discussing mathematical ideas through programming them. At the same time, Papert proposed his theory of constructionism, that proposed that learning tended to be effective when making an artefact that was personally or socially meaningful, could be shared with others, reflected upon, and debugged (see for example (Kafai & Resnick, 1996)).

In this early work, the notions of powerful ideas and design were stressed; that is a *well-designed* constructionist activity should have *personal* meaning and *emotional* connection with learners and empower them in some way, put simply so they could do something that before they were unable to do. Such ideas had deep resonance for me as so much of mathematics is not experienced as personally meaningful, just seen as simply a 'dance of symbols' without underlying structure and linked to little emotion except anxiety, feeling stupid and 'not getting it'.

During this time, a group of us set up the LogoMathematics group which met regularly with participation from across the world, leading directly and indirectly to publications over the subsequent 50 or more years (see for example, (Papert, 1972; Hoyles & Noss, 1992; Noss & Hoyles, 1996; Monaghan, Trouche, & Borwein, 2016)).

After setting the scene I will describe the ScratchMaths (SM) project, (now called *UCL ScratchMaths*), the latest research project

[1] http://teachcomputing.org

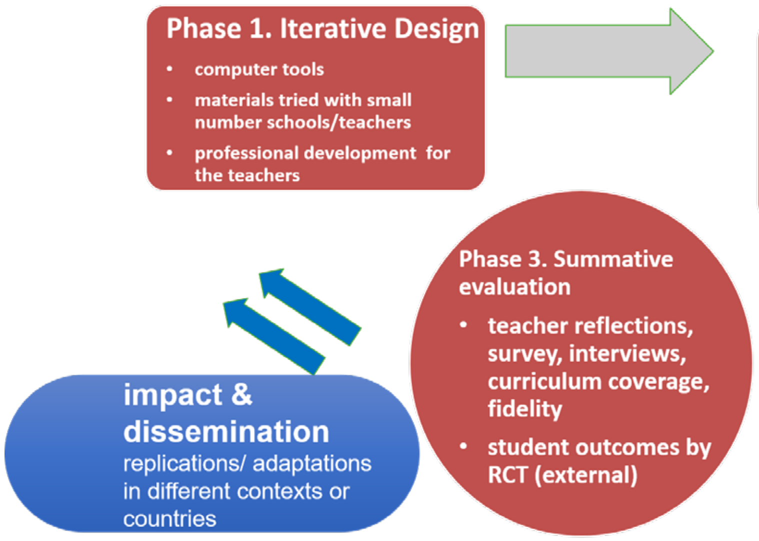

Phase 1. Iterative Design
computer tools
materials tried with small number schools/teachers
professional development for the teachers
Phase 2. Implementation at scale
> 100 schools across country
PD in regional 'hubs'
formative evaluation
Phase 3. Summative evaluation
teacher reflections, survey, interviews, curriculum coverage, fidelity
student outcomes by RCT (external)
impact & dissemination
replications/ adaptations in different contexts or countries

Figure 1. Phases of the ScratchMaths research

in which I have worked that looked at the programming/mathematics interface. The UCL ScratchMaths designed and implemented a longitudinal two-year intervention at the intersection of mathematics and computing, targeted for 8- to 11-year-old students in English schools and involving programming in Scratch (Noss, Hoyles, et al., 2020).

The phases of the ScratchMaths research are shown in Figure 1. Much of the effort of the ScratchMaths team took place in the first phase, the Iterative Design Phase, where the we worked with a small group of schools to iteratively design computer tools and student/teacher materials and pilot them in the schools, along with a programme of professional development for the teachers.

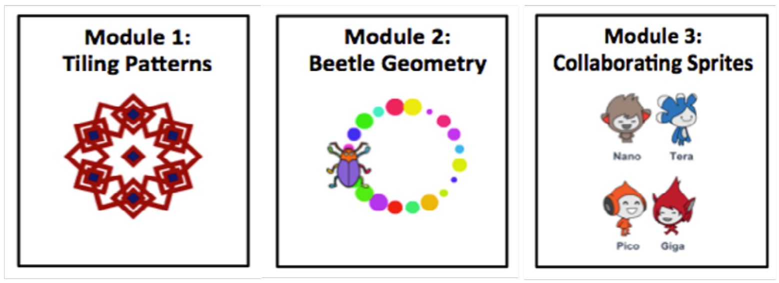
Our team was interdisciplinary[2] with expertise in mathematics education, computing, and design, and we worked closely with teachers to iteratively develop our original designs. We aimed to foster *mathematical thinking*. *This can be defined as an awareness and appreciation of mathematical structure, the articulation of coherent explanations for outcomes and the reasoning behind them, and being comfortable and fluent with the formal expression of relationships*.

To pursue this aim, we developed student and teacher curriculum support materials organised into six modules, three to be taught per year, involving about 20 hours teaching. The modules can be considered as *microworlds*, designed to provoke engagement with key ideas in mathematics and in computing. (For background on microworld development, see (Hoyles, 1993), and the recent contributions of Chronis Kynigos to the Mathematics Knowledge Network lecture series.[3])

[2] Professor Dame Celia Hoyles (Mathematics) and Professor Richard Noss (Mathematics) - UCL Knowledge Lab, Professor Ivan Kalas, (Computing) - Comenius University, Bratislava, Slovakia Dr Laura Benton (Computing) and Piers Saunders (Mathematics) - UCL Knowledge Lab, Prof Dave Pratt (Mathematics) - UCL Institute of Education

[3] http://mkn-rcm.ca/online-seminar-series-on-programming-in-mathematics-education

Year 5 (9-10 yrs) – Computing focs (20+ hours)



Year 6 (10-11 yrs) – Mathematics focs (20+ hours)

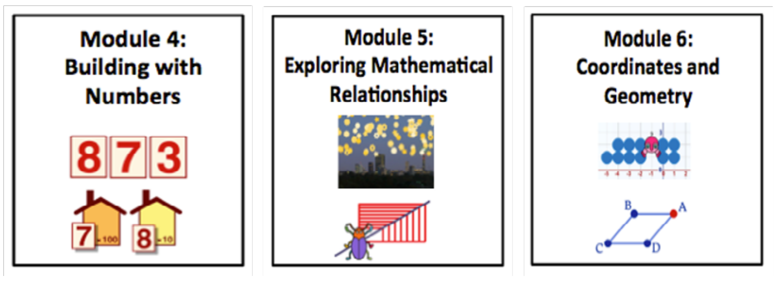


Figure 2. Overview of UCL ScratchMaths

Figure 2 shows a summary of the six UCL ScratchMaths microworlds produced.

All the materials are freely available, now updated to Scratch 3.0, through the UCL website.[4]

The ScratchMaths team took as the following components of computational thinking[5] derived from a large number of rather similar definitions and resources available at that time. (For background, see (Benton et al., 2017)) :

- *Abstraction*: seeing a problem and its solution at many levels of detail
- *Algorithms*: thinking about tasks as a series of logical steps
- *Decomposition*: understanding that solving a large problem can involve breaking it down into smaller problems
- *Pattern recognition*: appreciating that a new problem is likely to be related to other problems already solved
- *Generalisation*: realising that a solution to a problem can be made in ways that can solve a range of related problems

In Phase 1, it also became clear that we needed an explicit pedagogic framework for ScratchMaths to facilitate our future work, and we devised one with the teachers in the four design schools referred to as the “five Es”, with each E derived from a wealth of prior research in mathematics education about effective pedagogy:

[4] http://www.ucl.ac.uk/scratchmaths

[5] For an up-to-date- summary of definitions and research on Computational Thinking, see Paul Dryvers lecture as par of the Mathematics Knowledge Network lecture series, available at http://mkn-rcm.ca/online-seminar-series-on-programming-in-mathematics-education/.

Explore: investigate, try things out yourself, debug in reaction to feedback.
Envisage: have a goal in mind, predict the outcome of the program before trying on the computer.
Explain: explain what you have done, articulate reasons behind your approach to yourself and to others.
Exchange: collaborate and share, try to see a problem from another's perspective as well as defend your own approach and compare with others.
bridgE: make explicit links to the mathematics curriculum.

In relation to the last E, *bridgE*, we had learned from our earlier research in programming and mathematics that often we had assumed connections between these two fields would be made, only to find that this was not the case. All was too implicit and assumed. The five Es framed the professional development programme we devised (two days per year), a programme that was a critical part of the SM intervention and the planned classroom implementation.

Alongside the design phase the SM team planned for *Phase 2. Implementation at scale*. We signed up to the project over 100 schools across England grouped around seven regional hubs that would form the focus for professional support and formative evaluation. At this point, an external independent evaluator was appointed by the funders who undertook to assess the project in terms of its effect on scores of computational thinking and of mathematics. The results of the ScratchMaths intervention are reported in full in the evaluation report.[6] We note that ScratchMaths had a positive and significant impact on student computational thinking (CT), as reported by the evaluator using a randomised control trial methodology with 111 schools across England and measured by a test of computational thinking designed and administered by them at the end of the first year of the intervention. We also note the important results that this positive effect was particularly evident among educationally disadvantaged students. There was no evidence of any interaction between the impact of SM on CT test scores and gender: thus girls and boys appeared to engage with SM to a similar extent, an outcome that is particularly important in view of the finding persistent in the literature that girls tend not to be so engaged in computing as boys.

However, there was no impact of SM on mathematics attainment as measured by the evaluators on the basis of the student results in the statutory national mathematics test, Key Stage 2 test taken by all 11-year-old students in England. As a way to seek to explain these findings, I called on the notion of *fidelity of implementation* (see (O'Donnell, 2008) for a review of *Defining, Conceptualizing, and Measuring Fidelity of Implementation*), and how in our study fidelity appeared to have been negatively influenced by the high-stakes testing in mathematics in England leaving little room for innovation in classrooms for 11-year-olds. These tests involved a formal paper and pencil mathematics test and are used to rank schools and teachers so much time is spent reviewing and revising, so teachers found little resource to devote to ScratchMaths.

Finally, I want to dwell a little on the final and still ongoing phase of the ScratchMaths project, concerning how it has been disseminated and its impact on other projects in England and internationally. The materials have been used in a great many countries across Europe, and beyond. I note in particular that the ScratchMaths project has been followed up in New South Wales in Australia (see Holmes, Prieto-Rodriguez, et al.,

[6] https://educationendowmentfoundation.org.uk/projects-and-evaluation/projects/scratch-maths/.

2018) where it continues to be widely adopted. Also of note is the nationwide project that took place in Spain (INTEF, 2020), part of which concerned a replication of the ScratchMaths project along with assessing its impact. I translated one finding from this report that was of particular relevance: namely that *"the results show that it is possible to include programming activities in 5th grade in the area of mathematics, so that students not only learn to program and engage in computational thinking, but also improve the development of their mathematical competence greater than their colleagues who have worked in this same area using other types of activities and resources not related to programming."*

I would like to end by reflecting on how the ScratchMaths research might be improved or updated in future work, not least as teachers are becoming more confident and competent in their understanding of computational concepts, in teaching them and in using them to explore mathematical ideas through programming. So I present some personal thoughts about our project and its limitations with the wisdom of hindsight, and pose some research challenges that might be interesting for others in the community to address. For example, the need to:

1. Develop more *nuanced, rigorous, and targeted assessment instruments of student and teacher content knowledge in mathematics, mathematical thinking, and in computing* to be administered as post-tests following engagement in each of the ScratchMaths microworlds and as delayed post-tests several months later, rather than use the standard national tests as adopted in the evaluation of UCL ScratchMaths.
2. Research in more detail the actual practices in classrooms to include documentation of teacher and student interactions and output, in order to provide detail of classroom implementation and how far the pedagogic framework was enacted. In particular, such research might provide some explanation of the outcomes reported for UCLScratchMaths in relation to socially disadvantaged students and girls as mentioned above, taking as a starting point the idea of fidelity while recognising the 'chaotic' nature of real classrooms, teacher practices, and policy demands.
3. Develop a more detailed description of the *nature and content of the professional development* that is undoubtedly needed prior to successful implementation of the ScratchMaths intervention.

At the time UCL ScratchMaths was conceived and operationalised, computing was very new in England. Teaching and learning has been transformed in the intervening years, not least as much education has moved online as a result of the coronavirus pandemic, and the magnificent efforts of the NCCE to support the teaching of computing across the country. Teachers and students have undoubtedly become more fluent in working online in general and in programming in particular. One might expect that the integrity of the SM materials would remain constant, given the principles of design on which they were based while its implementation would be less challenging in these changed circumstances. But this is a matter for further research.

References

Benton, L. Hoyles, C., Kalas, I & Noss, R. (2017). Bridging primary programming and mathematics: preliminary findings of design research in England. In *Digital Experiences in Mathematics Education*, pp 1- 24

Holmes, K. Prieto-Rodriguez, E., Hickmott, D. & Berger, N. (2018). Using coding to teach mathematics: Results of a pilot project. *Proceedings of the 5th International STEM in Education Conference*. Brisbane.

Hoyles, C. (1993). Microworlds/Schoolworlds: The transformation of an innovation. In Keitel, C., Ruthven, K. (eds) *Learning from Computers: Mathematics Education and Technology*. NATO ASI, Series F: Computer and Systems Sciences, 121, 1-17.

Hoyles C. and Noss, R. (1992). (eds) *Learning Mathematics and Logo*. Cambridge MA: MIT Press.

Kafai, Y., & Resnick, M. (1996). Constructionism in Practice: Designing. Thinking, and Learning in a Digital World. New York: Taylor & Francis Group.

INTEF (2020). La Escuela de Pensamiento Computacional (School for Computational Thinking (The School for Computational Thinking). Instituto Nacional de Tecnologías Educativas y De Formación Del Profesorado (National Institute of Educational Technologies and Teacher Training) Final Report . Available at: https://intef.es/tecnologia-educativa/pensamiento-computacional/

Monaghan, J., Trouche, L., Borwein, J. M. (2016). Tools and Mathematics. Springer

Noss, R. and Hoyles, C. (1996) *Windows on Mathematical Meanings: Learning Cultures and Computers*. Dordrecht: Kluwer Academic Publishers.

Noss, R., Hoyles, C., Saunders, P., Clark-Wilson, A., Benton, L., and Kalas, I., (2020). Making constructionism work at scale: the story of ScratchMaths: In Holbert, N., Berland, M., Kafai, Y. *Designing Constructionist Futures: The Art, Theory, and Practice of Learning Designs*, Cambridge, MA: MIT Press.

O-Donnell, C. (2008). Defining, conceptualizing, and measuring fidelity of implementation and its relationship to outcomes in K–12 curriculum intervention research. *Review of Educational Research*, 78, 1, 33-84

Papert, S. (1972). Teaching children to be mathematicians vs. teaching about mathematics. *International Journal of Mathematics Education in Science and Technology*, 3, 249-262.

The Royal Society (2012). *Shut down or restart? The way forward for computing in UK schools*. Available at: https://royalsociety.org/topics-policy/projects/computing-in-schools/report/

The Royal Society (2017). *After the reboot: computing education in UK schools*. Available at: https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf

Section 1: Computing in context

Measuring the enacted K-12 computing curriculum

Elizabeth Cole (University of Glasgow, UK), Monica McGill (CSEdResearch.org, USA), Keith Quille (Technological University Dublin, Ireland), and Rebecca Vivian (University of Adelaide, Australia)

Abstract

The *intended curriculum* — the curriculum that is intended to be taught through policy, curriculum documents, or other required mandates — and the *enacted curriculum* — the curriculum that is actually taught in classrooms by teachers — are ideally aligned. However, often there is a chasm between the two. With computing education being relatively new to schools and teachers across many countries, we wanted to learn if a chasm existed and, if it did, how wide it is across different countries. Working as part of an international team, we created a set of templates for measuring intended curricula and a survey instrument, MEasuring TeacheR Enacted Curriculum (METRECC), to measure enacted curricula. The original pilot investigated the enacted curriculum in seven countries (with 244 teacher participants). Our research found that both visual and text-based programming languages are being used across K-12, warranting further research into potential impact on student learning and motivations. Unplugged activities are commonly used across K-12, extending into later years despite not being explicitly defined in intended curricula. Further, teachers' motivations for programming language choice are consistent across countries and our study revealed that student-driven factors motivate selection. This initial study was followed by additional analysis with respect to teacher self-esteem that was found to differ across multiple factors such as experience in teaching CS in years and gender. We punctuate our work with the adaptation of the instrument for use in South Asia and a call to the community to consider middle- and low-income nations in future research.

Introduction

While there has been some efforts to collate intended curricula, internationally, nationally, and regionally (Porter and Smithson, 2001; Hubwieser, Armoni, and Giannakos, 2015; Gander et al., 2013; Balanskat and Engelhardt, 2014; The Royal Society, 2012; The Royal Society, 2017; Hong et al., 2016; Moller and Crick, 2016; Sysło and Kwiatkowska, 2015), there is a need to understand the enacted curriculum and how well it aligns with the curriculum as intended to be taught. In June 2019, a working group led by Sue Sentance, Katrina Falkner, and Rebecca Vivian at the Association of Computing Machinery's (ACM) 2019 conference on Innovation and Technology in Computer Science Education (ITiCSE), conducted an international study of K-12 computer science (CS) implementation across Australia, England, Ireland, Italy, Malta, Scotland, and the United States. The purpose of the study was to develop instrumentation that would provide descriptive data about the intended and enacted computing education curriculum in K-12 schools. The genesis of the work arose from the analysis of previous attempts to assess and identify K-12 CS implementation efforts internationally.

This chapter describes three major efforts related to this research:

- Results originating from a detailed collaborative international process which resulted in the design, pilot, and evaluation of an international survey instrument for Measuring Teacher Enacted Computing Curriculum (METRECC) (Falkner, 2019a; Falkner, 2019b)
- Results of a deeper analysis of the self-esteem scale that was included as part of the METRECC survey (Vivian et al., 2020)
- The adaptation of METRECC for usage in middle- and lower-income nations, with a focus on South Asian classrooms (Anwar et al., 2020)

While some quantitative and qualitative reports existed with respect to measuring the intended and enacted curriculum, none succinctly provided a tool that could be applied at scale and easily provide the descriptive data for international comparison. During the initial planning phase of our study, it became apparent that we needed a formal method for framing the intended curriculum as well as instrumentation to capture the curriculum as it is taught in the K-12 classrooms. Both instruments also needed to be framed in a way that took into account the various differences in primary and secondary structures across seven countries.

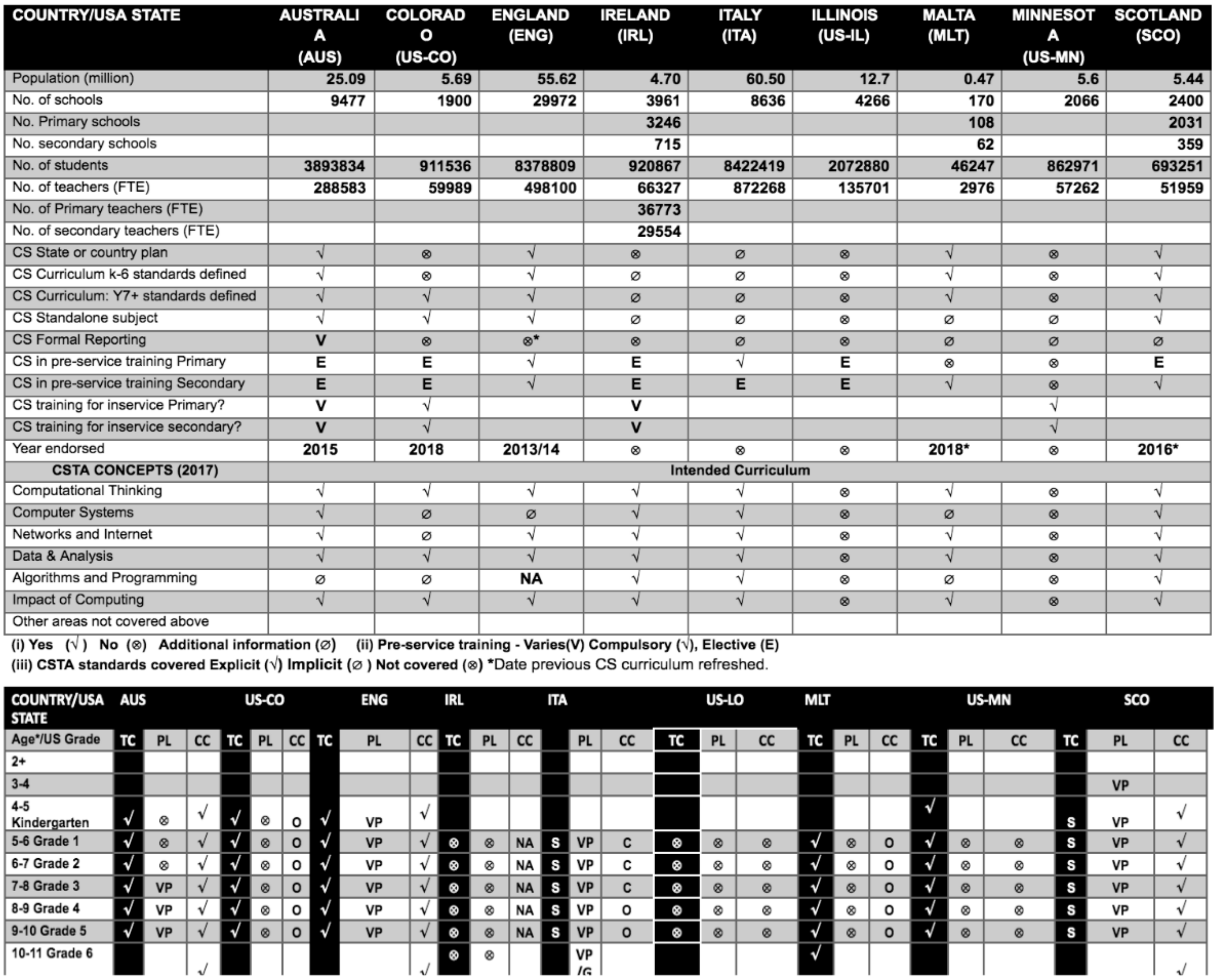
In addition to providing evidence of the validity of the survey instrument, the pilot phase reviews of the curriculum landscapes from Europe, the UK, and the US provide some information on international CS K-12 efforts. This work also laid the foundation for several follow up studies, one of which looks specifically at teacher computer science self-esteem (Vivian, 2020) across the seven jurisdictions using the pilot data. Teacher self-esteem was found to differ across multiple factors such as experience in teaching CS in years and gender. This work might be useful to CS educators who are designing CPD or pre-service teacher programmes.

Many of the countries examined in the original study consisted of high-income nations, while little is known about primary and secondary computing education efforts in both middle- and low-income nations. Another follow-up study, therefore, reinterpreted METRECC for use in middle- and low-income countries in South Asia (Anwar et al., 2020). This pilot of the METRECC South Asia instrument is a step towards the validation of the instrument across nations with varying socio-economic demographics. This work sets a solid foundation for the continued longitudinal implementation of the METRECC instrument to further investigate international enacted curricula.

Measuring the intended and enacted curriculum

An initial body of work focused on the alignment between the intended and enacted curriculum in the areas of topics taught and programming languages used (Falkner, 2019b). These critical areas of CS curricula require further analysis and monitoring not only in terms of alignment and its ensuing benefits, but also in relation to our assumptions as tertiary educators on prerequisite knowledge and experience.

Two instruments were developed for the study by the 2019 ITICSE working group: a country report template to capture the *intended curriculum* and a teacher survey instrument to capture the *enacted curriculum*. The following explanation briefly describes the instruments with more detail available in the published report (Falkner et al., 2019a).



COUNTRY/USA STATE	AUSTRALIA (AUS)	COLORADO (US-CO)	ENGLAND (ENG)	IRELAND (IRL)	ITALY (ITA)	ILLINOIS (US-IL)	MALTA (MLT)	MINNESOTA (US-MN)	SCOTLAND (SCO)
Population (million)	25.09	5.69	55.62	4.70	60.50	12.7	0.47	5.6	5.44
No. of schools	9477	1900	29972	3961	8636	4266	170	2066	2400
No. Primary schools				3246			108		2031
No. secondary schools				715			62		359
No. of students	3893834	911536	8378809	920867	8422419	2072880	46247	862971	693251
No. of teachers (FTE)	288583	59989	498100	66327	872268	135701	2976	57262	51959
No. of Primary teachers (FTE)				36773					
No. of secondary teachers (FTE)				29554					
CS State or country plan	√	⊗	√	⊗	∅	⊗	√	⊗	√
CS Curriculum k-6 standards defined	√	⊗	√	∅	∅	⊗	√	⊗	√
CS Curriculum: Y7+ standards defined	√	√	√	∅	∅	⊗	√	⊗	√
CS Standalone subject	√	√	√	∅	∅	⊗	∅	∅	√
CS Formal Reporting	V	⊗	⊗*	⊗	∅	⊗	∅	∅	∅
CS in pre-service training Primary	E	E	√	E	√	E	⊗	⊗	E
CS in pre-service training Secondary	E	E	√	E	E	E	√	⊗	√
CS training for inservice Primary?	V	√		V				√	
CS training for inservice secondary?	V	√		V				√	
Year endorsed	2015	2018	2013/14	⊗	⊗	⊗	2018*	⊗	2016*
CSTA CONCEPTS (2017)	Intended Curriculum								
Computational Thinking	√	√	√	√	√	⊗	√	⊗	√
Computer Systems	√	∅	∅	√	√	⊗	∅	⊗	√
Networks and Internet	√	∅	√	√	√	⊗	√	⊗	√
Data & Analysis	√	√	√	√	√	⊗	√	⊗	√
Algorithms and Programming	∅	∅	NA	√	√	⊗	∅	⊗	√
Impact of Computing	√	√	√	√	√	⊗	√	⊗	√
Other areas not covered above									

(i) Yes (√) No (⊗) Additional information (∅) (ii) Pre-service training - Varies(V) Compulsory (√), Elective (E)
(iii) CSTA standards covered Explicit (√) Implicit (∅) Not covered (⊗) *Date previous CS curriculum refreshed.

COUNTRY/USA STATE	AUS			US-CO			ENG			IRL			ITA			US-LO			MLT			US-MN			SCO		
Age*/US Grade	TC	PL	CC	TC	PL	CC	TC	PL	CC	TC	PL	CC		PL	CC	TC	PL	CC	TC	PL	CC	TC	PL	CC	TC	PL	CC
2+																											
3-4																										VP	
4-5 Kindergarten	√	⊗	√	√	⊗	O	√	VP	√													√			S	VP	√
5-6 Grade 1	√	⊗	√	√	⊗	O	√	VP	√	⊗	⊗	NA	S	VP	C	⊗	⊗	⊗	√	⊗	O	√	⊗	⊗	S	VP	√
6-7 Grade 2	√	⊗	√	√	⊗	O	√	VP	√	⊗	⊗	NA	S	VP	C	⊗	⊗	⊗	√	⊗	O	√	⊗	⊗	S	VP	√
7-8 Grade 3	√	VP	√	√	⊗	O	√	VP	√	⊗	⊗	NA	S	VP	C	⊗	⊗	⊗	√	⊗	O	√	⊗	⊗	S	VP	√
8-9 Grade 4	√	VP	√	√	⊗	O	√	VP	√	⊗	⊗	NA	S	VP	O	⊗	⊗	⊗	√	⊗	O	√	⊗	⊗	S	VP	√
9-10 Grade 5	√	VP	√	√	⊗	O	√	VP	√	⊗	⊗	NA	S	VP	O	⊗	⊗	⊗	√	⊗	O	√	⊗	⊗	S	VP	√
10-11 Grade 6			√						√	⊗	⊗			VP /G					√								√

Figure 1. Country report templates for the METRECC

Capturing the intended curriculum

The country report template is designed to be completed by the survey administrator and captures the country demographics relating to schools and the intended curriculum. For analysis and comparison, the broad curriculum and information for each country is organised as follows:

1. Demographics (e.g. such as total population, number of schools, number of teachers)
2. CS curriculum state or country plan standards and requirements
3. Year level (with age for comparisons) mapped to prescribed curriculum and programming requirements
4. General CS topics covered

Although the aim is to analyse intended and enacted curricula for individual states/countries, the templates show similarities and differences across states/countries. However, a key consideration for high-level country by country intended curricula comparison is the CS content variations. To overcome the issue of variance in the short term, the country report template includes a list of broad CS topics based on available literature. Future work will refine this list based on the pilot study results.

Capturing each country's enacted curriculum
The Measuring Teacher Enacted Computing Curriculum (METRECC) instrument captures the enacted curriculum. The survey measures what teachers are doing in the classroom, taking account of their context. A set of key categories of interest internationally in terms of the enacted CS education curriculum were curated and, where possible, refined from existing surveys with evidence of reliability and validity. The developed survey includes 11 sections and a total of 53 questions, including:

1. Introduction
2. Demographics
3. Current work (position)
4. Qualifications
5. Student composition
6. Support and resourcing
7. Assessment of student learning
8. Classroom practice and motivation
9. Self-efficacy/self-esteem
10. Professional development
11. Consent for publishing data

The intention was that each section can be administered independent of each other, with survey administrators being able to piece together a survey to suit their needs.

Findings

The final dataset includes 244 responses across 7 countries: USA (n=115), England (n=52), Italy (n=20), Ireland (n=19), Scotland (n=18), and Malta (n=6). The highlights of the demographic data showed that the respondents had these characteristics:

- 61% female; 37% male
- 87% ages 30 to 59
- 49.6% teaching for 12 or more years
- 89% from government/public schools
- 36% from disadvantaged schools
- 29% rural/remote areas; majority urban/metro
- All were teaching computing in school in some capacity across age 3 to 19
- The majority of respondents are teaching across middle school and secondary (age 13 to 17)

The pilot study sample below investigates the questions relating to curriculum topics and programming languages enacted in the classroom for comparison against intended curriculum requirements.

Country reports (intended) curriculum broad topics covered

Firstly, we examined topics that featured in the intended curriculum across the several countries as a comparison point for what teachers were working with when it comes to their enacted curriculum. Table 1 presents the results for the intended curriculum — broad topics that are explicitly or implicitly defined in country or state curriculum documents as identified in the country report snapshot. This information was used to not only compare countries or states but to also map similarities and gaps in enacted curriculum.

Concepts	AUS	US-CO	ENG	IRL	ITA	US-IL	MLT	SCO
Computational Thinking	✓	✓	✓	✓	✓	✕	✓	✓
Computer Systems	✓	❖	✓	✓	✓	✕	❖	✓
Networks and Internet	✓	❖	✓	✓	✓	✕	✓	✓
Data & Analysis	✓	✓	✓	✓	✓	✕	✓	✓
Algorithms and Programming	❖	❖	✓	✓	✓	✕	❖	✓
Impact of Computing	✓	✓	✓	✓	✓	✕	✓	✓

Table 1. Country report information of topics featured explicitly (✓), implicitly (❖), or not at all (X) in intended curriculum across countries

CS Topics	Australia	England	Ireland	Italy	Malta	Scotland	USA
Algorithms	**79%***	**100%***	**68%***	**70%***	**33%***	**100%***	**82%***
Artificial Intelligence	7%	44%	32%	10%	0%	6%	**30%***
Computational Thinking	**57%***	96%	68%	**45%***	17%	**89%***	72%
Cybersecurity	71%	83%	16%	35%	17%	**72%***	**57%***
Data analysis and visualisation	**29%***	44%	26%	25%	0%	11%	**43%***
Data representation (e.g. digital data, binary)	**57%***	**88%***	**53%***	**45%***	**33%***	100%	**68%***
Databases	14%	71%	42%	**45%***	**17%***	89%	*27%
Design process (or Design Thinking)	**86%***	**54%***	**58%***	**20%***	17%	56%	72%
Ethics	**29%***	**88%***	58%	35%	0%	**56%***	**75%**
Hardware	**26%***	90%	**68%***	**55%***	**50%***	**94%***	**61%***
Information Systems	**50%***	58%	21%	**30%***	33%	**72%***	35%
Machine Learning	7%	23%	26%	5%	17%	11%	21%
Networks and Digital Systems	**64%***	90%	16%	40%	**17%***	**39%***	**45%***
Privacy	**64%***	77%	42%	40%	**17%***	**61%***	**64%***
Programming skills and concepts	**79%***	**100%***	**100%***	**80%***	**50%***	**100%***	**87%***
Robotics	79%	33%	42%	40%	**50%***	11%	47%
Web Systems	36%	62%	**37%***	**50%***	17%	**94%***	38%
Total sample (n)	14	52	19	19	6	18	115

Table 2. Percentage of teachers teaching CS topics across countries (asterisk denotes the topic is part of the country's intended curriculum)

In the example above we can easily see differences, such as where formal curricula are not available for CS topics and that in a number of countries topics taught are implicitly defined in the curriculum rather than explicitly.

Survey responses (enacted) curriculum broad topics covered

In Table 2 data from the country reports is mapped against the percentage (%) of teachers who indicated "yes, I teach this [topic]" in the survey. An asterisk indicates that the topic is part of the intended curriculum, providing a comparison point in terms of what topics are expected to be covered and what teachers are explicitly teaching.

Topics mostly being taught across countries are algorithms, programming, computational thinking, and data representation. Topics taught less frequently are machine learning and artificial intelligence. We can also see gaps, particularly in Malta and some topics within countries.

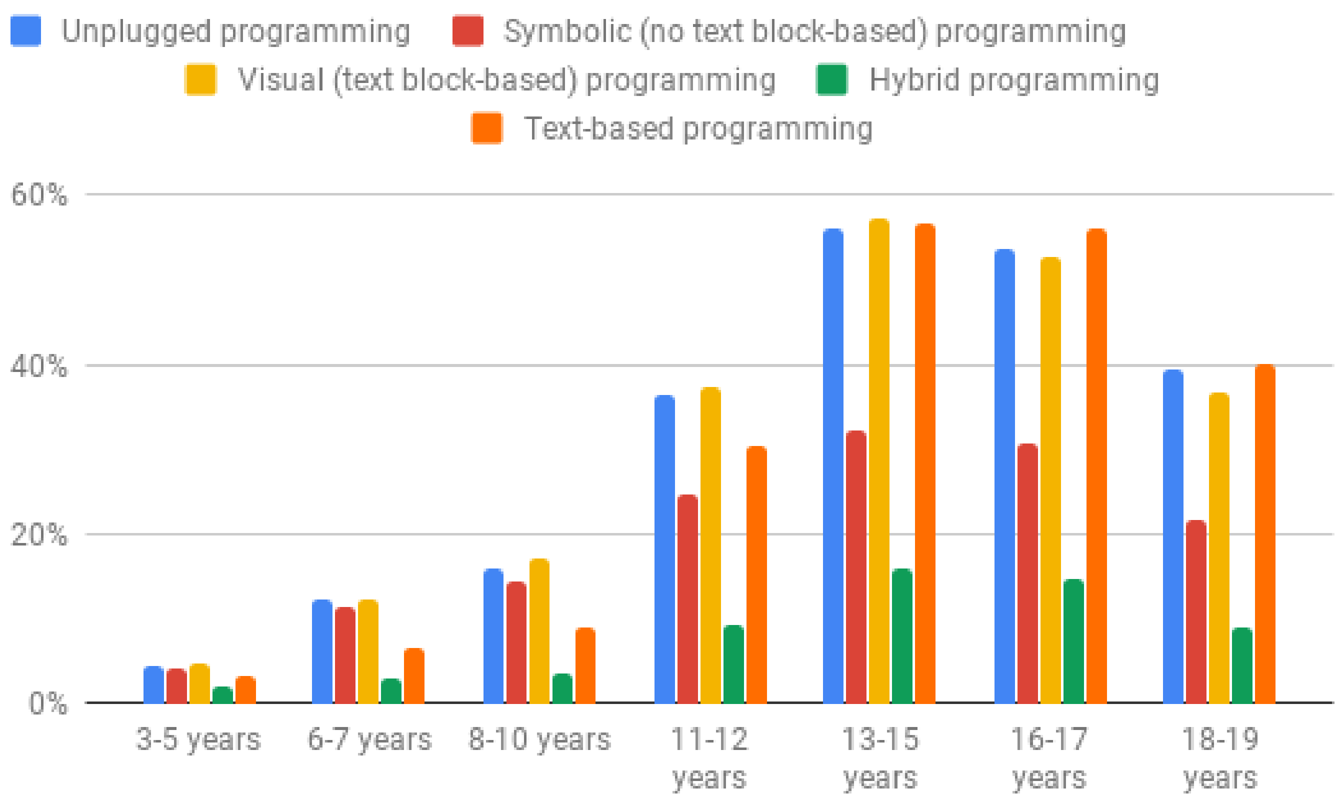


Figure 2. Percentage of teachers using programming environments according to year level bands

Programming paradigms

Exploring programming paradigms as implemented by classroom teachers provides interesting insight as differences might be attributed to a range of reasons such as intended curriculum, but also the resources they have available and student needs. We present results from what programming environments teachers reported implementing in their classrooms according to age groups (see Figure 2). In our analysis we see that programming environments reflected what would be expected across age groups, for example a shift from symbolic programming tools (e.g. Scratch JR) toward block-based environments (e.g. Scratch) and text-based programming environments (e.g. Python). We also noted a strong presence of 'unplugged' programming experiences across age groups

We were then able to take this information about what teachers reported and compare country data we acquired in the country reports that identified any curriculum requirements around programming environments according to age groups.

An example of country-level programming paradigm analysis used England as a test case with the greatest representation of teachers across grade levels. We present teachers'

Ages	Unplugged	Symbolic (no text)	Visual (text)	Hybrid	Text-Based
3-5	8%	**6%**	**10%**	4%	10%
6-7	17%	**15%**	**17%**	6%	13%
8-10	21%	**17%**	**21%**	6%	**13%**
11-12	60%	40%	54%	10%	**58%**
13-15	65%	37%	58%	10%	**73%**
16-17	63%	31%	52%	8%	**67%**
18-19	48%	21%	33%	2%	**48%**

Table 3. Percentage of teachers in England using programming environments according to age group (highlighted cells indicates where the environment is advised in intended curriculum)

reported use in Table 3 with what is identified in the intended curriculum as yellow highlighted cells. Lower percentages of teachers reported using symbolic programming (no text — such as flowcharting, describing sequences as steps, etc.) and visual programming in elementary grades. The lack of teaching with symbolic and visual coding illustrates a mismatch in the intended and enacted curriculum. Higher percentages of teachers reported using text-based programming environments matching more closely the intended curriculum.

Self-esteem of CS teachers

The foundational research conducted by the working group in 2019 captured a wide range of data, thus allowing international comparisons on a multitude of factors, such as the aforementioned enacted curriculum. In 2020, the working group further examined one of the constructs in detail, the teachers' computer science self-esteem (Vivian et al. 2020).

Study background

The Bergin programming self-esteem (Bergin, 2006; Quille & Bergin 2019) construct has shown strong prediction capability in previous studies using CS1 students at third-level, and reported insights when CS1 subcohorts were compared such as by gender, performance, or age (Quille, Culligan, and Bergin, 2017; Quille & Bergin, 2020). The construct was adapted for K-12 teachers and included in the MEasuring TeacheR Enacted Curriculum (METRECC) pilot study. The study consisted of 219 teachers across the seven countries who completed this construct in the METRECC survey. This scale was designed to determine if there were any differences in teachers' computer science self-esteem by country, teacher age, teacher computer science experience, the age groups that are being taught by the teacher, teaching location (rural, metro, etc.), and by gender. The goal of this work was to identify insights that might inform future curriculum developments and teacher PD design and implementation.

Country	N	%
USA	98	45
England	49	22
Italy	18	8
Ireland	19	9
Scotland	17	8
Australia	14	6
Malta	4	2
Total	**219**	**100**

Age Range	N	%
18 - 29	7	3
30 - 39	52	24
40 - 49	70	32
50 - 59	70	32
60 or over	20	9
Total	**219**	**100**

Table 4. Participants per country and age demographics

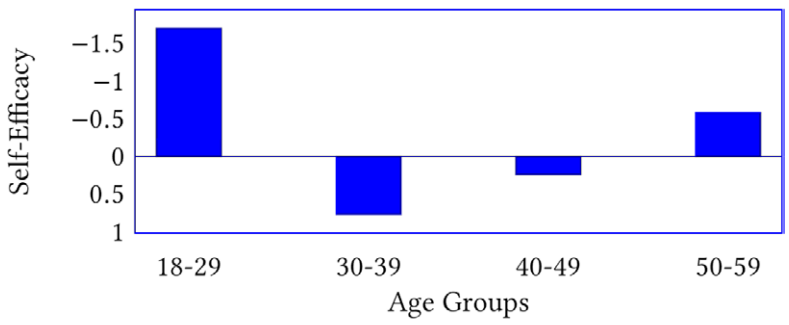


Figure 3. Teachers' computer science self-esteem by age group. (The y-axis scale is inverted, as a negative value represents a positive computer science self-esteem, and a positive value represents a negative computer science self-esteem. This is due to the data reduction algorithm applied, Principal Component Analysis -PCA.)

Results

The participants were primarily from the US and England and were predominantly 40 to 59 years old (see Table 4). Though the majority of teachers were from the US and the UK (n=67%), if student population were considered, the US would be significantly underrepresented. However, as a pilot study, we were evaluating the scale for evidence of reliability and validity.

A comparison was conducted across each of the seven countries. No statistically significant differences between the teachers' computer science self-esteem were found.

Since previous studies identified programming self-esteem differences by student age, this led the working group to investigate across teachers' age groups as used in the METRECC survey. Figure 3 presents the teachers' computer science self-esteem per age group. There was no statistically significant difference between the age groups (using a one-way ANOVA test where F (4, 214) = 1.5485, p = 0.1893. A Tukey HSD post-hoc test was also administered which confirmed the findings of the one-way ANOVA test) while acknowledging that visually there are

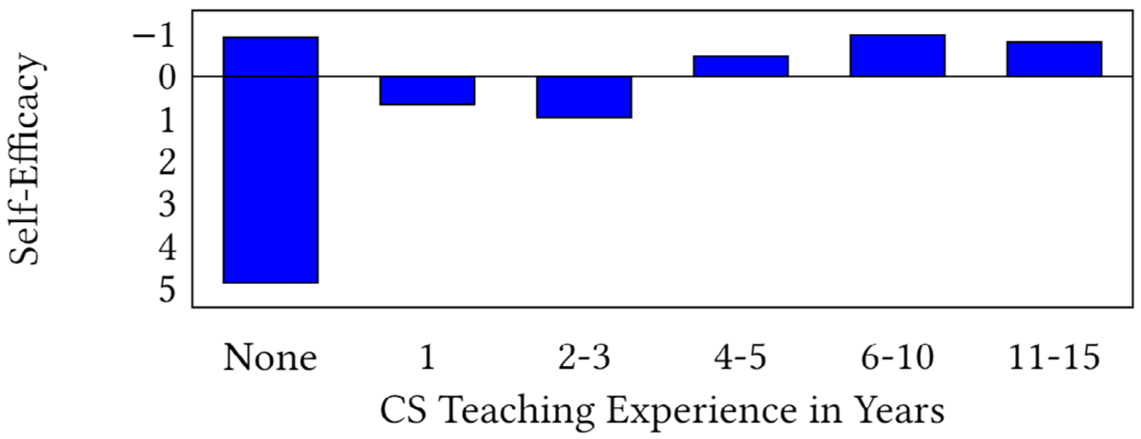


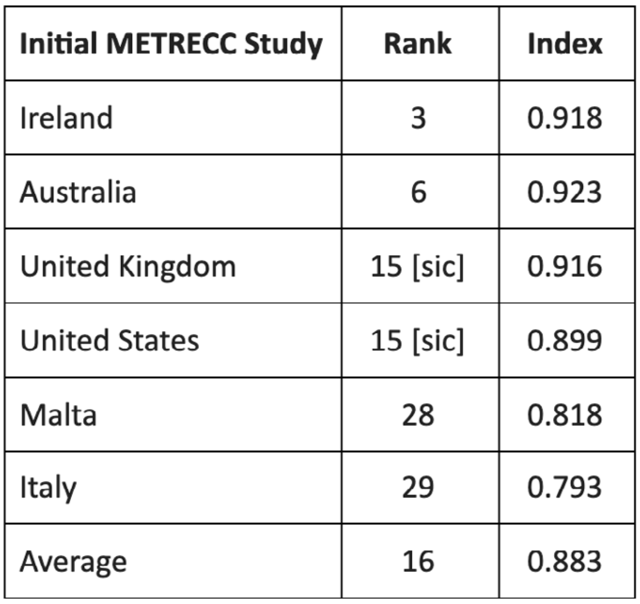
Figure 4. Teachers' computer science self-esteem by CS teaching experience

differences in Figure 3. An interesting finding is that teachers between the age of 30 and 49 have the lowest CS self-esteem. Though the sample size of teachers in the 18 to 29 age group is relatively small, they have the most positive CS self-esteem. Teachers 50 years or older also present a higher positive CS self-esteem to that of the 30 to 49 group. This could suggest a need to focus PD towards specific age groups, however, further work is required to investigate reasons for these age-related variations.

The working group further investigated if the CS teaching experience in years reported differences as reported in the METRECC study. We found a statistically significant difference between groups where a deeper analysis reported that the difference was between the "no experience group" and all other groups. This was expected as K-12 teachers who have not taught a formal CS class would not be expected to have a high CS self-esteem. The results of this comparison are presented in Figure 4. In addition, another insight from Figure 4, illustrates that teachers with four to five years of CS experience or more report a positive computer science self-esteem (where teachers with less than this experience report a negative computer science self-esteem), where perhaps this is the minimum time required to be confident to teach the subject.

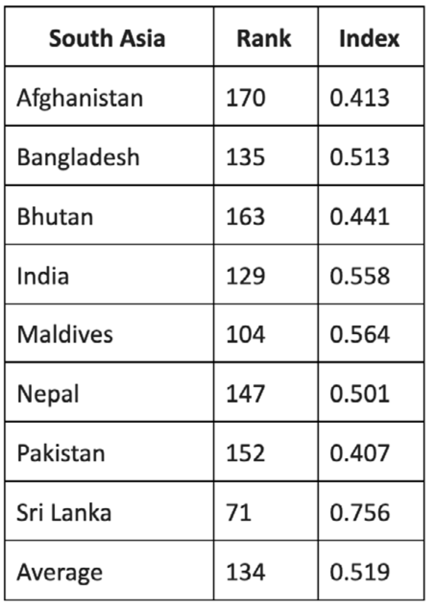
Next the working group examined teachers' computer science self-esteem by the age groups being taught (this was binomial with this analysis categorising teachers by primary- or second-level education) and by teaching location (as categorised by metro, urban, rural, or remote). The difference is statistically significant with primary teachers reporting lower self-esteem than secondary teachers. Future studies could compare the level of CS required for primary- and second-level teachers (where secondary content and depth would be significantly more advanced), to unpack this finding. On a positive note, teaching location did not report significant teachers' computer science self-esteem differences, where PD availability was hypothesised to be a factor for more remote teachers.

Finally, the working group investigated teachers' computer science self-esteem differences based on gender. Previous work has reported for CS1 students significant differences in programming self-esteem (Quille & Bergin, 2017), thus this work was to identify if the same findings were



Initial METRECC Study	Rank	Index
Ireland	3	0.918
Australia	6	0.923
United Kingdom	15 [sic]	0.916
United States	15 [sic]	0.899
Malta	28	0.818
Italy	29	0.793
Average	16	0.883

Table 5. 2018 Education index as specified by United Nations Development Programme for the countries involved in the original METRECC study



South Asia	Rank	Index
Afghanistan	170	0.413
Bangladesh	135	0.513
Bhutan	163	0.441
India	129	0.558
Maldives	104	0.564
Nepal	147	0.501
Pakistan	152	0.407
Sri Lanka	71	0.756
Average	134	0.519

Table 6. 2018 Education index as specified by United Nations Development Programme for South Asian countries

present for K-12 teachers' computer science self-esteem. The difference in teachers' computer science self-esteem reported is statistically significant. These findings align with the aforementioned research (although this study had no metric to examine teacher performance) and prompts questions for future research, such as: "Are male teachers overrating their CS self-esteem, while female teachers underrate theirs?" and "If males and females correctly rate their CS self-esteem, why do female teachers have significantly lower CS self-esteem than males?".

Adapting METRECC for the South Asian classrooms

The research conducted by our working group and highlighted in the previous sections focused on high-income nations, as designated by the World Bank (Table 5) (World Bank, 2020). But, little is known about primary and secondary computing education efforts in both middle- and low-income nations. When examined in aggregate, K-12 computing education research in low and lower-middle income countries is underrepresented in many international publication venues. This can introduce a false understanding as well as biases and prejudice against these nations and paint a deceptive picture that we are achieving more than we actually are on an international scale.

To investigate this, we conducted a follow-up study focusing on K-12 computing education in South Asia, in which limited research on computing education exists and with low- and middle-income countries (Anwar, et al., 2020) (Table 6). Rather than covering all eight countries in South Asia, we instead chose to focus on four: Bangladesh, Nepal, Pakistan, and Sri Lanka. These countries were chosen because they provide a mix of one low-income country (Nepal), two lower-middle income countries (Bangladesh and Pakistan), and one upper-middle income country (Sri Lanka). Further, it gave us a mix of three countries with lower ranked education indices (Bangladesh, Nepal, and Pakistan) and one from a higher ranked index (Sri Lanka). We also note that two authors of this study were from Pakistan, one was from Nepal, and three had connections with Sri Lanka and Bangladesh.

To gauge what types of enacted curriculum are taught in the classrooms, we carefully modified METRECC to meet the cultural and current educational climate of these countries. Before trialing the new METRECC South Asia instrument, we modified it based on our cultural knowledge of the countries, then acquired feedback from professionals in South Asia and modified the instrument again based on their input. We then tested this instrument across Nepal and Pakistan. Based on the results and the face validity performed on the instrument, we consider METRECC South Asia to be ready for larger scale usage with recommendations for a broader study. In addition, we consider the role and importance of computing education research in low-income countries in order to support the beliefs and values of the CS for All movement.

In light of the social justice and economic prospects promoted through the CS for All initiative across high-income countries, our literature fails to step back and understand the tens of millions of students in low and lower-middle income countries who deserve the opportunity to lift their families and their countries from their low-income status. Exploration of other ways to improve and to enable low-income countries to achieve their technological aspirations as noted in several plans (Ministry of Education Sri Lanka, 2015; Government of Pakistan, 2018) could prove to be invaluable in enabling these countries to address gaps in their education (Coloma and Harris, 2009; Wikramanyake, 2014). These nations' leaders understand that CS for All is vital to their nation's growth. Therefore, though our research into this area is just starting, we hope that this study gives readers a chance to reflect and consider how we can be more aware of the CS for All movement across the full international landscape.

Conclusion and future work

This chapter brings together the original enacted curriculum investigation work (the pilot study administered during the development of the METRECC instrument) with two follow-up bodies of work, including expanding to efforts in both middle- and low-income nations. This work seeks to understand the intended curriculum and enacted curriculum to determine if what is intended to be taught through policy, curriculum documents, or other required mandates are being implemented by classroom teachers. We identified in our country analysis differences between what is expected to be taught across countries in terms of CS topics and programming environments, as well as differences between what is expected to be taught and what teachers reported implementing in classroom learning.

This work provided some early insights on teachers' computer science self-esteem, but it was a pilot in its nature based on teacher sample size and jurisdiction representation. More work is required to conduct a broader study and re-validate the findings. In addition, a deeper analysis is required (such as qualitative data collection, interviews, etc.) to unpack the insights presented in this section of work. These preliminary findings, however, could be valuable to professional learning developers, for developing a more differentiated suite of professional learning sessions.

This work has demonstrated the value in including both middle- and low-income nations in the design and development of instruments to capture and monitor intended and enacted curriculum across the world, however, more work can be done to extend METRECC to be customised for other contexts. By continuing to capture more countries and to replicate METRECC, we will be able to gather more comparison data and to monitor changes over time. Throughout this work we have identified some differences between what is intended to be taught and what is taught, as well as some slight differences in teacher confidence. This opens up more questions and opportunities to further explore the METRECC dataset and to continue work to investigate the potential reasons for these differences, for example whether it be due to qualifications, professional learning, resources, or other factors.

This body of work highlights the importance of international researchers working together with a strategic effort to monitor what is happening in CS classrooms around the world. By working together and collaboratively developing and validating instruments, we are working toward a common goal and more robust and consistent ways of being able to capture and monitor our CS education landscapes.

References

Anwar, T., Jimenez, A., Bin Najeeb, A., Upadhyaya, B., & McGill, M. M. (2020, August). Exploring the enacted computing curriculum in K-12 Schools in South Asia: Bangladesh, Nepal, Pakistan, and Sri Lanka. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 79-90).

Balanskat, A., & Engelhardt, K. (2014). Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe. European Schoolnet.

Bergin, S (2006). *A computational model to predict programming performance*. PhD thesis, Department of Computer Science, Maynooth University.

Coloma, J., & Harris, E. (2009). From construction workers to architects: developing scientific research capacity in low-income countries. *PLoS Biol*, 7(7), e1000156.

Falkner, K., Sentance, S., Vivian, R., Barksdale, S., Busuttil, L., Cole, E., … Quille, K. (2019a). An international study piloting the measuring teacher enacted computing curriculum (METRECC) instrument. In *Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE (pp. 111–142). https://doi.org/10.1145/3344429.3372505

Falkner, K., Sentance, S., Vivian, R., Barksdale, S., Busuttil, L., Cole, E., … Quille, K. (2019b). An international comparison of K-12 computer science education intended and enacted curricula. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 1–10). New York, NY, USA: ACM. https://doi.org/10.1145/3364510.3364517

Government of Pakistan. (2018). *Ministry of Information & Broadcasting Government of Pakistan*. Retrieved from http://moib.gov.pk/ on February 20, 2020.

Hubwieser, P., Armoni, M., & Giannakos, M. N. (2015). How to implement rigorous computer science education in K-12 schools? Some answers and many questions. *ACM Transactions on Computing Education*, 15(2), 1–12. https://doi.org/10.1145/2729983

Ministry of Education Sri Lanka. (2015). Moe.gov.lk. Retrieved from https://web.archive.org/web/20191126145533/http://www.moe.gov.lk/english/images/subject_related/ICT/ALsyllabusrevised.pdf on March 10, 2021

Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., … Meyer, B. (2013). *Informatics Education: Europe Cannot Afford to Miss the Boat. ACM Europe: Informatics Education Report*. New York.

Hong, H., Wang, J., & Moghadam, S. H. (2016). K-12 Computer science education across the U.S. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 142–154). https://doi.org/10.1007/978-3-319-46747-4_12

Crick, T., & Moller, F. (2016). A national engagement model for developing computer science education in Wales. In *9th International Conference on Informatics in Schools (ISSEP)* (pp. 1–13). Munster, Germany.

Porter, A. C., & Smithson, J. L. (2001). *Defining, developing, and using curriculum indicators. CPRE Research Reports*.

Quille, K., Culligan, N., & Bergin, S. (2017). Insights on gender differences in CS1. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 263–268). New York, NY, USA: ACM. https://doi.org/10.1145/3059009.3059048

Quille. K & Bergin, S (2019) CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education*, 29:2-3, 254-282, DOI: 10.1080/08993408.2019.1612679

Quille, K. & Bergin, S. (2020). Promoting a growth mindset in CS1: Does one size fit all? A pilot study. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE '20). Association for Computing Machinery, New York, NY, USA, 12–18. DOI: https://doi.org/10.1145/3341525.3387361

Sysło, M. M., & Kwiatkowska, A. B. (2015). Introducing a new computer science curriculum for all school levels in Poland. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 141–154). https://doi.org/10.1007/978-3-319-25396-1_13

The Royal Society. (2012). *Shut down or restart? The way forward for computing in UK schools*. Technical report. London. Retrieved from https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf

The Royal Society. (2017). *After the reboot: Computing education in UK schools*. Technical report. London. Retrieved from https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf

Vivian, R., Quille, K., McGill, M. M., Falkner, K., Sentance, S., Barksdale, S., … Maiorana, F. (2020). An international pilot study of K-12 teachers' computer science self-esteem. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 117–123). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3341525.3387418

Wikramanayake, G. N. (2005). Impact of digital technology on education. In *24th National Information Technology Conference* (pp. 82–91).

World Bank. (2020). *World Bank Country and Lending Groups*. Retrieved from https://datahelpdesk.worldbank.org/knowledgebase/articles/906519-world-bank-country-and-lending-groups, February 26, 2020

Section 1: Computing in context

Factors that impact gender balance in computing

Katharine Childs (Raspberry Pi Foundation, UK)

Abstract

The use of digital technology is pervasive in almost every part of our lives, and careers which require advanced computing skills are amongst the fastest-growing sectors globally. Learning computer science and digital skills offers young people the opportunity of a career in a flourishing sector, yet a lack of gender equity has been identified as a consistent and enduring issue which prevents girls from fully participating in these opportunities. In this short paper, I briefly review what we have learned to date from the literature on gender balance in computing education and outline some of the key barriers to full participation across genders: sense of belonging, relevance to self, and attitudes to technology. The use of collaborative teaching approaches to facilitate engagement and increase gender balance is also highlighted.

Introduction

A lack of gender equity in the uptake of both computing and wider STEM subjects has been identified as a consistent and enduring issue (Royal Society, 2017). There has been a considerable amount of literature published in the last twenty years that aims to explain why so few girls choose computing and then outline theories or interventions that could make a change to the current educational landscape. In this paper I reviewed the literature in this area to identify key factors that influence gender balance in computing.

The literature review was conducted with a search for terms relating to gender balance in computing to inform the implementation of the interventions with the most up-to-date evidence. In order to use robust and rigorous findings, only peer-reviewed journal papers or published conference proceedings were included. The ACM digital library (dl.acm.org) was selected as the primary database for the search. The scope of the literature survey was research published from January 1995 to June 2020, and included studies which showed the potential to transfer from STEM subjects to computing. Research conducted with university undergraduates as well as primary and secondary (K-12) pupils were included to identify any emerging findings in higher education which had the potential to be explored with younger cohorts.

Throughout this report, the term 'gender' is used as in the following definition: “either of the two sexes (male and female), especially when considered with reference to social and cultural differences rather than biological ones. The term is also used more broadly to denote a range of identities that do not correspond to established ideas of male and female” (Lexico, 2021, para 1).

Attitudes

Many studies have identified gender differences between learners in their attitudes towards computing. These attitudes include beliefs about the type of person who achieves well in computing, perceptions about the specialist nature of the subject content, and opinions about the potential for using computing skills in future careers. In this review, the connections between attitudes and subject choice will be explored first, followed by a more detailed consideration of the causes and impact of gender differences in attitudes towards computing.

Learners' attitudes are defined as the evaluative judgements which they hold about a curriculum subject, whether these be positive or negative, strong or weak, and formed from thoughts, feelings, or prior experiences (Maio & Haddock, 2009). There are a number of theoretical frameworks that can be used to understand students' attitudes towards computing and how likely they are to persist in the discipline (Denner & Campe, 2018). For example, expectancy-value theory (Eccles et al., 1998) suggests that subject choice and career goals are affected by the perceptions that an individual has of parents' and society's attitudes towards the subject. If a female student does not perceive the subject or career to be valued by others, she is less likely to value it herself and may focus time and energy on other subjects which are more highly valued. Both expectancy-value theory and social cognitive career theory (Lent et al., 2008) also highlight the role of a student's expectations of success or 'self-efficacy' (Bandura, 1999) on their persistence in computing: students are more likely to choose computing if they believe they will succeed and if they have a sense of support from those around them (Lent et al., 2008). The theories highlight the importance of interventions focusing on a range of individual, environmental, and societal factors to improve the gender balance in computing.

Attitudes towards a subject can be seen as a precursor to learners' motivation to succeed in them, and this has been notable in the work of Cheryan et al. (2009) who showed that when students held a positive attitude towards computing, they were more likely to be motivated to participate in further computing study. Wider research in STEM subjects has also shown that pupils' attitudes play an important part in shaping educational choices. Else-Quest et al. (2013) found significant gender differences in attitude towards mathematics which were also an accurate predictor of education-related choices. They suggested that the lower self-concept reported by girls in mathematics would reduce the chance of them choosing it for further study because they did not believe that they would achieve well. In the context of computing study, Goode et al. (2018) examined similar connections with high school computer science pupils (aged 14 to 18). Through the use of data drawn from qualitative case studies, they suggested that female students experienced an accumulation of negative experiences in computing classes. They cited examples such as lack of contextual information to link computing to the real world and pedagogy without a higher-order thinking focus which had affected girls' attitudes towards computing in an unfavourable way.

There is a clear need to examine more closely which factors influence female pupils when they form opinions about a subject and to identify possible interventions which will either augment existing positive connotations about computing or change attitudes towards the subject by illuminating new possibilities.

Do I belong?

Girls' interest and motivation in STEM subjects can be predicted by their sense of belonging in the subject. When students attend classes in subjects they have chosen to study, they create a group, or community, who are working towards a common goal to achieve a formal qualification in that subject. Evidence suggests that a sense of belonging develops from both the extent to which girls feel that they fit into the community and also how they perceive that they are valued and accepted by other members of the community (Good et al., 2012). In the workplace, women's sense of belonging to computing as a career is affected by the effort they perceive they need to exert in order to succeed. Smith et al. (2013) found that women who worked in STEM subjects thought that they would have to expend more effort than others to do well and suggested that this may affect the extent to which women feel that they belong in the STEM field of careers. Some girls face barriers to taking part in computing because they feel that they do not belong there and this can be improved.

Research into increasing girls' sense of belonging often draws on theories from the field of psychology. An example of this is self-determination theory (SDT), which suggests that students have three basic needs in order to sustain motivation and persistence in any given subject: competency, autonomy, and relatedness. Mishkin (2019) applied SDT to female high school computing students (aged 14 to 18) and found that of the three needs, the feeling of being related to others was the most important condition for girls' motivation to study computing, and that a sense of belonging was a significant predictor of their motivation. This reinforces the idea that although girls typically achieve well in computing, they do not choose to study it because they see themselves as isolated or unwelcome amongst other computing students.

This need for a sense of belonging can be problematic for gender balance in computing because it creates a repetitive cycle of female inequity. Girls and women do not see themselves represented in the field and are therefore not motivated to pursue it, which then perpetuates the lack of representation and means that future generations do not feel that they fit into the community either. One way of breaking this cycle is to explicitly call out existing representations of female computing students or women in computing careers as role models for the next generation. The term 'role model' is generally used to describe an individual who displays behaviour, attitudes, or achievements that can be emulated by others.

The literature survey revealed considerable variation amongst studies about the use of computing role models. Black et al. (2011) distributed a booklet containing the stories of successful women in computing to secondary school pupils in order to inspire and encourage female students to consider computing as a career. Role models in this study were presented as people with achievements that could be admired and followed. This approach contrasts markedly with research conducted by Townsend (1996), who created videos describing the journey of female undergraduates including how they had juggled childcare responsibilities and overcome fears or adversity to achieve success. In this way, the attitudes, behaviours, and achievements of the role models were presented together as a coherent whole. It is difficult to draw any conclusions on whether one approach was more effective than the other, as the studies lacked any commonality in measuring their outcomes. Black et al. (2011) used a mixed-methods approach based on qualitative access figures and quantitative teacher feedback, whereas Townsend (1996) used quantitative data sampled from undergraduate students to compare between a control and treatment group. The variety of evaluation methods used highlights the importance of careful trial design in order to effectively and confidently measure the impact of an intervention.

A variety of places to find role models was also highlighted in the literature. A common theme was to introduce a gender-balanced group of undergraduate students to primary and secondary pupils, either to teach computing lessons or to speak about their experiences. Such an approach was found to help pupils perceive computing as a subject that was equally difficult for girls as well as boys (Zaidi et al., 2017) and to increase girls' self-efficacy in computing (Lang, et al., 2010). This approach was tested on a small scale due to the logistical practicalities of matching students with classrooms in both studies. It contrasts with the larger-scale research conducted by Black et al. (2011) which drew on role models from history, workplaces, and classrooms, as well as first-person accounts to create a booklet for mass distribution. As mentioned previously, the variety of research design, tools, and instruments in these studies means it is difficult to draw any conclusions about whether any approach was more effective than another. Further research could provide insight into this through the use of a scalable trial design and a reliable, validated evaluation instrument.

Young people choose their own role models; teachers cannot choose role models for them. There is also a gender difference to take into account, wherein female students choose role models with a higher number of self-perceived likenesses to themselves than male students do (Wohlford et al., 2004). It was notable that self-esteem was also a predictor of female students' chosen role models. This suggests that some high-achieving role models may provide the opposite effect from that which was intended, and deter girls from emulating the individual because they feel that the achievements are too far out of reach. At the other extreme, if girls select role models based on perceived likeness, they may focus on the people around them, such as friends and family, and this may not provide them with any examples of women in computing at all.

There have been investigations in other STEM subjects relating to the influence of role models on girls' attitudes towards the subject. One very recent study looks at the effects of a role-model intervention in maths with girls aged 12 to 16 years old. It links to Eccles' (1998) expectancy-value theory to measure the effect that the intervention had on girls' personal enjoyment of maths and the importance they attached to maths. The intervention provided a significant increase in both enjoyment and importance, and the authors concluded that it was important to introduce such interventions at a relatively young age before pupils begin to make specialist academic choices (Gonzalez et al., 2020).

Children and young people are also influenced by parents and other family members when they make choices. Eccles' (1998) expectancy-value theory also states that young people's attitudes towards a subject are influenced by their perceptions of the values their parents have about that subject. Where parents are seen to support their children's choices in computing, girls are more likely to express interest in computing as a career (Creamer et al., 2004; Denner, 2011). Some parents may hold less traditional attitudes about gender roles and have daughters who are more likely to pursue nontraditional careers such as computing (Chhin et al., 2008). There is room to further explore the role that parental encouragement plays based on evidence which suggests that interventions based on positive messaging from parents show a positive influence on their children's attitudes (Kraft & Rogers, 2014).

There is still some ambiguity in the literature around the effects that role-model interventions and parental encouragement have on girls' attitudes towards computing. Although studies have provided evidence of their effectiveness on a small scale with innovative approaches, there are still questions to be addressed around both intervention design and trial methodology. Future research could explore the impact of parental encouragement and the impact of introducing role models on girls' sense of belonging in computing.

Relevance to me

The use of technology is pervasive in almost every aspect of our daily lives. This provides opportunities for educators to show how studying computing can be relevant for many jobs and careers and, specifically, how learning computer science skills can be applied to everyday situations. Learning to program then moves away from acquiring the skills to write code towards the ability to be able to create authentic applications such as games, stories, and mobile phone apps that can be used outside of the classroom in the real world (Kafai & Burke, 2013).

Computer science can be perceived as a very abstract subject, in which there is a great deal to learn about programming concepts in order to use them to efficiently write code. However, Fisher and Margolis (2003) identified that the contexts in which computer science skills can be used are important for female students. Through a series of longitudinal surveys, they observed gender differences in students' motivations for studying computer science at university. Female undergraduates were much more likely to identify links between their learning and other disciplines, whereas male students were more invested in the value of computer science as a subject in itself. Subsequent studies have drawn on this finding to explore a variety of different ways to introduce real-world contexts into computing lessons.

Four principles were proposed by Guzdial and Tew (2006) to contextualise computing so that students could connect their learning to their prior experiences and future expectations:

1. Learning activities were aligned with real-world scenarios
2. Topics were aligned with students' own interests
3. Assessments were aligned with the material which had been taught
4. The methods of inquiry used in the classroom were aligned with professional standards in the workplace

The first and second of these principles were applied to two introductory programming modules for undergraduates which situated learning to program in the contexts of media manipulation and computer-generated animation sequences. Guzdial and Tew (2006) hypothesised that students would perceive the course to be of value because of the explicit links to real-world scenarios. Although they did not specifically set out to create a learning experience which would appeal to female students, it is notable that when averaged out over several presentations of the modules, 51% of students were female, which reinforces the findings from Fisher and Margolis (2003).

Subsequent studies have explored further ways that computing can be made relevant by introducing the idea that computing is a tool for bringing societal benefits to others. Dewitt et al. (2017) built upon the links between programming and media generation in a project for boys and girls at a summer camp, where they were tasked with creating a piece of artwork to address a social issue. This led to a positive increase in attitudes towards computing amongst both boys and girls. This finding seems to contrast with a study that compared university student opinions about humanitarian contexts, practical contexts, and games-based contexts in computer science courses (Rader et al., 2011). When asked to rank assignments in order of preference, students preferred assignments using games-based contexts. However, the authors do acknowledge the very low number of female students amongst the respondents and so greater value may be found in Rader et al.'s (2011) finding that 79% of students expressed a positive opinion about programming assignments which showed how computer science could benefit society.

Finally, computing is not just relevant in social contexts. Franklin et al. (2011) explored how learning to program could be made culturally-relevant for middle school students in the US. Using the theme of Animal Tlatoque, they successfully recruited a group of students who were previously unengaged in computer science, and subsequently found that this small group (n=34) became more interested in computing as a career and were more likely to view computing as a subject for girls.

The variety of approaches taken within the literature towards defining relevant computing contexts is perhaps indicative of the lack of depth in computer science education research to date. It is entirely plausible that in order to situate learning about computing in a relevant way to interest girls, an approach which draws upon the deeper insights from research into other STEM subjects is needed. Lyons (2006) recommended that science curricula were more likely to interest girls if they provided opportunities for genuine inquiry, involved real-world experience and integrated social and scientific issues, as well as opportunities for experimentation, practice, reflection, and conceptualisation. Thinking about computing, the relevance of the subject in society and the opportunity to apply computing skills to solve real-world problems need to be carefully embedded within a curriculum so that girls can see that computer science has many potential applications in future study and careers.

Learning together

An emerging body of evidence suggests that collaborative teaching approaches can engage more girls with computing. This is of particular interest when learning to write computer programs, which can be seen as the most difficult section of the computing curriculum for learners (Kallia & Sentance, 2018). Introducing a shared, group approach requires a shift from traditional computing pedagogy. Learning to code changes from a series of tasks undertaken by individuals, to a sociocultural experience in which students work together to create and share digital content (Kafai & Burke, 2013). The initial inspection of the research and subsequent literature survey found evidence to support two collaborative teaching approaches in learning to program which merited further investigation: pair programming and peer instruction.

Pair programming

The idea of writing computer programs in pairs has been directly drawn from industry, where colleagues often work in partnership to write and review code in order to maximise the quality and design of a program (McDowell et al., 2006). Transferring a workplace practice into a classroom teaching approach offers pupils an authentic experience of real-world programming. The idea of paired work is commonly used in many other subjects, where pupils discuss ideas or contribute towards a shared piece of work. Pair programming differs from other paired work as it has a defined structure. In pair programming, one student takes the role of ‘driver’ and has control of the keyboard and mouse to write the code. The other student is the ‘navigator’ who reads out the instructions and monitors the code for errors (McDowell et al., 2006). The teacher’s role includes training the students in successful pair interactions and ensuring the pairs rotate after a given time so that each student experiences both roles. The success of pair interactions is actively managed by the teacher as well as being evaluated by the pairs themselves (Williams et al., 2008).

Werner et al. (2004) advocated for the use of pair programming in introductory university programming courses based on their findings that collaborative work had a positive impact on female students’ perceptions of computing as a subject for further study. Pair programming has been shown to improve student confidence and have a positive impact on student retention in computing and has also demonstrated that the quality of programs written in pairs is significantly higher than those written individually in an introductory undergraduate course (McDowell et al., 2006). Similar findings in K-12 (primary and secondary) environments demonstrated that pair programming generally increased pupil attitudes and confidence towards computing (Denner et al., 2014). This suggests that using pair programming has potential to be used as an inclusive pedagogy to benefit girls' perceptions of computing, whilst also supporting all learners to develop skills and knowledge of programming concepts.

Further research has explored different approaches to pairing pupils along with how those pairs might affect the interactions which take place between partners. A small-scale study from Tsan et al. (2016) suggested that by the age of 11, all-female pairings were producing significantly lower program quality than mixed or all-boy pairs. However, this study was limited in focus and only assessed the quality of the completed artefact rather than pedagogy required to promote high-quality outcomes. Ruvalcaba et al. (2016) noted that ethnicity of pairs may affect the types of interaction between pairs, with Latina/o students more likely to use non-verbal communication signals to interact with their partner whereas white and mixed student pairs relied more on verbal communication to check understanding. Both studies indicate that pair programming requires careful training of teachers to ensure that the whole pedagogy is understood and applied, without bias.

The use of pair programming as a teaching approach in schools is likely to appeal to girls, and make them more likely to both choose a subject and achieve well in it. Further research can help strengthen the evidence of how to effectively pair pupils, provide guidance to teachers on the characteristics of successful pairings, and demonstrate the value of this pedagogy within the specific context of the English school system.

Peer instruction

The idea of working together with peers to build knowledge has been explored in the literature relating to both computing and wider STEM subjects.

Peer instruction is an approach which was developed by Mazur (1997) through a series of studies conducted with physics undergraduates. In these studies, peer instruction was positioned as a pedagogy to help students understand difficult concepts through classroom interaction. Lessons were structured around a series of multiple choice questions (MCQs) which the teacher could devise to stimulate discussion around physics concepts. These concepts would be first of all introduced using an instructor-led presentation, followed by a series of MCQs which students could answer with electronic clickers or flashcards (Vickrey et al., 2015). The instructor assessed the understanding of the class through the MCQ scores and chose to briefly recap the answer if a large proportion of class understood, or else to initiate paired or group discussion of the question so that students could evaluate the options presented and work out which one was correct together (Watkins & Mazur, 2013).

Watkins and Mazur (2013) highlighted that the use of peer instruction in introductory STEM courses led to increased retention of students in STEM disciplines during subsequent intermediate and advanced courses. The authors proposed several reasons for this improvement. First, the pedagogy included inherent formative assessment and so instructors were better able to adapt their teaching to meet student needs. Secondly, students responded well to the opportunity to interact and discuss with their peers, and developed their fluency in explaining scientific concepts. Finally, an additional outcome was to increase student self-efficacy, which promoted a positive attitude towards further study of STEM courses. A separate study conducted in an introductory physics course at Harvard University investigated the effect of peer instruction on student achievement (Lorenzo et al., 2006). An intervention which used peer instruction was delivered to 1,048 physics students and was evaluated qualitatively using pre- and post-tests. A statistically significant gender gap in the pre-instruction test scores was reported following the intervention, and Lorenzo et al. (2006) attributed this to the interactive and collaborative nature of the teaching approach which helped to create a classroom environment that supported both genders.

Although research on collaborative teaching approaches and gender balance is as yet limited, the research to date signals this as a worthwhile area to explore further.

Discussion

The literature discussed here describes an accumulation of historical, social, and cultural barriers faced by girls in the computing classroom which have developed alongside the growth of computing as a subject in schools.

The concept of an incredible shrinking pipeline (Camp, 2002) has been used to describe the decreasing number of girls involved in each stage of computing education. However, it has been noted that there are too few girls entering the pipeline of computing qualifications initially, and so it would seem insufficient to direct efforts into research that aims to retain female students from GCSE through to A level and beyond. This report recommends building on research which presents computing as an equitable subject that is relevant, applicable, and achievable to all pupils, regardless of gender. Because pupils make subject choices in England at the relatively young age of 14, a range of interventions in both primary and secondary phases are suggested in order to present computing as a subject where girls can succeed.

This review has underlined the importance of girls' attitudes towards computing and their motivations for studying the subject. As with other STEM subjects, computing has acquired an image of being a subject which is taken by 'geeky' students to be used in a very specialist way in jobs and careers (Creamer et al., 2004). Whereas other sciences have had to unpick layers of inequity which have built up over decades, computing is a relatively young subject and this offers the opportunity to identify robust changes which can be made rapidly to alter the gender imbalance currently seen in the subject.

It has been highlighted that the sociocultural context of learning computing appears to play an important role in shaping girls' attitudes (Denner, 2011; Kafai & Burke, 2013). Underpinning research with learning science theories relating to attitudes, beliefs, and motivation can provide a rigorous approach to measuring changes in attitude. Much of the initial work to explore girls' attitudes towards computing and identify the obstacles which prevent them from participating in the field has been conducted in the US (e.g. Fisher & Margolis, 2003) where girls are similarly underrepresented in computing study and careers.

What next?

The Gender Balance in Computing project has been funded by the Department for Education in England from 2019 to 2022 to examine the key factors affecting pupil attitudes towards computing early in their education, and to identify promising approaches which have the potential to be delivered at scale in a wide variety of educational settings, through a series of randomised-controlled trials. More information can be found at https://teachcomputing.org/gender-balance. The results from this project promise to add to our understanding of what specific actions we can take to address the factors identified here.

References

Bandura, A. (1999). Moral disengagement in the perpetration of inhumanities. Personality and social psychology review, 3(3), 193-209.

Black, J., Curzon, P., Myketiak, C., & McOwan, P. W. (2011). A Study in Engaging Female Students in Computer Science Using Role Models. *ITiCSE'11 - Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science*, 63–67. Darmstadt, Germany: ACM.

Camp, T. (2002). The incredible shrinking pipeline. *ACM SIGCSE Bulletin*, 34(2), 129-134.

Cheryan, S., Plaut, V. C., Davies, P. G., & Steele, C. M. (2009). Ambient belonging: how stereotypical cues impact gender participation in computer science. *Journal of personality and social psychology*, 97(6), 1045.

Chhin, C. S., Bleeker, M. M., & Jacobs, J. E. (2008). Gender-typed occupational choices: The long-term impact of parents' beliefs and expectations.

Chipman, H., Adams, H., Sanders, B., & Larkins, D. (2018). Evaluating computer science camp topics in increasing girls' confidence in computer science. *Journal of Computing Sciences in Colleges*, 33(5), 70–78.

Creamer, E. G., Burger, C. J., & Meszaros, P. S. (2004). Characteristics of High School and College Women interested in Information Technology. *Journal of Women and Minorities in Science and Engineering*, 10(1), 67–78. https://doi.org/10.1615/jwomenminorscieneng.v10.i1.50

Denner, J. (2011). What Predicts Middle School Girls' Interest in Computing? *International Journal of Gender, Science and Technology*, 3(1), 53–69. Retrieved from http://genderandset.open.ac.uk

Denner, J., & Campe, S. (2018). Equity and Inclusion in Computer Science Education. In S. Sentance, E. Barendsen, & C. Schulte (Eds.). Computer Science Education: Perspectives on Teaching and Learning in School, pp. 189-205. Bloomsbury Academic: London.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, 46(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

DeWitt, A., Fay, J., Goldman, M., Nicolson, E., Oyolu, L., Resch, L., … Rebelsky, S. A. (2017). Arts coding for social good a pilot project for middle-school outreach. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, 159–164. https://doi.org/10.1145/3017680.3017795

Eccles, J. S., Wigfield, A., & Schiefele, U. (1998). Motivation to Succeed. Handbook of Child Psychology: Vol 3. Social, emotional, and personality development. N. Eisenberg.

Else-Quest, N. M., Mineo, C. C., & Higgins, A. (2013). Math and Science Attitudes and Achievement at the Intersection of Gender and Ethnicity. *Psychology of Women Quarterly*, 37(3), 293–309. https://doi.org/10.1177/0361684313480694

Fisher, A., & Margolis, J. (2003). Unlocking the clubhouse. *ACM SIGCSE Bulletin*, 35(1), .23. https://doi.org/10.1145/792548.611896

Franklin, D., Conrad, P., Aldana, G., & Hough, S. (2011). Animal Tlatoque: Attracting Middle School Students to Computing through Culturally-Relevant Themes. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 453–458. Dallas, Texas: ACM.

Gonzalez, S., Mateos de Cabo, R., & Sáinz, M. (2020). Girls in STEM: Is It a Female Role Model Thing?. *Available at SSRN 3541939*.

Good, C., Rattan, A., & Dweck, C. S. (2012). Why do women opt out? Sense of belonging and women's representation in mathematics. *Journal of Personality and Social Psychology*, 102(4), 700–717. https://doi.org/10.1037/a0026659

Goode, J., Estrella, R., & Margolis, J. (2018). Lost in Translation: Gender and High School Computer Science. In *Women and Information Technology*. https://doi.org/10.7551/mitpress/7272.003.0005

Guzdial, M., & Tew, A. E. (2006). Imagineering inauthentic legitimate peripheral participation: An instructional design approach for motivating computing education. *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop, 2006*, 51–58. https://doi.org/10.1145/1151588.1151597

Kafai, Y. B., & Burke, Q. (2013). *The social turn in K-12 programming*. 603. https://doi.org/10.1145/2445196.2445373

Kallia, M., & Sentance, S. (2018). Are boys more confident than girls? The role of calibration and students' self-efficacy in programming tasks and computer science. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3265757.3265773

Kraft, M. A., & Rogers, T. (2014). Teacher-to-Parent Communication: Experimental Evidence from a Low-Cost Communication Policy. *Society for Research on Educational Effectiveness*.

Lang, C., Craig, A., Fisher, J., & Forgasz, H. (2010). Creating Digital Divas – Scaffolding Perception Change Through Secondary School and University Alliances. *ITiCSE 2010*, 38–42. ACM.

Lent, R. W., Lopez Jr, A. M., Lopez, F. G., & Sheu, H. B. (2008). Social cognitive career theory and the prediction of interests and choice goals in the computing disciplines. Journal of Vocational Behavior, 73(1), 52-62.

Lexico (2021) *Gender*. www.lexico.com/definition/gender

Lorenzo, M., Crouch, C. H., & Mazur, E. (2006). Reducing the gender gap in the physics classroom. *American Journal of Physics*, 74(2), 118–122. https://doi.org/10.1119/1.2162549

References

Lyons, T. (2006). Different countries, same science classes: Students' experiences of school science in their own words. *International Journal of Science Education, 28*(6), 591–613. https://doi.org/10.1080/09500690500339621

Maio, G. R. & Haddock, G. (2009). *The psychology of attitudes and attitude change*. Sage Publications Limited.

Mazur, E. (1997). Peer instruction: A user's manual. Upper Saddle River, NJ: Prentice Hall.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, Vol. 49, pp. 90–95. https://doi.org/10.1145/1145287.1145293

Mishkin, A. (2019). Applying self-determination theory towards motivating young women in computer science. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1025–1031. https://doi.org/10.1145/3287324.3287389

Porter, L., Bailey Lee, C., Simon, B., & Zingaro, D. (2011). Peer instruction: Do students really learn from peer discussion in computing? ICER'11 - Proceedings of the *ACM SIGCSE 2011 International Computing Education Research Workshop*, 45–52. https://doi.org/10.1145/2016911.2016923

Rader, C., Hakkarinen, D., Moskal, B. M., & Hellman, K. (2011). Exploring the Appeal of Socially Relevant Computing: Are Students Interested in Socially Relevant Problems? *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 423–428. Dallas, Texas: ACM.

Royal Society. (2017). *After the reboot: computing education in UK schools*. https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf

Ruvalcaba, O., Werner, L., & Denner, J. (2016). Observations of pair programming: Variations in collaboration across demographic groups. SIGCSE 2016 - *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 90–95. https://doi.org/10.1145/2839509.2844558

Smith, J. L., Lewis, K. L., Hawthorne, L., & Hodges, S. D. (2013). When Trying Hard Isn't Natural: Women's Belonging With and Motivation for Male-Dominated STEM Fields As a Function of Effort Expenditure Concerns. *Personality and Social Psychology Bulletin, 39*(2), 131–143. https://doi.org/10.1177/0146167212468332

Townsend, G. C. (1996). Viewing video-taped role models improves female attitudes toward computer science. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education), 28*(1), 42–45. https://doi.org/10.1145/236462.236491

Tsan, J., Boyer, K. E., & Lynch, C. F. (2016). How early does the CS gender gap emerge? A study of collaborative problem solving in 5th grade computer science. *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 388–393. https://doi.org/10.1145/2839509.2844605

Vickrey, T., Rosploch, K., Rahmanian, R., Pilarz, M., & Stains, M. (2015). Research-based implementation of peer instruction: A literature review. *CBE Life Sciences Education, 14*(1), 1–11. https://doi.org/10.1187/cbe.14-11-0198

Watkins, J., & Mazur, E. (2013). Retaining Students in Science, Technology, Engineering, and Mathematics (STEM) Majors. *Journal of College Science Teaching, 42*(5), 36–41. https://www.jstor.org/stable/43631580?seq=1

Werner, L. L., McDowell, C., & Hanks, B. (2004). Pair-Programming Helps Female Computer Science Students. *ACM Journal on Educational Resources in Computing, 4*(1), 4. https://doi.org/10.1145/1060071.1060075

Williams, L., McCrickard, D. S., Layman, L., & Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. *Proceedings - Agile 2008 Conference*, 445–452. https://doi.org/10.1109/Agile.2008.12

Wohlford, K., Lochman, J., & Barry, T. (2004). The Relation Between Chosen Role Models and the Self-Esteem of Men and Women. Journal of Business Ethics, 51(1), 31–39. https://doi.org/10.1023/B:SERS.0000023076.54504.ca

Zaidi, R., Freihofer, I., & Townsend, G. C. (2017). Using scratch and female role models while storytelling improves fifth-grade students' attitudes toward computing. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, 791–792. https://doi.org/10.1145/3017680.3022451

Section 2: Teaching and assessing computing in the curriculum

Section 2: Teaching and assessing computing in the curriculum

Is it a wave? Linking the abstract to the everyday and back again

Jane Waite (Queen Mary University of London, UK)

Abstract

Irrespective of the subject being taught and the pupils who are learning, contexts and vocabulary matter. In this article, I will describe a knowledge-building framework, which reveals the changes, over time, of contexts and vocabulary in learning experiences. I report on recent research conducted with Paul Curzon and Karl Maton, where we used this framework to review and improve computing lessons and illustrate where teachers and resource creators can use semantic wave profiling to do the same.

Introduction

Karl Maton has created a framework which helps educators review how knowledge is built during learning experiences (Maton, 2014; Maton et al., 2016). The framework is called Legitimation Code Theory (LCT) and introduces dimensions which reveal specific aspects of knowledge building. LCT is a sociological framework that primarily builds on the work of Basil Bernstein and specifically his code theory (Bernstein, 1971). Within LCT, several dimensions have been suggested and one of them is semantics. We can use semantics as a basis to review explanations as well as general learning activities. Lessons are reviewed by profiling them as a drawn diagram. Semantic profiling enables us to abstract the process of learning and gain a better understanding of how knowledge is developed over time. This process enables educators to reflect on, and improve, learning experiences for their students. A vibrant community of researchers and practitioners from across the world have started to use the framework to evaluate and improve teaching and learning in subjects as diverse as ballet and law.

Semantic profiling

The semantics dimension of LCT looks at the changes in two aspects of knowledge over time, the change in context and the change in the complexity of the knowledge. Simply put, for context, we look at whether learning is more abstract and context independent compared to learning set in a specific everyday context. For complexity, we look at whether the vocabulary used is highly abstract, technical, and complex or is familiar language. For example, a lesson might start by introducing the term algorithm, with no context given and no use of everyday language. Students then might take part in an activity where they follow everyday instructions to make a jam sandwich, they then review how precise these instructions are and a link is made back to the concept of an algorithm.

An example of a semantic profile

To evaluate a lesson using semantics, we draw a semantic profile. The profile shows the changes in context and vocabulary on a simple graph, with the time passing along the x-axis. Different shapes of profiles suggest different learning

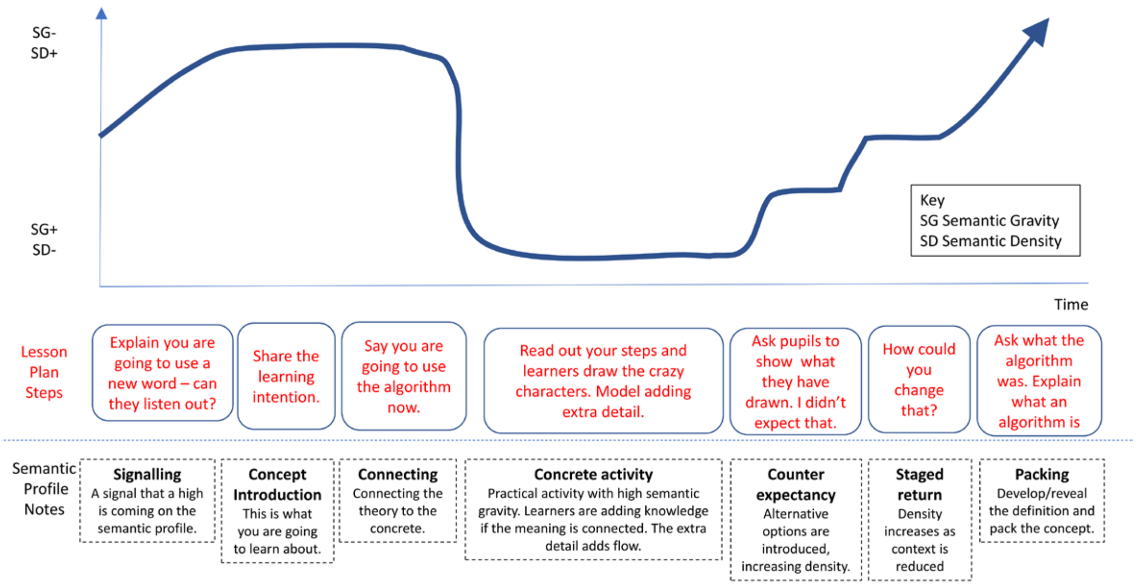

SG-
SD+
SG+
SD-
Key
SG Semantic Gravity
SD Semantic Density
Time
Lesson Plan Steps
Explain you are going to use a new word – can they listen out?
Share the learning intention.
Say you are going to use the algorithm now.
Read out your steps and learners draw the crazy characters. Model adding extra detail.
Ask pupils to show what they have drawn. I didn't expect that.
How could you change that?
Ask what the algorithm was. Explain what an algorithm is
Semantic Profile Notes
Signalling A signal that a high is coming on the semantic profile.
Concept Introduction This is what you are going to learn about.
Connecting Connecting the theory to the concrete.
Concrete activity Practical activity with high semantic gravity. Learners are adding knowledge if the meaning is connected. The extra detail adds flow.
Counter expectancy Alternative options are introduced, increasing density.
Staged return Density increases as context is reduced
Packing Develop/reveal the definition and pack the concept.

Figure 1. Crazy Characters semantic profile (Waite, Maton, Curzon, & Tuttiett, 2019)

experiences, and we can then compare these shapes when planning or reviewing learning.

An example semantic profile is shown in Figure 1. This is the profile of the introduction of a popular key stage 1 (pupils aged from 5 to 11 years old) lesson plan. The lesson, Crazy Characters, is one of the early Barefoot computing activities, written in 2017 to help demystify the term algorithm.

In the profile, we can see how knowledge is framed in an abstract context, using more technical language at the start of the teaching segment. During the practical part of the lesson, the knowledge is situated in a specific context: drawing a character using everyday familiar vocabulary.

After the practical activity, learners are required to repack their experience to a context-independent view of their newly developed knowledge using more technical vocabulary. This profile is a wave shape, similar to Figure 2, and is called a semantic wave.

Not all teaching experiences follow the same pattern, some can be a flat line, such as an activity which is only situated in a specific practical context, with no technical language, called a low flat line, see Figure 3. In creating the Crazy Character activity, the activity could have been designed where pupils followed instructions to draw a made-up persona with no reference to algorithms, no building of knowledge related to this core concept, in which the profile would have been a low flat line.

Another profile type is a high flat line, in which the learning is all theory and complex vocabulary, see Figure 4. In creating the Crazy Character activity, the activity could have been designed where the teacher explained what an algorithm is with no everyday examples nor any everyday words or phrases.

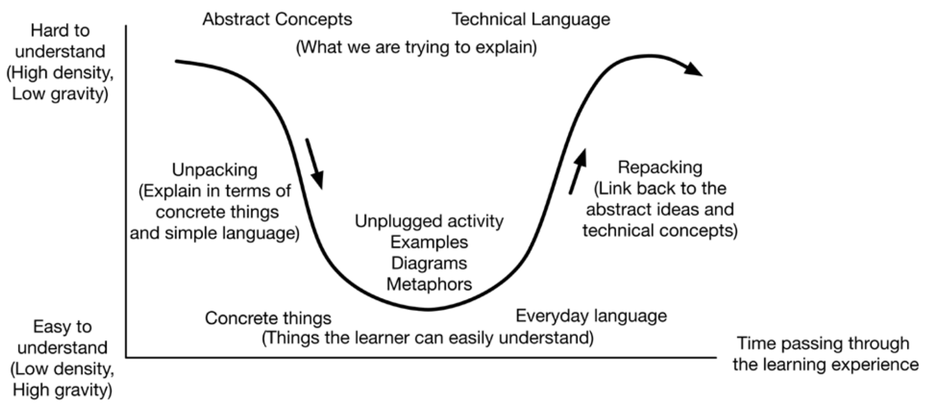

Hard to understand (High density, Low gravity)
Abstract Concepts
Technical Language
(What we are trying to explain)
Unpacking (Explain in terms of concrete things and simple language)
Unplugged activity
Examples
Diagrams
Metaphors
Repacking (Link back to the abstract ideas and technical concepts)
Easy to understand (Low density, High gravity)
Concrete things
Everyday language
(Things the learner can easily understand)
Time passing through the learning experience

Figure 2. A semantic wave (Curzon, 2019)

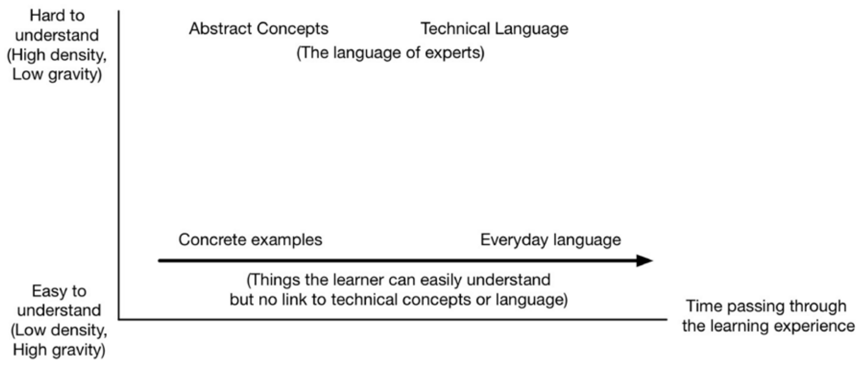

Hard to understand (High density, Low gravity)
Abstract Concepts
Technical Language
(The language of experts)
Concrete examples
Everyday language
(Things the learner can easily understand but no link to technical concepts or language)
Easy to understand (Low density, High gravity)
Time passing through the learning experience

Figure 3. Low flat line (Curzon, 2019)

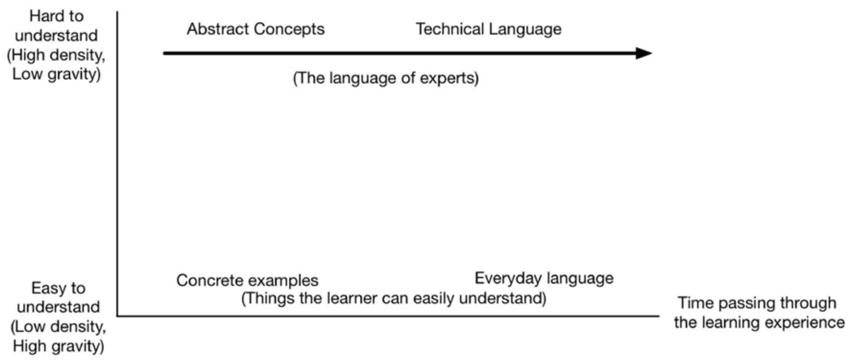

Hard to understand (High density, Low gravity)
Abstract Concepts
Technical Language
(The language of experts)
Concrete examples
Everyday language
(Things the learner can easily understand)
Easy to understand (Low density, High gravity)
Time passing through the learning experience

Figure 4. High flat line (Curzon, 2019)

How to use semantic profiling

Research on profiles indicates that experiences which incorporate waves help learners build mastery and a depth of understanding (Maton, 2020). These moves from context independence to context dependence and vice versa or from everyday, less dense vocabulary to technical, complex, rich vocabulary, are essential to knowledge building.

As well as using semantic profiling to see the shape of an existing lesson, we can use profiling to improve a lesson. To do this, we have trialled the use of three key questions that help teachers think about the vocabulary and context used and change their teaching to improve the wave.

The three key questions are:

- Is the profile a wave?
- Who unpacks and repacks?
- How high or low does the wave go? (Curzon et al., 2020)

The first question of "Is the profile a wave?" leads us to think about adding in steps that get the learner to use more or less technical language or introduce a more or less specific context. For example, in an activity that is a flat line, say because it is all theory, then add a practical everyday context using everyday language to explain it. If on the other hand, the activity has no theory, then add learning steps in which learners pay attention to the overarching concepts, before and after the practical work.
The second question, "Who unpacks and repacks?", asks us to think whether learners are engaged in linking their learning from the abstract to the concrete and vice versa or whether we, the teachers, do it for them. It is easy to recall lessons when at the end of the main teaching activity, we state what the activity has taught the students, linking the activity to the concepts for our learners. Whether our students, at that point, made the connection themselves is difficult to say. But if we add a distinct task that gets our learners to make their links, we can be sure that everyone follows the wave in the class. For example, for unpacking, we could ask learners to jot down everything they know about a concept, such as an algorithm, and to include an example that a novice would understand. For repacking, we could ask learners to share with a partner, in their own words, a general definition of a concept. As a point to note, in making student knowledge visible, there are opportunities for formative assessment during unpacking and repacking.

The third question of "How high or low does the wave go?" draws our attention to the depth of learning. To go lower, we could introduce role play or tangible objects. To go higher, we might request that an abstract view of a concept is provided that is context-free, such as a generalised definition. For example, when explaining variables, we could use an analogy such as a variable is like a box with a photocopier and shredder and explain this verbally. We could take the wave lower, drawing pictures of our analogy, or we could take it lower still by students acting out the analogy, physically with boxes, copying data, ripping up paper. There are similarities here with the ideas in maths of representing concepts in abstract, iconic (drawings) and physical forms.

Conclusion

In our research (Curzon et al., 2020) we have found that asking these three questions helps us redesign learning activities, and we have initial evidence that these new lessons are improved for learners. However, we need more evidence to be sure and are planning more research on the impact of making changes, based on these questions, to the profile of lessons.

References

Bernstein, B. (1971). Class, codes and control, volume I: Theoretical studies towards a sociology of language. London: Routledge & Kegan Paul

Cuzon, P. (2019) Semantic Waves, TIP 9 : FOLLOW SEMANTIC WAVES Blog post, Available online https://teachinglondoncomputing.org/semantic-waves/semantic-waves-tip9/

Curzon, P., Waite, J., Maton, K. & Donohue. J. (2020). Using semantic waves to analyse the effectiveness of unplugged computing activities. In Proceedings of the 15th Workshop on Primary and Secondary Computing Education (WiPSCE '20). Association for Computing Machinery, New York, NY, USA, Article 18, 1–10. DOI : https://doi.org/10.1145/3421590.3421606

Maton, K. (2014). Knowledge and Knowers: Towards a realist sociology of education. Routledge, Milton Park, Abingdon, Oxon.

Maton, K. (Ed.), Hood, S. (Ed.), Shay, S. (Ed.). (2016). Knowledge-building. London: Routledge, https://doi.org/10.4324/9781315672342

Maton, K. (2020). Semantic waves: Context, complexity and academic discourse. In Accessing Academic Discourse: Systemic functional linguistics and Legitimation Code Theory , J. R. Martin, K. Maton, and Y. J Doran (Eds.). Number 59-85. Routledge, London, Chapter 3. https://doi.org/10.4324/9780429280726

Waite, J., Maton, K., Curzon, P. & Tuttiett, L. (2019). Unplugged Computing and Semantic Waves: Analysing Crazy Characters. In Proceedings of the 1st UK & Ireland Computing Education Research Conference (Canterbury, United Kingdom) (UKICER) . Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3351287.3351291

Appendix: Further Reading

1. Pedagogy Quick Read on semantic waves
2. Teaching London computing pages
3. Legitimate Code Theory website

Section 2: Teaching and assessing computing in the curriculum

BCTt: Beginners Computational Thinking Test

María Zapata Cáceres and Estefanía Martín-Barroso (Universidad Rey Juan Carlos, Spain), and Marcos Román-González (Universidad Nacional de Educación a Distancia, Spain)

Abstract

Computational thinking (CT) is a cognitive ability that is considered one of the core skills to be developed in order to successfully adapt to the future. Therefore, it is being included in school curricula all over the world and, gradually, at an earlier age. However, as the incorporation of CT learning in schools is recent, there is still no consensus on its exact definition or on how it should be assessed. Recent research suggests that systems of assessments should be used for this purpose, using various instruments, and thus covering the different CT dimensions. However, there is a lack of validated instruments for the assessment of CT, particularly for early ages. Taking as a reference a three-dimensional CT framework, based on a validated CT test, and aimed at early ages (five- to ten-year-old students), the Beginners Computational Thinking Test has been developed as a tool to be used within a system of assessments. This instrument has been designed, submitted to a content validation process through an expert judgement procedure, and administered to primary school students, obtaining very favourable results in terms of its reliability.

Introduction

In a changing society, where technology and programming play a crucial role, students must be able to think critically and solve complex problems to adapt to the world in which they are expected to live in. Therefore, computational thinking (CT) is a core skill, necessary to adapt to the future and, consequently, it is an important area of education in many countries, some of them consider CT as a national program (Hsu, Chang, & Hung, 2018). In the early stages," in addition to reading, writing, and arithmetic, computational thinking should be added to every child's analytical ability" (Wing, 2006, p.33). However, there is still not enough research on how to teach and assess CT when it comes to young children (Rich et al., 2018).

Although CT is considered an essential skill and it should be learned at school, there is a lack of consensus in its definition and possible breakdown (Shute, Sun, & Asbell-Clarke, 2017). Abstraction, decomposition, algorithms, and debugging are the four CT components that arise most often in the literature and, furthermore, Shute et al. (2017) identify iteration and generalisation as two more components to add to CT skills. Brennan and Resnick (2012), developed the three-dimensional (3D) framework of CT which divides CT into three computational dimensions: (a) computational concepts, (b)

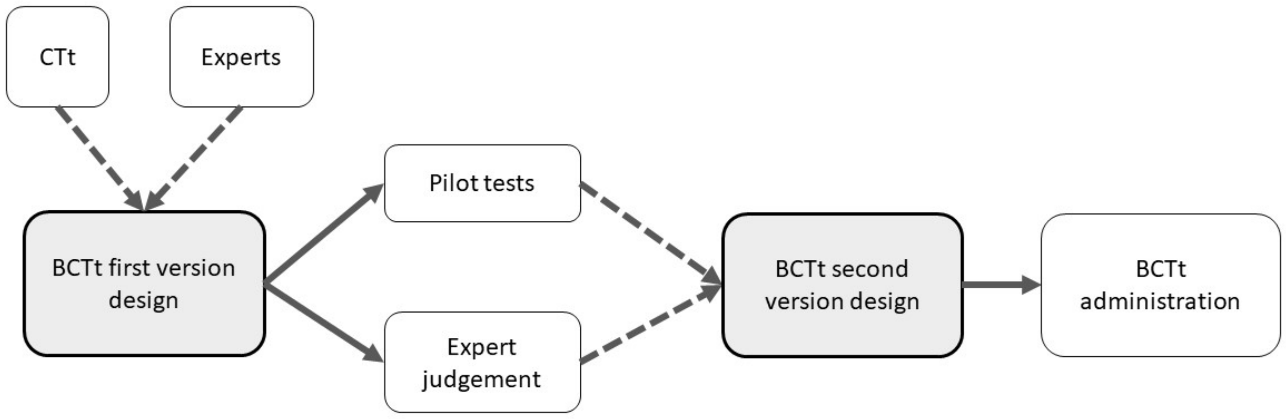


Figure 1. Design and validation steps

computational practices, and (c) computational perspectives (Brennan, Resnick, & MIT Media Lab, 2012).

With regard to the assessment of CT, Shuchi Grover proposes a system of assessments (Grover & Pea, 2015) that combines instruments of different types such as portfolio, survey, interview, and traditional test, and are thus able to cover all the CT dimensions. However, although efforts have been made in the last two decades in the development of assessment instruments, most of them are aimed at middle/high school students and are based on the analysis of programming projects carried out by students in specific environments such as Dr. Scratch that assess CT skills through the analysis of Scratch projects (Moreno-León & Robles, 2015). There are hardly any traditional tests that are independent of a programming environment and that can be used as pre-test and post-test instruments. Among them, the Computational Thinking Test (Román-González, Pérez-González, & Jiménez-Fernández, 2017) stands out, as it is designed under a psychometric approach and provides evidence about its reliability and content validity (Román González, 2015), but it is aimed at students between 10 and 16 years old. Based on the CTt, the Beginners Computational Thinking Test (BCTt) has been developed, aimed at younger students and, therefore, has required a deep adaptation in both form and content. Moreover, substantial improvements were added. A validation process was then carried out, including the administration of the test to students from 5 to 12 years of age from three different schools in Spain.

Method

A first version of the test was designed, aimed at students from primary school stage. Based on the CTt, the test was adapted both in form and content to younger students, thus, several substantial changes were made. Then, the BCTt was pilot tested on small subsamples, and evaluated by experts in the field. Based on these preliminary results, changes were made to obtain a second version that was administered to students from 5 to 12 years old in three schools in Spain. The results obtained were analysed statistically. In Figure 1, the validation process is shown.

Test questions	Computational concepts in BCTt					
		Loops		Conditionals		
	1. Sequences	2. Simple loop	3. Nested loop	4. If-then	5. If-then-else	6. While
1 - 6	■					
7 - 11		■				
12 - 18			■			
19 - 20				■		
21 - 22					■	
23 - 25						■

Table 1. BCTt computational concepts considered in each question

BCTt first version design

The test is aimed at young children who may not yet have acquired reading and writing skills, so the test was designed to include symbols and drawings that were self-explanatory. Aimed at older students, minimal helpful texts were included that redound to the symbol-based explanations. The symbolism is clear and aims to connect emotionally with the children so the learning process is enhanced (Căprioară, 2017) as it is a chick that must reach its mother.

The first version of the BCTt takes approximately 40 minutes to complete. It is divided into six sets of questions, and each set is related to one basic computational concept (Table 1). It is 25 questions long with three alternative responses per question.

There are two types of questions: canvas type — which is a "follow the line" design — and maze type or square matrix — in which there is

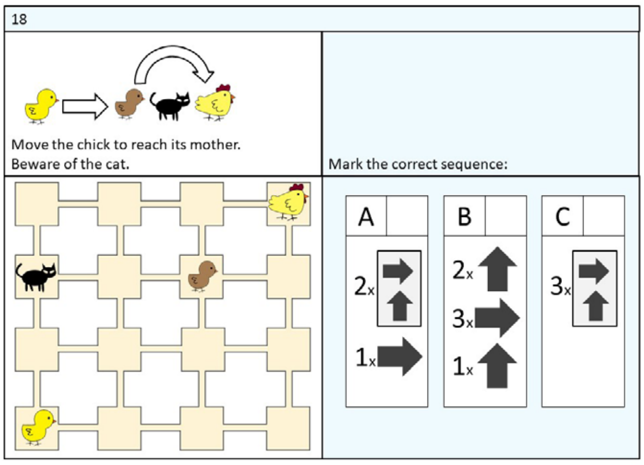

18
Move the chick to reach its mother.
Beware of the cat.
Mark the correct sequence:
A
2x
1x
B
2x
3x
1x
C
3x

Figure 2. BCTt first version question example (question number 18)

a starting square and the students must reach the goal square, avoiding and/or picking up objects along the way, for example, another chick (Figure 2). The possible answers are sequences of movements represented by arrows, symbols, and numbers. Visual transitions were added

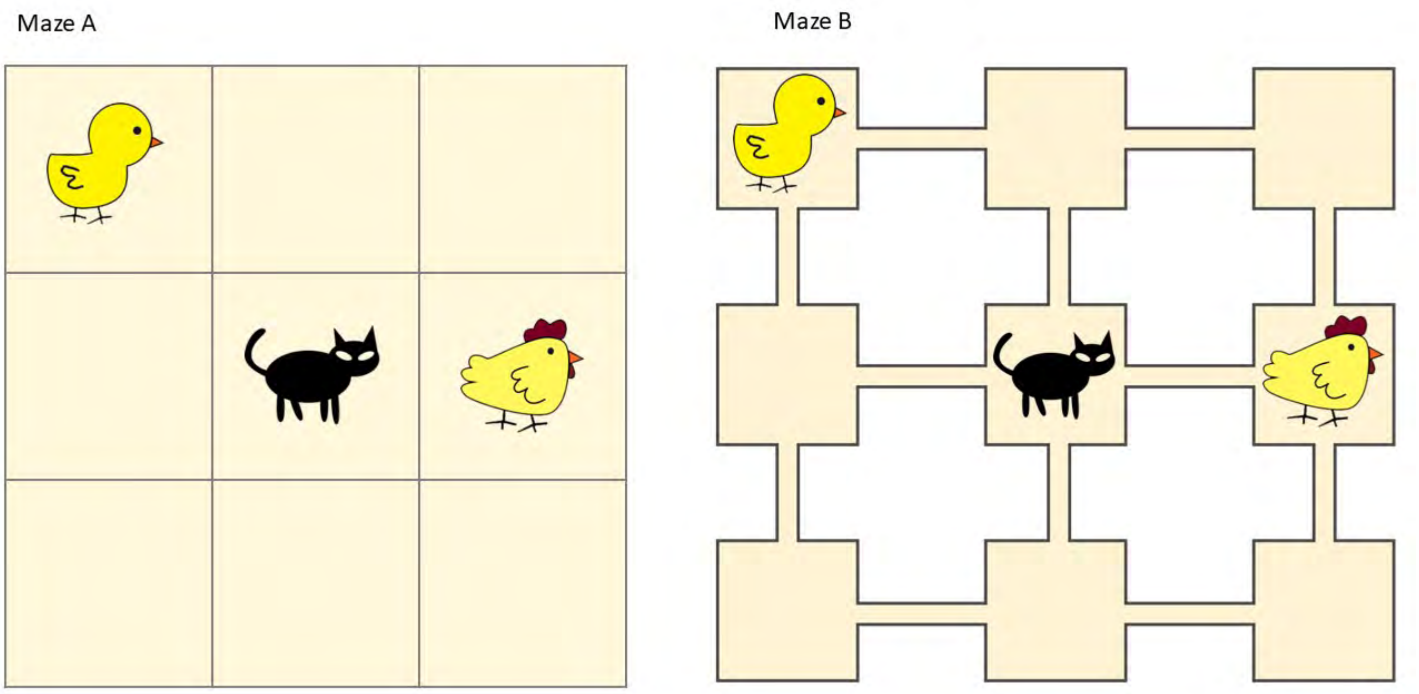


Figure 3. Maze A: no transitions; Maze B: transitions are added between squares turning the maze into a state diagram

in the maze layout (Figure 3), as a substantial improvement for young students, to help the visualisation of the step between one square and another, so that the maze becomes a state diagram which is a main programming element and it is proven to improve understanding of problems (Chen & Herbst, 2013; Durak & Saritepeci, 2018; Watanabe, 2015)

Expert judgement procedure

The BCTt was submitted to an expert judgement procedure in which 45 experts (computer science professionals and schoolteachers) were consulted.

The experts were asked, via an online form consisting of 66 questions, about the adequacy of the questions in terms of its relevance to measure each of the different computational concepts included in the test. In addition, the experts were also asked several questions regarding length, symbolism used, and other issues. A special survey was also conducted on the transitions included in the maze-type questions. Finally, experts' comments and suggestions were collected.

Most of the experts considered the length of the test to be adequate, as well as the order of the questions, as their difficulty was perceived to be incremental throughout the test. The experts also considered that the test questions measured the computational concepts addressed and considered the nested loops as the most relevant concept of all (relevance to measure CT, Likert scale from 1 to 5: Sequences=3,66; Simple loops=3,82; Nested loops=4,14; If-then=3,95; If-then-else=4,15; While=4,05). As to whether the test as a whole assesses the computational

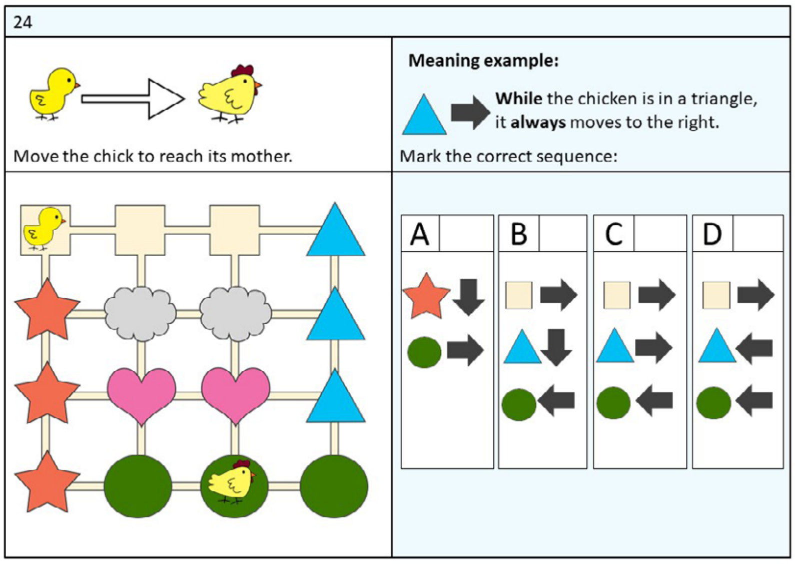

24
Move the chick to reach its mother.
Meaning example:
While the chicken is in a triangle, it always moves to the right.
Mark the correct sequence:
A
B
C
D

Figure 4. BCTt question number 24

concepts dimension of computational thinking in primary school students, 75.6% of the experts considered that completely or almost completely.

One of the substantial improvements of the BCTt was the inclusion of transitions in the maze questions. In this sense, the experts were very much in agreement with the improvement and considered that it would help younger students to better understand the answers in the test since, for example, the association between the arrows in the answers and a possible movement is easier, diagonal movements through the maze would be avoided or it is reflected more clearly when the chick has reached the goal square.

The comments and suggestions of the experts were very much considered. For example, some of their most frequent comments were about the need to first explain orally to the children the meaning of each set of questions, to add one more possible answer to each question, to replace the pickable elements (chicks) with other symbol that do not lead to confusion, to refine the questions concerning some computational concepts to get closer to their exact definition, etc.

Many of the experts considered the test too hard for such young children. This was refuted later when the test was administered to primary school students.

BCTt second version design

Taking into account this feedback, the second version of the BCTt was designed with several modifications and additions. For example, one more answer alternative was added to each question, the statements of the questions were refined, the collectable elements were replaced by others, and the questions of some of the sets were reformulated to come closer to the computational concept formal definition.
One of the most remarkable changes was the reinforcement of the colours in the questions with a shapes symbolism, to allow students who are colour-blind to be able to take the test. This improvement makes the test suitable to be printed in black and white format (e.g. Figure 4).

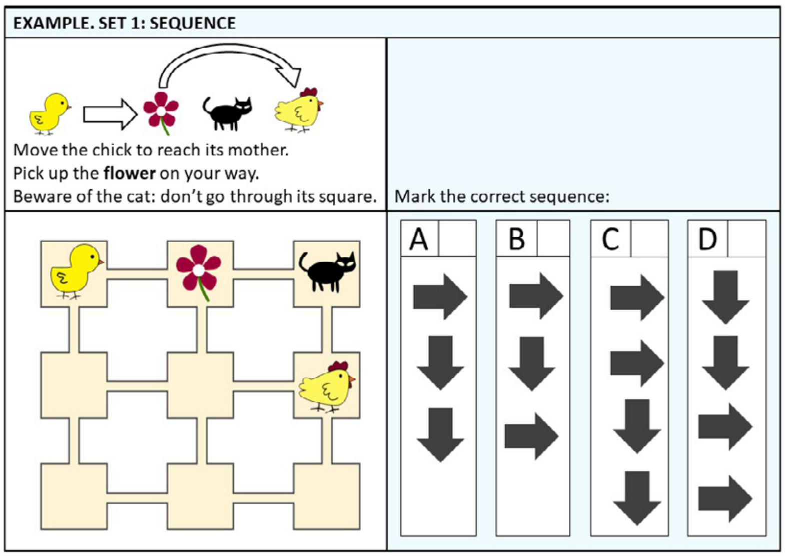

EXAMPLE. SET 1: SEQUENCE
Move the chick to reach its mother.
Pick up the **flower** on your way.
Beware of the cat: don't go through its square.
Mark the correct sequence:
A
B
C
D

Figure 5. BCTt action protocol, example for set 1: sequences

In addition, an administration protocol was developed specifying that an oral explanation must be given to students prior to taking the test, with an explanatory example of each of the computational concepts addressed in the test. The protocol includes these examples and a guide on how to carry out the explanation (e.g. Figure 5).

Educational stage	Grade	Identifier	BCTt variation 1	BCTt variation 2
1st	1	A	A1: n=52	
	2	B	B1: n=18	B2: n=18
2nd	4	C	C1: n=54	
	4	D	D1: n=28	D2: n=28
3rd	5	E	E1: n=51	
	6	F	F1: n=25	F2: n=25

Table 2. BCTt administration subsamples (n: number of students)

Grad e	Subsample	BCTt variation	N	Mean	Std. Deviation	Levene's Test	t-test for Equality of Means	
						Sig.	t	Sig. (2-tailed)
2	B1	1	18	16.778	2.487	0.841	3.042	**0.005**
	B2	2	18	14.278	2.445			
4	D1	1	28	21.357	2.438	0.113	0.122	**0.904**
	D2	2	28	21.286	1.922			
6	F1	1	25	21.720	2.622	0.185	0.499	**0.620**
	F2	2	25	21.280	3.542			

Table 3. Subsamples statistics and student t-test comparing BCTt variations (1: with transitions and 2: without transitions)

BCTt administration

The second version of the BCTt was administered to 5- to 12-year-old primary school students (n=299), following the action protocol, from three schools in Spain.
Two different variations of the second version of the BCTt were carried out, one including the transitions between the squares (variation 1), and the other without them (variation 2), in order to be able to compare the performance of the students with and without this aid. All tests were printed in black and white, so students who are colour-blind could take the test under the same conditions as the rest.

The sample of students was divided into several subsamples as shown in Table 2, considering the

	Subsample	A1	B1	C1	E1	F1
	Grade	1	2	4	5	6
	N	52	18	54	51	25
	Mean	**16.52**	**16.78**	**21.57**	**21.84**	**21.72**
	Median	16.00	18.00	23.00	23.00	22.00
	Std. Deviation	3.31	2.49	3.04	2.61	2.62
	Variance	10.96	6.18	9.27	6.81	6.88
	Minimum	8.00	11.00	14.00	13.00	15.00
	Maximum	24.00	20.00	25.00	25.00	25.00
Percentiles	25	14.00	15.75	19.00	20.00	19.50
	50	16.00	18.00	23.00	23.00	22.00
	75	19.00	18.00	24.00	24.00	24.00

Table 4. BCTt score statistics by grade

age of the students, the variation of the test they would take, and the school group they belonged to. One of the subsamples (D1) was retested a second time five weeks later.

Results and discussion

The main results of the BCTt validation process are presented below. The complete results are detailed in the paper presented at the EDUCON congress (Zapata-Cáceres, Martín-Barroso, & Román-González, 2020).

Transitions between squares in maze questions
The transitions between squares in the maze-type questions were intended to be a substantial improvement in the BCTt. To check this, test scores were compared between the BCTt variation 1 (with transitions) and BCTt variation 2 (without transitions). The results indicate that there is no significant difference in the test scores obtained in the samples of students from the fourth grade onwards, but there is a very significant difference between the scores of children in lower grades (p=0.005<0.01), indicating that this help is highly noticeable for younger children, but not for older ones.

Descriptive statistics

A statistical analysis of the results obtained by all students in the BCTt, considering the total score of each student in the test as the sum of the correct answers (Table 4), shows in the first place that the overall average is high (19.92 points), which contradicts the opinion of the experts that the difficulty of the test is very high and, on the contrary, indicates that the test is too easy for older students, a ceiling effect is observed, and BCTt target could be students in the early stages of primary education.
Analysing the scores obtained in each of the computational concepts sets separately, in the questions dealing with nested loops and conditionals, students obtain low scores at all grades, while sequences and simple loops seem

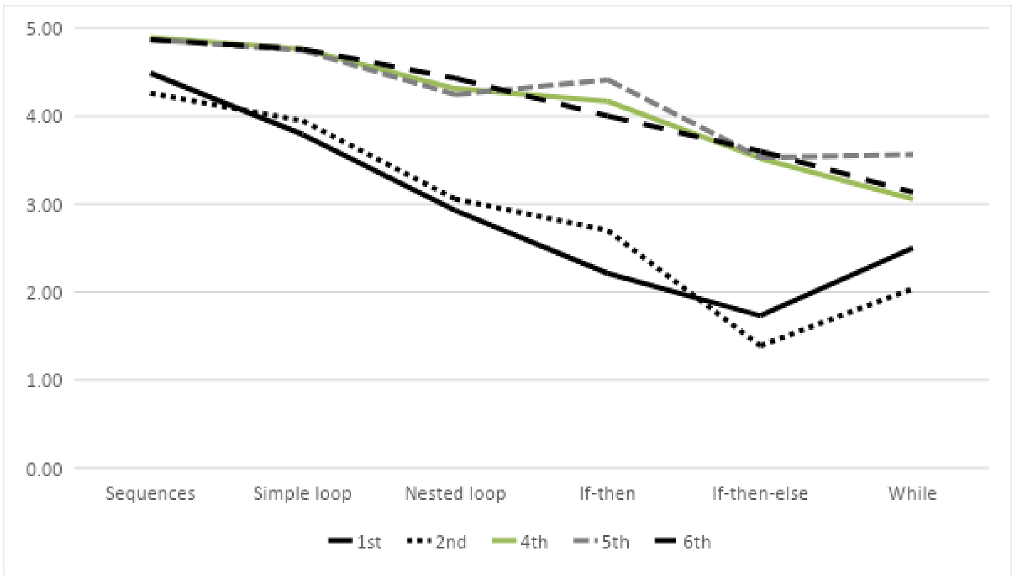


Figure 6. Abscissa axis: computational concept by grade. Ordinate axes: BCTt question score, normalised from 0 to 5: 5 maximum score.

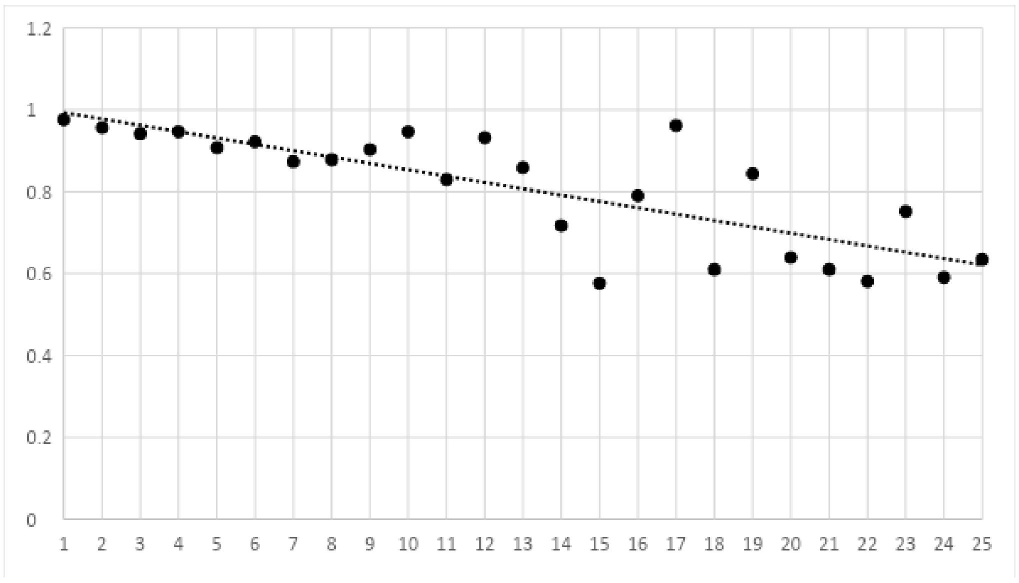


Figure 7. Question difficulty index (ordinate axis) for each BCTt question (abscissas axis).

too simple for high grades (Figure 6).

The difficulty index of each question confirms the increasing difficulty of the test anticipated by the experts, with the average index being very high (0.81) for the overall sample (Figure 7) and medium (0.70) for the first educational stage, in which also is balanced in terms of difficulty as its histogram is symmetric and fits the normal curve.

The histogram showing the distribution of the BCTt score along 1st and 2nd grades subsamples fits the normal curve and it is fairly symmetric, which suggests that the BCTt is balanced in terms of the difficulty of its questions for primary school 1st educational stage (Mean=16,59; Std. Dev.=3,104; N=70).

Reliability

The BCTt showed a very good reliability considering all grades (Cronbach's Alpha: α=0.824), but when considering each educational stage separately, Cronbach's Alpha is lower the higher the grade (1st grade: α=0.833; 2nd grade: α=0.793; 4th grade: α=0.771; 5th grade: α=0.660; 6th grade: α=0.657). Therefore, the BCTt is more reliable in the early stages of primary education. In addition, Spearman's non-parametric test was used in a task and re-task method on the D1 sample (the BCTt test was administered twice under identical conditions with five weeks lapse) and showed a very strong positive correlation (rs=0.93; p<0.01). Therefore, the reliability as stability was very high.

Conclusions

The expert judgement procedure showed that the BCTt was adequate, both in design and content, being a balanced and incremental test in terms of difficulty. Furthermore, the concepts to be assessed seemed relevant in terms of the evaluation of computational thinking in its computational concepts dimension. The results of the administration of the test to primary education students confirm this, although it can be concluded that the test is aimed at students in the first grades (five to ten years old), as the first part of the test might be too easy for older students.

The test is balanced in difficulty, and in terms of reliability has proved to be very high, especially, again, for the early stages of primary education. The transitions added in the maze-type questions proved to be very significant as a positive aid for students in the first grades and had no effect (either positive or negative) on older students. Therefore, it is recommended to include transitions in this type of test questions in the future.

The BCTt has proven to be an instrument aimed at the early stages of primary education (five to ten years old), as an extension of the CTt (10 to 16 years old), independent of any environment, it focuses on 3D framework computational concepts, partially on computational practices, and ignores computational perspectives. It is recommended as a pre-test and post-test tool to be used within a "system of assessments" together with other instruments that assess other dimensions of CT (Román-González, Moreno-León, & Robles, 2019).

The BCTt in its second version is considered a good start for successive versions and improvements. As the first questions have proved to be too easy for high grades, and a ceiling effect has been observed, further adaptations of the test are currently being made for these groups and more difficult questions are being included. On the other hand, the lower age limit for taking the test has not been described and efforts are also currently being made in this regard. In addition, several translations and administrations of the BCTt are being carried out with other populations and countries.

Acknowledgments

Thanks to the participants, teachers, and schools involved in the learning experiences.
This work is partially supported by Comunidad de Madrid. Research grant e-Madrid-CM (ref. P2018/TCS-4307).

References

Brennan, K., Resnick, M., & MIT Media Lab. (2012). New frameworks for studying and assessing the development of computational thinking. *American Educational Research Association Meeting, Vancouver, BC, Canada*.

Căprioară, D. (2017). Emotions and the learning of school mathematics. *Bulletin of the Transilvania University of Braşov, Series VII: Social Sciences and Law, 10*(1), 9-18. Retrieved from Publicly Available Content Database database. Retrieved from http://www.ceeol.com/search/article-detail?id=566146

Chen, C., & Herbst, P. (2013). The interplay among gestures, discourse, and diagrams in students' geometrical reasoning. *Educational Studies in Mathematics*, 83(2), 285-307. doi:10.1007/s10649-012-9454-2

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

Grover, S., & Pea, R. (2015). " *Systems of assessments" for deeper learning of computational thinking in K-12*

Hsu, T., Chang, S., & Hung, Y. (2018). *How to learn and how to teach computational thinking: Suggestions based on a review of the literature* doi:// doi.org/10.1016/j.compedu.2018.07.004

Moreno-León, J., & Robles, G. (2015). *Dr. scratch: A web tool to automatically evaluate scratch projects* doi:10.1145/2818314.2818338

Rich, P. J., Browning, S. F., Perkins, M., Shoop, T., Yoshikawa, E., & Belikov, O. M. (2018). *Coding in K-8: International trends in teaching elementary/primary computing*. Washington: Springer Science & Business Media.

Román González, M. Computational thinking test: Design guidelines and content validation. *EDULEARN15* at: Barcelona.

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. In S. Kong, & H. Abelson (Eds.), *Computational thinking education* (pp. 79-98). Singapore: Springer Singapore. doi:10.1007/978-981-13-6528-7_6

Román-González, M., Pérez-González, J., & Jiménez-Fernández, C. (2017). *Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test*. doi:// doi.org/10.1016/j.chb.2016.08.047

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

Watanabe, T. (2015). Visual reasoning tools in action: Double number lines, area models, and other diagrams power up students' ability to solve and make sense of various problems. *Mathematics Teaching in the Middle School*, 21(3), 152.

Wing, J. M. (2006). Computational thinking test. *CACM Viewpoint*, 33-35. Retrieved from http://www.cs.cmu.edu/~wing/

Zapata-Cáceres, M., Martín-Barroso, E., & Román-González, M. (2020). *Computational thinking test for beginners: Design and content validation*, IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, pp. 1905-1914, https://ieeexplore.ieee.org/document/9125368

Section 2: Teaching and assessing computing in the curriculum

A framework for formative assessment and feedback to support student learning in CS classrooms

Shuchi Grover (Looking Glass Ventures, USA)

Abstract

This is a conceptual paper that makes a case for making formative assessment an integral part of computing in classrooms and formative assessment literacy a key part of computer science (CS) teacher training and preparation. The paper distils key ideas of formative assessment from education research that are key to understanding the what and why of this crucial classroom practice that can help improve teaching and learning in computer science classrooms. Drawing on prior research in CS education on assessment (albeit summative assessment, mainly) and programming comprehension, as well as ongoing research led by the author, the paper also presents dimensions of a preliminary framework that can help guide the adoption of formative assessment in K-12 CS and progress on three key aspects of formative assessment in K-12 CS: formative assessment design, formative assessment literacy in teacher professional development (PD), and leveraging community and community-developed resources for formative assessment.

Introduction

Over the last half decade, teaching of computer science has become widespread in school settings with curriculum, tools, teacher preparation, and classroom implementation concurrently making steady progress. Programming is central to K-12 CS experiences. Research literature on student difficulties in learning programming, even in easy-to-use block-based environments, continues to grow, as is the large body of literature on addressing novice programmer misconceptions that transcend age, context, and even programming environments. Formative (or classroom) assessment — aimed at assessment for learning, and often targeting student misconceptions — is a critical omission from K-12 CS education discourse and practice, especially in the US. Several studies have identified huge gaps in formative assessment and assessment literacy for K-12 CS teachers (e.g. Vivian & Faulkner, 2018; Vivian et al., 2020; Yadav et al., 2015), even when evidence in research asserts that attention to classroom formative "assessment can produce greater gains in student achievement than any other change in what teachers do" (Wiliam & Leahy, 2012).

Even though formative assessment has been a topic of scholarship for a long time since the 1960s (see Bloom, 1969), it was Paul Black and Dylan Wiliam's seminal research in 1998 that crystallised the importance of formative assessment to improve learning and launched a very active field of educational research. Black & Wiliam (1998) defined formative assessment as "all those activities undertaken by teachers, and/

or by their students, which provide information to be used as feedback to modify the teaching and learning activities in which they are engaged." About ten years later, they established formative assessment as an explicit domain of assessment practice and defined it as follows, "Practice in a classroom is formative to the extent that evidence about student achievement is elicited, interpreted, and used by teachers, learners, or their peers, to make decisions about the next steps in instruction that are likely to be better, or better founded, than the decisions they would have taken in the absence of the evidence that was elicited" (Black & Wiliam, 2009, p. 9).

This paper addresses the need and rationale to push for more deliberate use of well-designed formative assessment in CS classrooms as well as formative assessment literacy for CS teachers. It first offers ten principles of formative assessment distilled from education research. These principles clarify what formative assessment is, and more importantly, is not, and serve to provide the rationale for why we need formative assessment. It then presents a preliminary framework to guide the K-12 CS education community. The framework highlights three key aspects or dimensions that need attention to bring formative assessments to classrooms — design of assessments, teacher preparation and formative assessment literacy, and community participation and resource repositories.

Ten principles of formative assessment

Seminal papers and groundbreaking research in education around three decades ago argued for and demonstrated that formative assessment improves student learning (Black & Wiliam, 1998; Crooks, 1988; Sadler, 1989). Since then, disciplinary-based education research in all core subjects has paid much attention to understanding and implementing formative assessment in classroom teaching to improve student learning. Google Scholar search results on formative assessment in school education run into hundreds of thousands of articles. Only a handful of these are situated in K-12 CS contexts. This section helps build a foundational understanding of formative assessment based on decades of education research through presenting ten principles of formative assessment distilled from education research. These ten principles essentially also describe what formative assessment is, and why it is important.

1. *Formative assessment is assessment* ***for,*** *rather than* ***of****, learning*. Assessment of learning is often referred to as summative assessment. Assessment for learning privileges the learning aspect, whereas assessment of learning privileges the assessment aspect.

2. *Formative assessment is all about feedback*. The raison d'être of formative assessment is to provide evidence and feedback to improve learning. Feedback is a key element in assisting the learning process for both instructors and students (Hattie & Timperley, 2007). In educational research, formative assessment is often not considered complete until it has resulted in feedback and action on the part of the teacher (or teaching agent) and/or the learner. However, feedback in formative assessment is mainly targeted at the student and is most valuable when students have the opportunity to use it to revise their thinking as they are working (Bransford, Brown, & Cocking, 2000). The feedback provided to the learner must impact:
    a. Learner's perception that there may be a gap between goal and where they are at currently, and
    b. What learners do to close the gap

3. *Formative assessment is not a "test"*. It

is NOT aimed to give a student a grade regardless of what pedagogy is being used in the classroom. W.J. Popham famously said "For some teachers, test is a four-letter word, both literally and figuratively" (Popham, 2009, p.5). However, the word assessment has often mistakenly led teachers to assume that formative assessment is about evaluating a student's performance rather than view it as a part of the ongoing teaching and learning process. Teachers also harbor misperceptions around assessment in project-based classrooms. Barron & Darling-Hammond (2008) asserted that "The most effective inquiry-based approaches use a combination of informal ongoing formative assessment and project rubrics that communicate high standards and help teachers make judgments about the multiple dimensions of project work" (p.6). This widespread misperception of formative assessment as 'tests of student learning' is troubling. Heritage (2010) rues that the education community is at the risk of losing the promise of formative assessment for teaching and learning because of the false, but widespread, assumption that formative assessment is a kind of measurement instrument rather than a process that is integral to the practice of teaching and learning.

4. *Formative assessment is a process*. For the teacher, this process involves monitoring *(Is learning taking place?)* to diagnosis (*What is learned / not learned?*) to action (*What to do about it?*). For the learner, the formative assessment process helps them understand — *Where am I going? Where am I now? What are my next steps?*

5. *Formative assessment* is a form of *regulation* — at the classroom level, it helps a teacher regulate the learning process (Hudesman et al., 2013). At the student level, it serves as a way of self-regulation. Many scholars have written about how the external feedback of formative assessment triggers internal processing for the student. Monitoring and external feedback generates internal feedback at a variety of levels such as cognitive, motivational, and behavioral (Nicol & Macfarlane-Dick, 2006). The seminal work on *How People Learn* (Bransford, Brown, & Cocking, 2002) asserts that a formative interaction is one in which an interactive situation influences cognition, i.e. it is an interaction between external stimulus and feedback, and internal production by the individual learner. Crooks (1998) perhaps articulates it best — classroom assessment guides students' judgment of what is important to learn, affects their motivation and self-perceptions of competence, structures their approaches to personal study, and affects the development of enduring learning strategies and skills.

6. *Formative assessment is critical for sharing learning goals with students and what constitutes "good" work*. A key part of formative assessment is to share day-to-day learning goals with students. If improvement in learning is to take place, students need to come to hold a concept of quality in line with that held by the teacher and the community (via standards, for example). This growing concept of what "good work" is forms part of the learning itself (Brookhart, 2003). As students begin to understand their intended learning goals, they develop the skills to make judgments about their learning in relation to a learning standard or instructional outcome and implement a variety of strategies to regulate their learning.

7. *Formative assessment is closely related to teacher pedagogical content knowledge* (PCK; Shulman, 1987). According to Heritage & Wylie (2018), who have done extensive

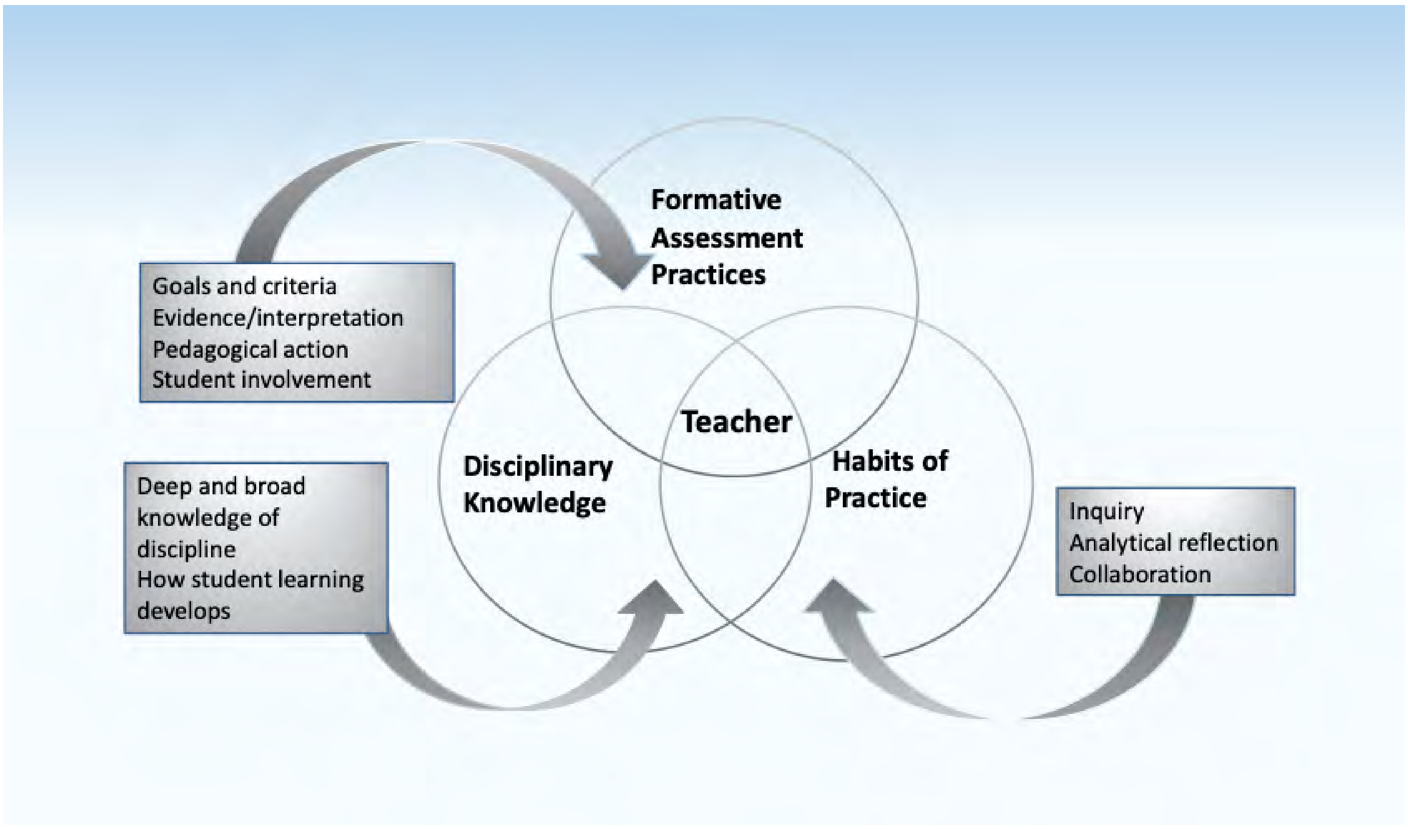


Figure 1. Teacher PCK is intertwined with classroom assessment and habits of practice (Image source: Heritage, 2018)

work on formative assessment in school mathematics teaching, teachers' formative assessment practices are closely intertwined with their disciplinary knowledge and classroom habits of practice (Figure 1).

8. *Formative assessment can and should take many forms*. Formative assessment can range from informal moves such as observation or a show of hands or even informal questions and conversations, to formal assessments that are administered to probe student understanding. Formal assessments can take many forms, including quick "quizzes" (including Entry/Exit Tickets), multiple choice (MC) and fixed answer probes, other innovative question types (such as Parson's Puzzles), open-response types questions (that would need manual grading), programming assignments (with rubrics to guide student work), peer and self-assessment, project showcase, self-explanation and reflection (written or audio-video recorded), and portfolios as well as artifact-based interviews. Examples of each of these are provided in Grover, Powers, & Sedgwick (2020). Ideally, teachers should employ "systems of assessment" that include various forms, target various cognitive, interpersonal, and intrapersonal learning goals of teaching CS, and provide a holistic multi-faceted view of student development

(Grover, 2017). These varied forms of assessment are key for equity and inclusion as well, since different forms of assessment privilege different learners.

9. *Formative assessment needs to be speedy and timely*. This aspect of formative assessment is key. Given that formative assessment is proximal to the learning process, it is key that a teacher and student receive feedback in time to remedy any lack of understanding. Research suggests that teachers' day-to-day classroom practices with an explicit focus on short-cycle assessment have been found to be most impactful (Wiliam, 2006). When teachers want to quickly survey student thinking, multiple choice items are efficient and have utility in terms of taking little time to ask, collect responses and process them. William & Black (2009) suggest attending to "Moments of Contingency", which are critical points where learning changes direction, depending on the information gleaned from the assessment.

10. *Formative assessment should target known misconceptions*. Research in computer science education over the last four decades has documented several difficulties that novice learners face when they first encounter programming. Several of these difficulties and misconceptions are "sticky" and have been observed in learners of various ages and across various programming languages and environments. "Diagnostic items" that target known misconceptions are ideal candidates for formative assessment to probe whether or not students have understood key concepts (Ciofalo & Wylie, 2006; Wylie & Ciofalo, 2008).

Toward a framework of formative assessment for computing in schools

This section outlines a preliminary framework for successful integration of formative assessment in computer science teaching and learning. This framework has been developed as part of an ongoing research project funded by the National Science Foundation (DRL-1943530) aimed at examining formative assessment design, aggregating and creating a community resource or "assessments hub" (leveraging an online platform and homework system, Edfinity.com[7]), and building teacher awareness and formative assessment literacy. This framework also draws on prior research in formative assessment in broader education research such as work on diagnostic assessments by Ciofalo & Wylie (in the context of mathematics). It also draws on prior research in CS education on assessments, albeit mostly summative assessment (e.g. Clear et al., 2008; González, 2015; Grover, 2017, 2020; Lister, 2005; Lister et al., 2006; Schulte et al., 2010; Tang et al., 2020; Wiebe et al., 2019, among others), formative assessments (Grover, 2017; Grover, Sedgwick, & Powers, 2020); programming comprehension and learning trajectories of programming (e.g. Armoni, 2014; Izu et al., 2019; Schulte et al., 2010); and teacher preparation literature in CS (Vivian & Faulkner, 2018; Vivian et al., 2020) and more broadly (deLuca et al., 2018). It should be noted that research in formative assessment specifically in the context of CS and in K-12 settings is very thin. Among the few that exist, many are in the context of automated tools for assessing student programs (e.g. Basawapatna et al., 2015; Moreno-León et al., 2015; Von Wangenheim et al., 2018). These studies are (a) not generalisable as they are restricted to a specific programming language or environment or programming tasks, (b) provide little guidance on identifying specific areas of difficulty or misconceptions, and (c) may not be completely accurate in truly assessing student understanding (Salac & Franklin, 2020).

How successful formative assessment is in impacting student learning in K-12 computer

[7] http://edfinity.com

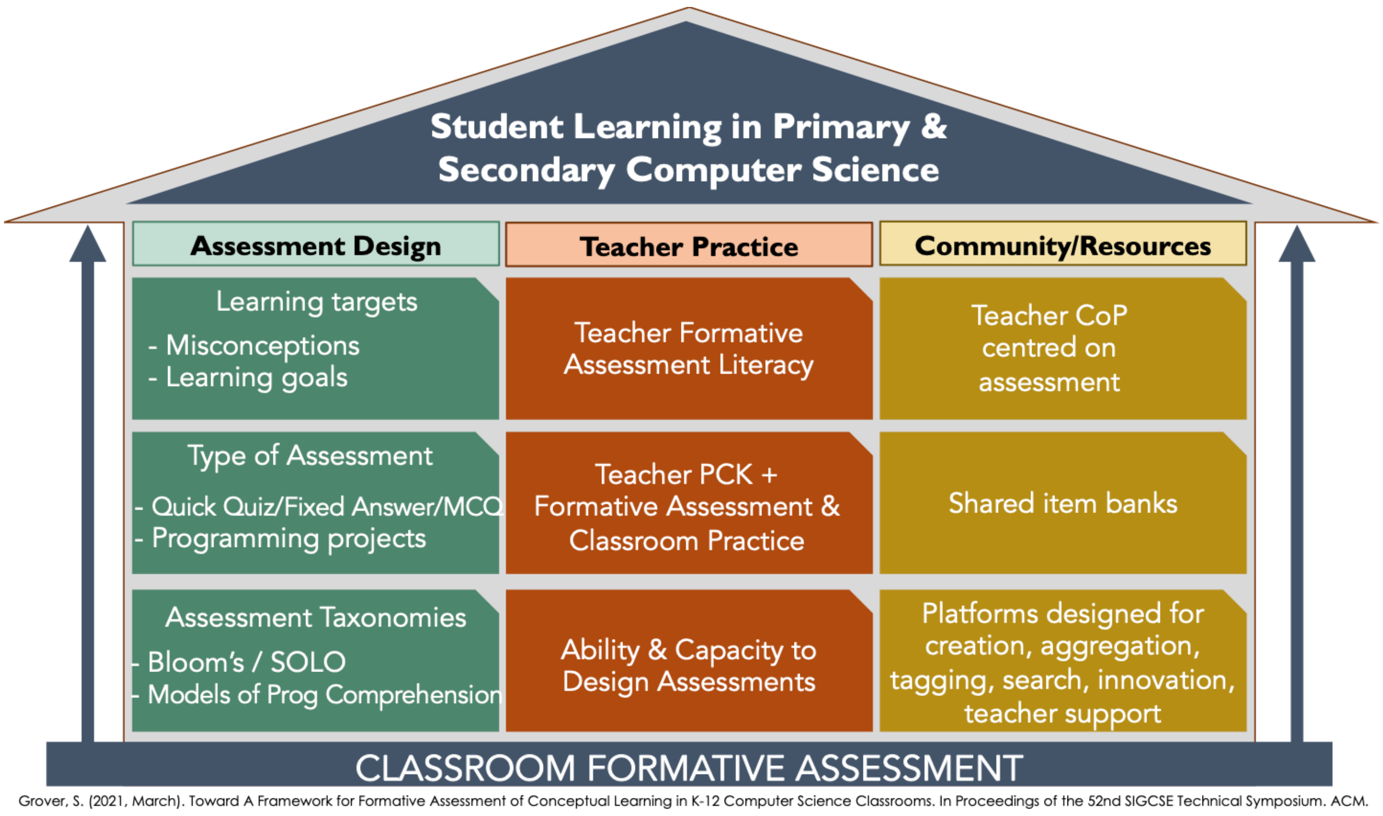
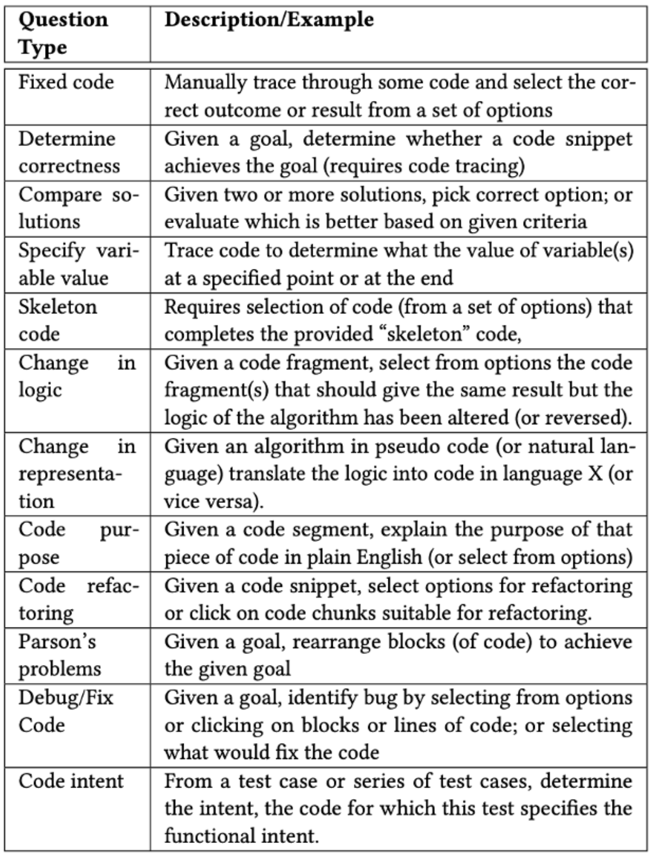

Student Learning in Primary & Secondary Computer Science
Assessment Design
Teacher Practice
Community/Resources
Learning targets
- Misconceptions
- Learning goals
Teacher Formative Assessment Literacy
Teacher CoP centred on assessment
Type of Assessment
- Quick Quiz/Fixed Answer/MCQ
- Programming projects
Teacher PCK + Formative Assessment & Classroom Practice
Shared item banks
Assessment Taxonomies
- Bloom's / SOLO
- Models of Prog Comprehension
Ability & Capacity to Design Assessments
Platforms designed for creation, aggregation, tagging, search, innovation, teacher support
CLASSROOM FORMATIVE ASSESSMENT
Grover, S. (2021, March). Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms. In Proceedings of the 52nd SIGCSE Technical Symposium. ACM.

Figure 2. Dimensions of a framework for formative assessment in CS school classrooms

science depends on three pillars or dimensions – the design of assessments, teacher assessment literacy and their classroom practice, and the broader community (Figure 2). These dimensions are interconnected but work at different levels of the 'computing at schools' enterprise. The remainder of this section describes each of these briefly. Grover (2021) provides more details on each.

Design of assessments for formative feedback

Formative assessments for K-12 CS classrooms could be formal or informal and target conceptual and/or affective learning goals. This effort recognises that programming assignments (in a programming language or environment) are popular as formative tasks, they are time-consuming to score especially on good rubrics that also shed light on exactly what specific concepts a student may or may not have understood (Grover, et al., 2018). This paper currently focuses mainly on formal, designed formative quiz-like check-ins for speedy feedback on conceptual understanding. These are strategic, targeted, autogradable, frequent, low-stakes, and provide quick feedback and explanation. Such items are suitable for probing understanding of key programming concepts (such a sequence, loops, conditionals, functions, expressions, variables, and other data structures) and CT practices (such as debugging, problem decomposition, algorithmic thinking, pattern recognition, and abstraction). These need not, however, always involve a code snippet or programming language. Well-designed multiple



Question Type	Description/Example
Fixed code	Manually trace through some code and select the correct outcome or result from a set of options
Determine correctness	Given a goal, determine whether a code snippet achieves the goal (requires code tracing)
Compare solutions	Given two or more solutions, pick correct option; or evaluate which is better based on given criteria
Specify variable value	Trace code to determine what the value of variable(s) at a specified point or at the end
Skeleton code	Requires selection of code (from a set of options) that completes the provided "skeleton" code,
Change in logic	Given a code fragment, select from options the code fragment(s) that should give the same result but the logic of the algorithm has been altered (or reversed).
Change in representation	Given an algorithm in pseudo code (or natural language) translate the logic into code in language X (or vice versa).
Code purpose	Given a code segment, explain the purpose of that piece of code in plain English (or select from options)
Code refactoring	Given a code snippet, select options for refactoring or click on code chunks suitable for refactoring.
Parson's problems	Given a goal, rearrange blocks (of code) to achieve the given goal
Debug/Fix Code	Given a goal, identify bug by selecting from options or clicking on blocks or lines of code; or selecting what would fix the code
Code intent	From a test case or series of test cases, determine the intent, the code for which this test specifies the functional intent.

Table 1. Item types for programming (Grover, 2021; Schulte et al., 2010)

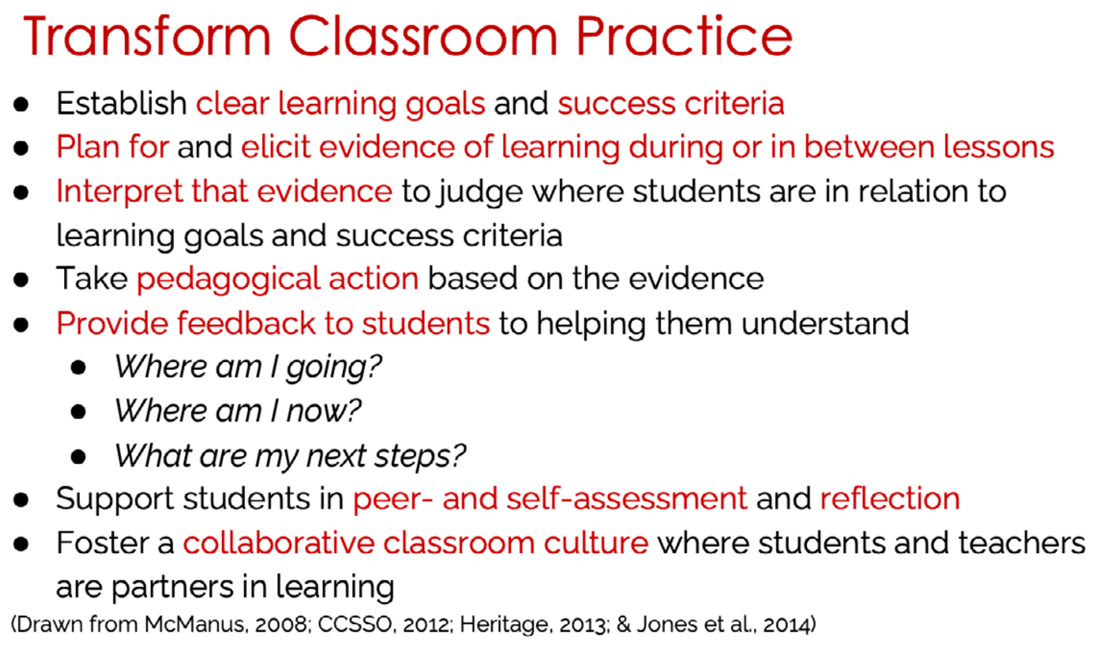
choice questions and easily gradable fixed answer types can probe and shine a light on conceptual understanding, and surface student difficulties and gaps in understanding. Past research in CS education has identified various kinds of good question types (Schulte et al., 2010). These, along with additional ones added by the author, are presented in Table 1.

Formative assessments, and specifically diagnostic items, should target the several known misconceptions and difficulties highlighted in CS education research (e.g. Soloway & Spohrer, 2013; Sorva, 2020). According to Ciofalo & Wylie (2006) and Wylie & Ciofalo (2008), what makes a diagnostic item particularly formative is that an incorrect response not only provides information about gaps in student understanding; it also provides insight into what it is that the student does not understand — in other words, the nature of their misconceptions.

Given the nature of formative assessment and its role in providing feedback on ongoing learning, formative assessments should also target learning progressions and the building blocks of programs that contribute to building program comprehension skills. Formative assessments can query student understanding of single concepts especially when a concept is first introduced. Given that learning of programming is intertwined with program syntax and semantics, it is also important that formative assessments target learning goals that encompass both structure and function as defined in Schulte's Block Model (Schulte, 2008, 2010) rather than only learning goals-oriented trajectories and progressions (such as those articulated by Rich et al., 2017). Examples of such items are shown in Figure 4. Bloom's taxonomy and SOLO taxonomy (Biggs & Collins, 1982; Clear et al., 2008; Lister et al., 2006; Thompson et al., 2008) have been used extensively in tertiary CS education assessment research and could similarly provide guidance on design of formative assessment items target varying levels of program comprehension and CT practices such as debugging, algorithmic thinking, and abstraction.

Teacher practice and formative assessment literacy

K-12 CS teachers' lack of confidence or knowledge and skills has impacted the implementation of assessments and the depth of feedback they provide (Vivian et al., 2020). It is therefore crucial to develop teachers' capacity and influence their habits of practice to make formative assessment integral to their teaching.



Transform Classroom Practice

- Establish clear learning goals and success criteria
- Plan for and elicit evidence of learning during or in between lessons
- Interpret that evidence to judge where students are in relation to learning goals and success criteria
- Take pedagogical action based on the evidence
- Provide feedback to students to helping them understand
    - *Where am I going?*
    - *Where am I now?*
    - *What are my next steps?*
- Support students in peer- and self-assessment and reflection
- Foster a collaborative classroom culture where students and teachers are partners in learning

(Drawn from McManus, 2008; CCSSO, 2012; Heritage, 2013; & Jones et al., 2014)

Figure 3. Classroom practice for supporting formative assessment (Source: Linquanti, 2014)

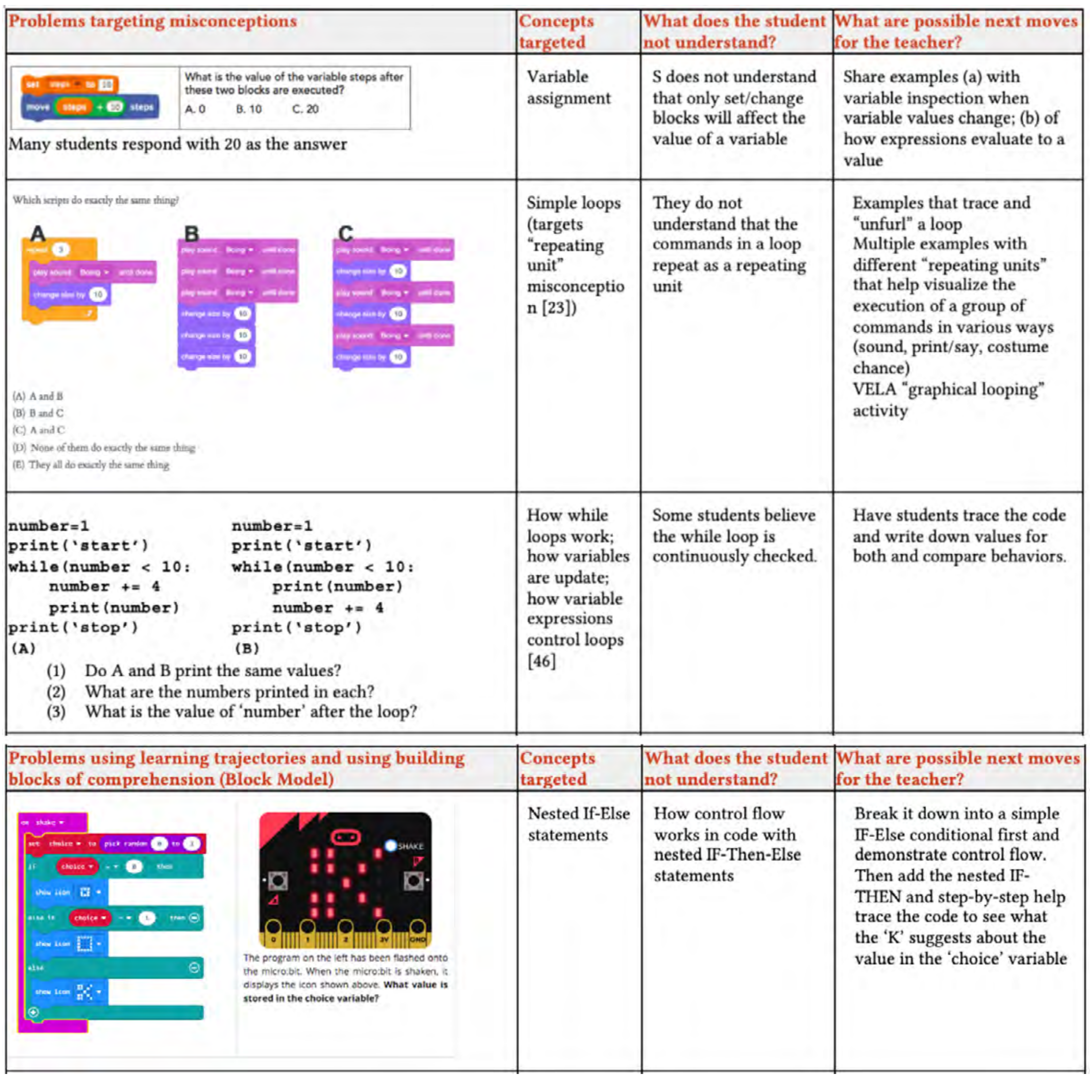
Teachers need to incorporate the formative assessment process as part of their routine classroom practice (Figure 3).

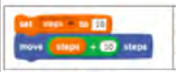
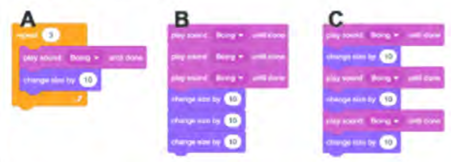
As part of the development of teacher PCK, teacher preparation needs to also help build in teachers an awareness of common targets of, and check students' understanding of, known targets of difficulty and misconceptions from CS education research. Teachers should have an understanding of how programming learning develops in novices and use items that target granular learning goals and elements of programming as guided by the Block Model (Schulte, 2008). Formative assessment literacy also provides teachers with the understanding of how to enact the formative assessment cycle of assessment, diagnosis, and formative action. Figure 4 provides examples of items targeting known misconceptions or granular learning goals along with the next moves teachers can make as part of the formative cycle.

Community resources: assessment repositories, feature-rich platforms, and collaboration

Teacher learning communities are a powerful mechanism to improve teachers' capabilities in using assessment in the service of learning. Teacher communities of practice (CoP) have been shown to sustain themselves around a shared need, and the give and take of shared resources for the benefit of all (Hoadley, 2012). Assessment item repositories are a useful mechanism, but only when they are well-designed to support a CoP of CS teachers (Fincher et al., 2010).

Extant and ongoing efforts for CS assessment item banks and repositories include Edfinity (edfinity.com), Project Quantum (https://diagnosticquestions.com/Quantum), Viva (Giordano et al., 2015), and the Canterbury Question Bank focused on introductory college-



Problems targeting misconceptions	Concepts targeted	What does the student not understand?	What are possible next moves for the teacher?
 What is the value of the variable steps after these two blocks are executed? A. 0 B. 10 C. 20 Many students respond with 20 as the answer	Variable assignment	S does not understand that only set/change blocks will affect the value of a variable	Share examples (a) with variable inspection when variable values change; (b) of how expressions evaluate to a value
Which scripts do exactly the same thing?  (A) A and B (B) B and C (C) A and C (D) None of them do exactly the same thing (E) They all do exactly the same thing	Simple loops (targets "repeating unit" misconception [23])	They do not understand that the commands in a loop repeat as a repeating unit	Examples that trace and "unfurl" a loop Multiple examples with different "repeating units" that help visualize the execution of a group of commands in various ways (sound, print/say, costume chance) VELA "graphical looping" activity

`number=1`
`print('start')`
`while(number < 10:`
`    number += 4`
`    print(number)`
`print('stop')`
`(A)`
 	
`number=1`
`print('start')`
`while(number < 10:`
`    print(number)`
`    number += 4`
`print('stop')`
`(B)`
(1) Do A and B print the same values?
(2) What are the numbers printed in each?
(3) What is the value of 'number' after the loop?
	How while loops work; how variables are update; how variable expressions control loops [46]	Some students believe the while loop is continuously checked.	Have students trace the code and write down values for both and compare behaviors.

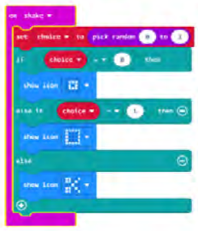
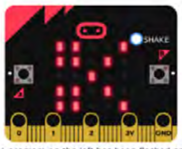
Problems using learning trajectories and using building blocks of comprehension (Block Model)	Concepts targeted	What does the student not understand?	What are possible next moves for the teacher?
  The program on the left has been flashed onto the micro:bit. When the micro:bit is shaken, it displays the icon shown above. **What value is stored in the choice variable?**	Nested If-Else statements	How control flow works in code with nested IF-Then-Else statements	Break it down into a simple IF-Else conditional first and demonstrate control flow. Then add the nested IF-THEN and step-by-step help trace the code to see what the 'K' suggests about the value in the 'choice' variable

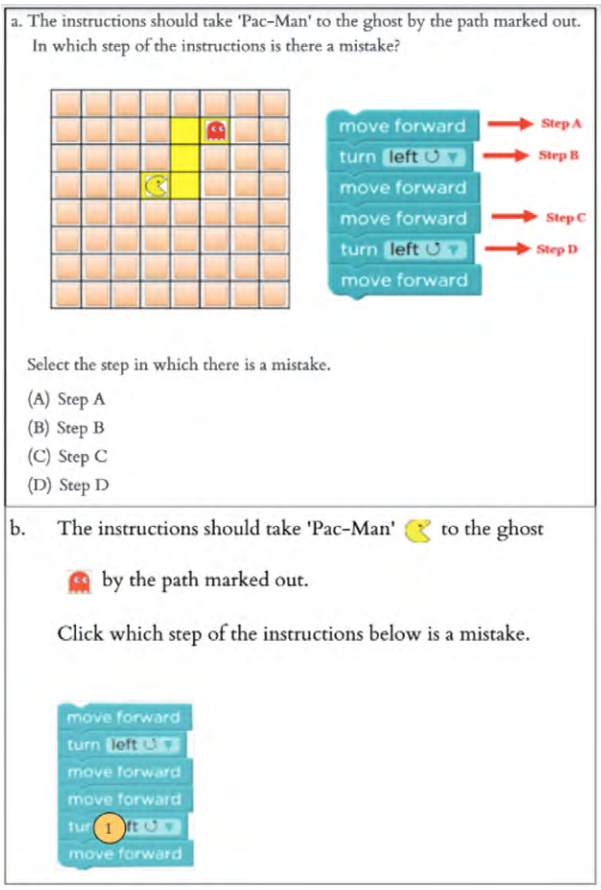
Figure 4. Examples based on research on misconceptions, learning trajectories, and levels of program comprehension, along with teacher diagnosis and formative action (Grover, 2021).

level CS (Sanders et al., 2013), with the author's current research contributing to the development of formative assessments on Edfinity.
In order to support formative assessment practice by teachers, assessment platforms and homework systems must be feature rich to support aggregation, creation, curation, and cataloging or taxonomising of assessments based on multiple and multi-level taxonomies relevant to CS teachers. This section uses Edfinity.com as an exemplar to describe such a platform. Taxonomies on Edfinity include CS/CT topics, learning standards (such as those from CSTA, 2017) or learning goals by curricula (such as AP CS Principles), grade, difficulty level, and ad hoc metadata (such as programming language) to support easy search and discovery. Edfinity also aids with

assessment delivery, administration, auto-grading, and teacher dashboards. Backend data and analytics on student performance provide teachers crucial insights into students' learning and understanding at individual and aggregate levels (Grover et al., 2014). Edfinity also provides for multiple attempts of a question, hints, and feedback (or explanation) for correct and incorrect options. Solution explanations accompany the item and serve as (a form of) feedback. These explanations, as also the question stem, support rich text, graphics, and video for better learner engagement and multiple modes (and languages) of presentation to equitably support diverse learners. Edfinity item types include technology-enhanced assessments that push the boundaries to include interactivity (such as hotspot and point-and-click items), drag-drop (for items types such as Parson's Problems), microworlds, and in-browser code entry and testing. Such items are not only engaging but also help reduce cognitive load (Figure 5). Assessment creation and aggregation functions on Edfinity also support features for teacher collaboration (à la Google Docs), contribution, attribution, and sharing, as well as interfaces for creation of both simple and technology-enhanced items. Furthermore, technology platforms could innovate with randomised variants of items, solution validation, and customised feedback to students. Such technology platforms should be affordable. However, tools such as Google Forms, while free and popular for formative assessment, do not auto-grade or afford many of the features important for formative assessment. Similarly, paper formative assessments cannot be autograded or leverage aforementioned affordances of technology.

In Closing

This position paper makes a persuasive argument for formative assessment and teacher formative assessment literacy in K-12 CS,



a. The instructions should take 'Pac-Man' to the ghost by the path marked out. In which step of the instructions is there a mistake?

```
move forward      → Step A
turn left ↺       → Step B
move forward
move forward      → Step C
turn left ↺       → Step D
move forward
```

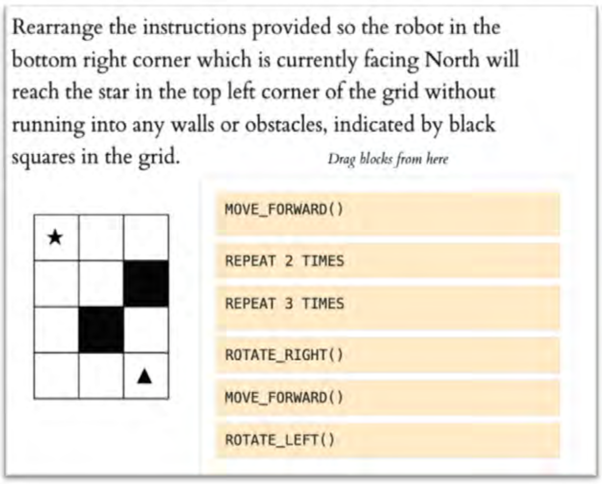
Select the step in which there is a mistake.

(A) Step A
(B) Step B
(C) Step C
(D) Step D

b. The instructions should take 'Pac-Man' to the ghost by the path marked out.

Click which step of the instructions below is a mistake.

```
move forward
turn left ↺
move forward
move forward
tur 1 ft ↺
move forward
```



Rearrange the instructions provided so the robot in the bottom right corner which is currently facing North will reach the star in the top left corner of the grid without running into any walls or obstacles, indicated by black squares in the grid.

```
MOVE_FORWARD()
REPEAT 2 TIMES
REPEAT 3 TIMES
ROTATE_RIGHT()
MOVE_FORWARD()
ROTATE_LEFT()
```

Figure 5a. An MCQ item from CTt () adapted into a point- and-click item (more intuitive and lower cognitive barriers) on Edfinity.com. (5b) A Parson's Puzzle Problem for AP CS Principles.

keeping the goal of robust student learning in mind. The framework presents several key ideas that can serve to provide guidance on taking important first steps to make both CS classroom teaching and learning as well as CS teacher preparation more robust through attention to formative assessments and formative assessment literacy, respectively. More work is needed, however, especially around classroom research on the use of formative assessments in different contexts and for various concepts. The framework as presented and explicated is focused on conceptual learning of programming. Although CS is certainly more than conceptual learning of programming, through focusing the framework and its dimensions on conceptual learning and examples of formative assessment forms, along with designs, tools, and guidance for providing convenient and powerful formative feedback, this paper makes a start in addressing a crucial lacuna.

References

Armoni, M. (2014). Spiral thinking: K--12 computer science education as part of holistic computing education. *ACM Inroads*, 5(2), 31-33.

CSTA et al. (2017). *CSTA K-12 computer science standards*, revised 2017. Computer Science Teachers Association, USA, 2017.

Barron, B., & Darling-Hammond, L. (2008). How can we teach for meaningful learning? In L. Darling-Hammond et al., editors, *Powerful learning: What we know about teaching for understanding*, 1, 11-16. Jossey-Bass, San Francisco, 2008.

Basawapatna, A. R., Repenning, A., & Koh, K. H. (2015, February). Closing the cyberlearning loop: Enabling teachers to formatively assess student programming projects. *In Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 12-17).

Biggs, J. B., & Collis, K. (1982). *Evaluating the Quality of Learning: the SOLO taxonomy*. New York: Academic Press..

Black, P., & Wiliam, D. (1998). Assessment and classroom learning. *Assessment in Education: principles, policy & practice*, 5(1), 7-74.

Black, P., & Wiliam, D. (2009). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education*), 21(1), 5.

Bloom, B. S. (1969). Some theoretical issues relating to educational evaluation. *Educational evaluation: new roles, new means: the 63rd yearbook of the National Society for the Study of Education*, 69, 26-50.

Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn* (Vol. 11). Washington, DC: National academy press.

Brookhart, S. M. (2003). Developing measurement theory for classroom assessment purposes and uses. *Educational measurement: Issues and practice*, 22(4), 5-12.

Ciofalo, J., & Wylie, C. E. (2006). Using diagnostic classroom assessment: one question at a time. *Teachers College Record*, 108(1).

Clear, T., Whalley, J., Lister, R. F., Carbone, A., Hu, M., Sheard, J., ... & Thompson, E. (2008). Reliably classifying novice programmer exam responses using the SOLO taxonomy. *National Advisory Committee on Computing Qualifications*.

Council of Chief State School Officers (CCSSO). (2012). *Distinguishing formative assessment from other educational assessment labels*. Washington, DC: Author. Retrieved from https://www.michigan.gov/documents/mde/CCSSO_Assessment__Labels_Paper_ada_601108_7.pdf

Crooks, T. J. (1988). The impact of classroom evaluation practices on students. *Review of educational research*, 58(4), 438-481.

DeLuca, C., Valiquette, A., Coombs, A., LaPointe-McEwan, D., & Luhanga, U. (2018). Teachers' approaches to classroom assessment: A large-scale survey. *Assessment in Education: Principles, Policy & Practice*, 25(4), 355-375.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008, September). Evaluating a new exam question: Parsons problems. *In Proceedings of the fourth international workshop on computing education research* (pp. 113-124).

Fincher, S., Kölling, M., Utting, I., Brown, N., & Stevens, P. (2010, August). Repositories of teaching material and communities of use: nifty assignments and the greenroom. *In Proceedings of the Sixth international workshop on Computing education research* (pp. 107-114).

Giordano, D., Maiorana, F., Csizmadia, A. P., Marsden, S., Riedesel, C., Mishra, S., & Vinikienė, L. (2015). New horizons in the assessment of computer science at school and beyond: Leveraging on the viva platform. In *Proceedings of the 2015 ITiCSE on Working Group Reports* (pp. 117-147).

González, M. R. (2015). Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 conference* (pp. 2436-2444).

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In *Emerging research, practice, and policy on computational thinking* (pp. 269-288). Springer, Cham.

Grover, S. (2020, February). Designing an Assessment for Introductory Programming Concepts in Middle School Computer Science. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 678-684).

Grover, S. (2021). Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM.

Grover, S., Basu, S., & Schank, P. (2018, February). What we can learn about student learning from open-ended programming projects in middle school computer science. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 999-1004).

Grover, S., Pea, R., & Cooper, S. (2014, March). Promoting active learning & leveraging dashboards for curriculum assessment in an OpenEdX introductory CS course for middle school. In *Proceedings of the first ACM conference on Learning@ scale conference* (pp. 205-206).

Grover, S., Sedgwick, V., and Powers, K. (2020) Feedback through formative check-ins. In S. Grover, Ed., *Computer Science in K-12: An A to Z Handbook on Teaching Programming*. Edfinity, 2020.

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of educational research*, 77(1), 81-112.

Heritage, M. (2010). Formative Assessment and Next-Generation Assessment Systems: Are We Losing an Opportunity?. *Council of Chief State School Officers*.

Heritage, M. (2013). *Formative assessment in practice: A process of inquiry and action*. Harvard Education Press.

Heritage, M. (2018). Supporting Teachers' Successful Implementation of Formative Assessment. Presentation for Assessment Learning Network, Michigan Assessment Consortium.

Heritage, M., & Wylie, C. (2018). Reaping the benefits of assessment for learning: Achievement, identity, and equity. *ZDM*, 50(4), 729-741.

Hoadley. C. (2012). What is a community of practice and how can we support it? In Land, S., & Jonassen, D. (Eds.). *Theoretical Foundations of Learning Environments* (2nd ed.). Routledge.

Hudesman, J., Crosby, S., Flugman, B., Issac, S., Everson, H., & Clay, D. B. (2013). Using formative assessment and metacognition to improve student achievement. *Journal of Developmental Education*, 37(1), 2.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., ... & Weeda, R. (2019). Fostering Program Comprehension in Novice Programmers-Learning Activities and Learning Trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 27-52).

Jones, B., Chang, S., Heritage, M., Tobiason, G., & Herman, J. O. A. N. (2014). Supporting students in close reading. *National center for research on Evaluation, Standards, and Student Testing.*—Los Angeles: University of California, 5.

Linquanti, R. (2014). Supporting formative assessment for deeper learning: A primer for policymakers. *Formative Assessment for Students and Teachers/State Collaborative on Assessment and Student Standards. Washington, DC: CCSSO*. Retrieved from https://www.michigan.gov/documents/mde/CCSSO_Supporting_Formative_Assessment_for_Deeper_Learning_601111_7.pdf

References

Lister, R. (2005, January). One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple choice exam for first semester programming students. In *Conferences in Research and Practice in Information Technology Series*.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin*, 38(3), 118-122.

McManus, S. (2008). *Attributes of effective formative assessment*. CCSSO.

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.

Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, 31(2), 199-218.

Parsons, D., & Haden, P. (2006, January). Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 157-163).

Popham, W. J. (2009). Assessment literacy for teachers: Faddish or fundamental?. *Theory into practice*, 48(1), 4-11.

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017, August). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM conference on international computing education research* (pp. 182-190).

Salac, J., & Franklin, D. (2020, June). If They Build It, Will They Understand It? Exploring the Relationship between Student Code and Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 473-479).

Sanders, K., Ahmadzadeh, M., Clear, T., Edwards, S. H., Goldweber, M., Johnson, C., ... & Spacco, J. (2013, June). The Canterbury QuestionBank: building a repository of multiple-choice CS1 and CS2 questions. In *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports* (pp. 33-52).

Sadler, D. R. (1989). Formative assessment and the design of instructional systems. *Instructional science*, 18(2), 119-144.

Schulte, C. (2008, September). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 149-160).

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports* (pp. 65-86).

Shulman, L. (1987). Knowledge and teaching: Foundations of the new reform. *Harvard educational review*, 57(1), 1-23.

Soloway, E., & Spohrer, J. C. (Eds.). (2013). *Studying the novice programmer*. Psychology Press.

Sorva, J. (2020). Naive conceptions of novice programmers. In S.Grover, Ed., *Computer Science in K-12: An A to Z Handbook on Teaching Programming*. Edfinity, 2020.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008, January). Bloom's taxonomy for CS assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78* (pp. 155-161).

Vivian, R., & Falkner, K. (2018, October). A survey of Australian teachers' self-efficacy and assessment approaches for the K-12 digital technologies curriculum. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education* (pp. 1-1)

Vivian, R., Franklin, D., Frye, D., Peterfreund, A., Ravitz, J., Sullivan, F., ... & McGill, M. M. (2020, June). Evaluation and Assessment Needs of Computing Education in Primary Grades. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124-130).

Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). CodeMaster--Automatic Assessment and Grading of App Inventor and Snap! Programs. *Informatics in Education*, 17(1), 117-150.

Wiebe, E., London, J., Aksit, O., Mott, B. W., Boyer, K. E., & Lester, J. C. (2019, February). Development of a lean computational thinking abilities assessment for middle grades students. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 456-461).

Wiliam, D. (2006). Formative assessment: Getting the focus right. *Educational assessment, 11*(3-4), 283-289.

Wiliam, D., & Leahy, S. (2012). Sustaining formative assessment with teacher learning communities. In *PERIHA Professional Learning Series Workshop, Ministry of Education*.

Wylie, E. C., & Ciofalo, J. (2008). Supporting teachers' use of individual diagnostic items. *Teachers College Record*.

Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., & Clayborn, L. (2015). Sowing the seeds: A landscape study on assessment in secondary computer science education. *Comp. Sci. Teachers Assn.*, NY, NY.

Section 3:
Computing topics

Section 3: Computing topics

The role of block-based programming in computer science education

David Weintrop (University of Maryland, USA)

Abstract

Block-based programming environments are increasingly becoming the way that young learners are being introduced to the practice of programming and the field of computer science more broadly. Environments such as Scratch, MIT AppInventor, Code.org's AppLab, and block-based interfaces for physical devices provide inviting and accessible pathways into the world of programming. In this article, I share findings from a series of studies investigating the use of block-based programming in K-12 classrooms. In particular, this research compares block-based programming to conventional text-based programming languages and explores the transition from introductory block-based tools to professional programming languages. The results of the study found that high school students score better on tests after learning to program in a block-based tool compared to peers who learned with a text-based language. The study also found that after transitioning to a professional text-based programming language (Java), there was no difference in programming performance in terms of scores on a content assessment or differences in programming practices employed. The implications of these findings suggest that block-based programming is an effective way to introduce learners to programming but open questions remain about how to best integrate it into formal classroom instruction.

Introduction

Led by the popularity of environments like Scratch, MIT AppInventor, and the growing ecosystem of programming environments built with the Blockly library, block-based programming is increasingly becoming the way that learners are being introduced to the practice of programming and the field of computer science more broadly (Bau et al., 2017; Resnick et al., 2009; Weintrop, 2019). Along with virtual programming environments, a growing number of physical devices support block-based programming, including Sphero, BBC micro:bit, Lego Mindstorms, and several block-based programming environments for the Raspberry Pi family of microprocessors. While not a recent innovation (block-based environment first emerged in the mid-1990s), the last decade has seen a blossoming of block-based programming environments and computing curricula that rely upon block-based tools. A recent review of the academic literature identified 99 unique block-based programming environments (Lin & Weintrop, 2021). This has, in turn, lead to a growing body of research seeking to understand the affordance of block-based tools and articulate their role in computer science education (Franklin et al., 2017; Grover & Basu, 2017; Price & Barnes, 2015; Weintrop, Hansen, et al., 2018). As block-based tools become more widespread, it is important that we as educators understand the affordances and drawbacks of these environments so we are best able to

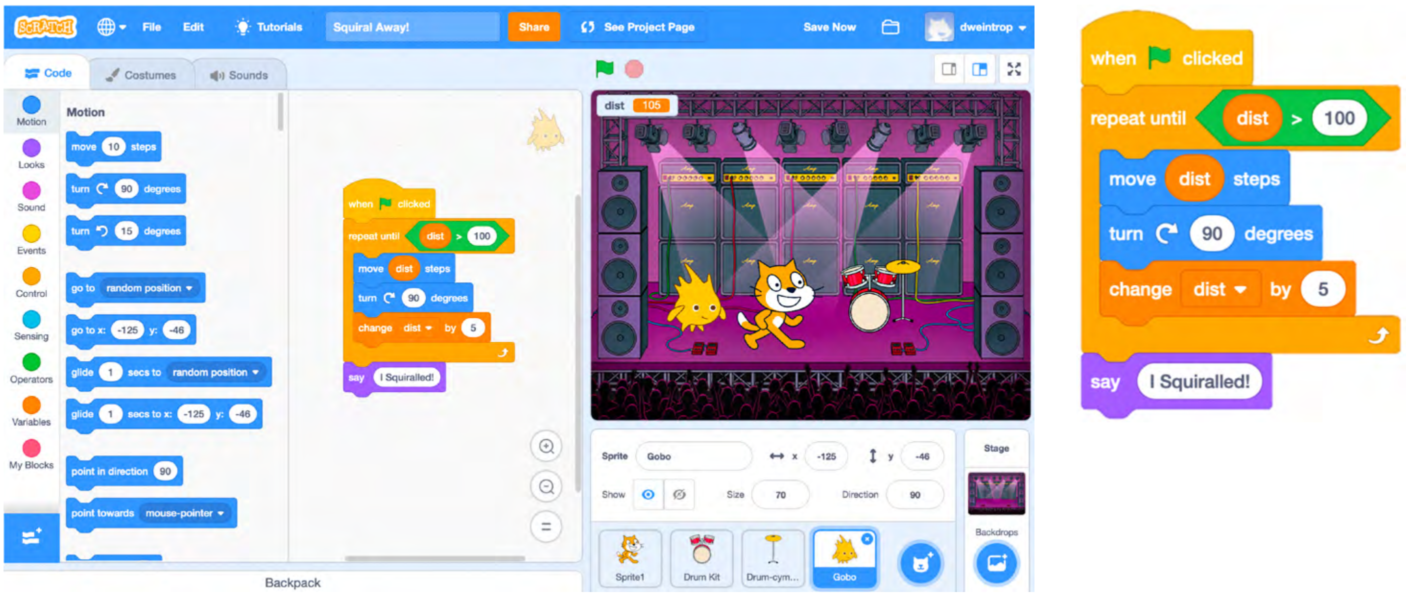


Figure 1. The Scratch programming environment (left) and a block-based program written in Scratch (right)

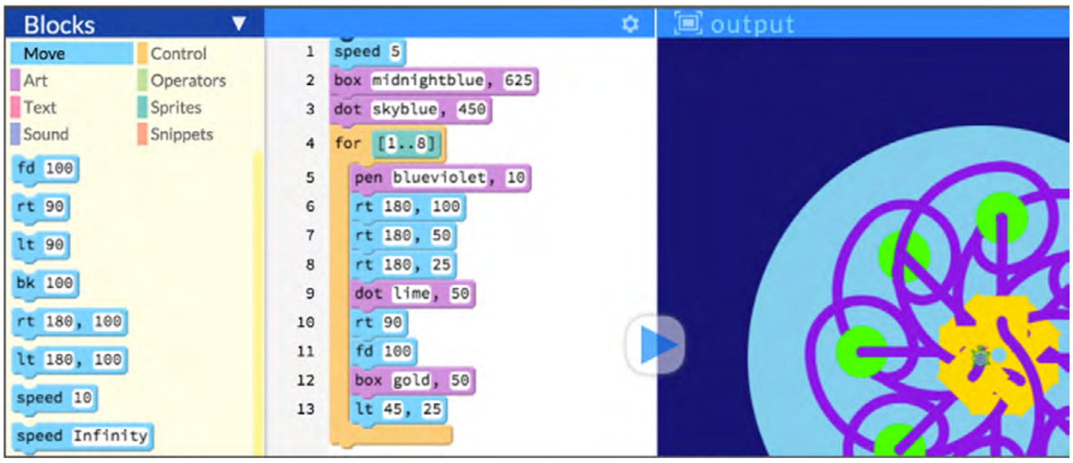
support learners early in their computer science careers.

The goal of this article is to present findings from a series of research studies seeking to understand the impact of using block-based programming environments in classrooms. In particular, we pursue questions seeking to understand how block-based instruction compares to text-based instruction and to understand if and how the experience of learning to program in a block-based environment better prepares learners for future text-based programming. In doing so, this work seeks to elucidate the potential role of block-based programming in formal education and equip educators to effectively use block-based programming as part of their instruction.

What is block-based programming?

Block-based programming is a graphical approach to programming that uses a programming-command-as-puzzle-piece metaphor to visually convey information about the programming commands available to the user and how they can be used (Figure 1). Through the inclusion of visual, organisational, and audio cues, block-based programming environments can help novices write functioning programs from the start. The defining feature of block-based programming environments, and the source of their name, is that programming commands are presented as blocks where the shape of the block defines how and where it can be used (Maloney et al., 2010). To assemble a program, the user drags blocks onto the canvas (the area where the program is written) and snaps the blocks together, often accompanied by an audible click. Only valid combinations of blocks can be snapped together, in this way, block-based programming environments can prevent syntax errors by not allowing for invalid programs to be written.

Along with the visual layout of the blocks, there



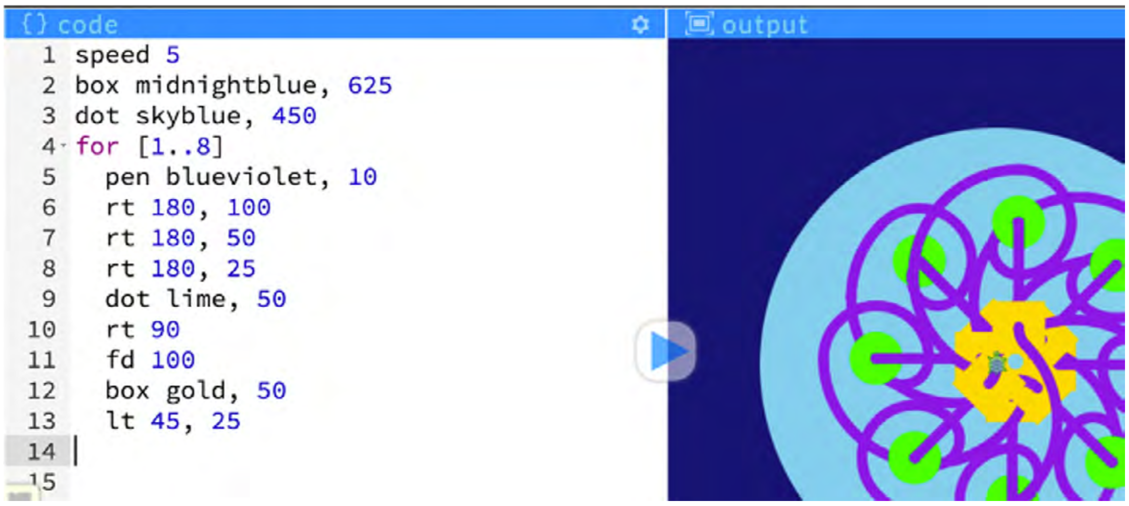
```
speed 5
box midnightblue, 625
dot skyblue, 450
for [1..8]
  pen blueviolet, 10
  rt 180, 100
  rt 180, 50
  rt 180, 25
  dot lime, 50
  rt 90
  fd 100
  box gold, 50
  lt 45, 25
```

Figure 2. The block-based version of the Pencil.cc programming environment.

are a number of other features that make block-based programming easy for novices with little prior programming experience. For example, as part of a study of high school students learning to program, students talked about how the arrangement of available blocks (left side of Figure 1) made it easy to figure out what was possible in the programming language (Weintrop & Wilensky, 2015a). Students also talked about how the drag-and-drop approach to assembling programs was easier than typing in programming commands one character at a time. This is especially true considering many programming languages require the user to type in uncommon and often mysterious punctuation as part of writing a functioning program. Another feature of block-based language students cited as contributing to their ease-of-use is how the blocks themselves are easier to read when compared to a conventional text-based language. As one student said, *"Java is not in English it's in Java language, the blocks are in English, it's easier to understand"*. Collectively, these various affordances lead learners to perceive block-based programming to be easier for novices.

The case for block-based programming

A central and important question about the potential role of block-based programming environments in formal education is whether or not students learn computer science concepts when programming in block-based environments. A related question is how students learn with block-based environments compared to comparable text-based programming languages? In other words, do students learn more in blocks or text? To answer this question, I conducted a quasi-experimental study in two high-school computer science classrooms. Students in one classroom learned using a block-based programming environment (Figure 2) while students in the other classroom used a text-based programming environment (Figure). Importantly, everything about the environments was identical aside from the way programs were presented and authored, including the programming language itself, which was the exact same character-by-character between the two environments. The study began on the first day of school and lasted for five weeks with both classes going through the same curriculum and being taught by the same teacher. As much as



```
speed 5
box midnightblue, 625
dot skyblue, 450
for [1..8]
  pen blueviolet, 10
  rt 180, 100
  rt 180, 50
  rt 180, 25
  dot lime, 50
  rt 90
  fd 100
  box gold, 50
  lt 45, 25
```

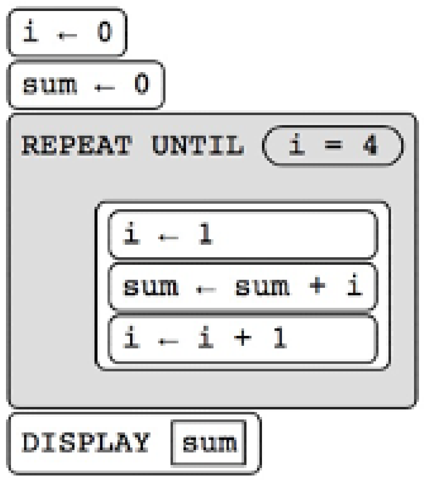
Figure 3. The text-based version of the Pencil.cc programming environment.

possible, everything was kept constant between the two classrooms aside from the programming environment.

After learning to program in either the block-based or text-based environments, students took a programming assessment where the questions were asked in both block-based and text-based forms (Weintrop & Wilensky, 2015b). At the conclusion of the five-weeks of instruction, students who learned with the block-based environment scored higher on the content assessment than their peers who learned with the text-based environment (Weintrop & Wilensky, 2017a). This finding is important evidence showing block-based programming to be an effective way to introduce novices to programming.

As part of this study, students also took an attitudinal survey to explore their interest in computer science, their confidence with the discipline, and get an overall sense of their feelings about computer science. After working in a block-based environment for five weeks, learners were significantly more confident in their computer science abilities and their interest in the field had grown (Weintrop & Wilensky, 2017a).

This study was particularly focused on one block-based programming environment (Figures 2 and 3), however, the finding that students perform better in block-based environments has been replicated in other work. For example, through a partnership with code.org, I investigated how students performed on a computer science content assessment that asked questions using pseudocode presented in both block-based and text-based forms (Figure 4). This pseudocode was developed for the Advanced Placement (AP) Computer Science Principles (CSP) exam that is administered to high school students across the United States. The challenge with this assessment is that the organisation that designs and administers the test does not know what programming language students have learned with or if they learned in a block-based or text-based environment. As such, the test must be appropriate for learners who learned to program with block-based environments and learners who learned with text-based languages. The solution to this problem was for the AP CSP test to use a pseudocode



```
i ← 0
sum ← 0
REPEAT UNTIL (i = 4)
    i ← 1
    sum ← sum + i
    i ← i + 1
DISPLAY sum
```

(a)

```
i ← 0
sum ← 0
REPEAT UNTIL (i = 4)
{
   i ← 1
   sum ← sum + i
   i ← i + 1
}
DISPLAY (sum)
```

(b)

Figure 4. The (a) block-based and (b) text-based pseudocode from the AP Computer Science Principles exam.

with both a block-based form (Figure 4a) and a text-based form (Figure 4b).

An analysis of over 5,000 students from across the United States who took a 20-question content assessment comprised both block-based and text-based questions using AP CSP's pseudocode found that students scored significantly higher on questions asked in the block-based form than questions asked in the text-based form (Weintrop et al., 2019). Further, in breaking down results by race and gender, we found that women and students from racial backgrounds that have been historically excluded in computing saw greater benefits to questions asked in the block-based form (Weintrop & Killen et al., 2018). This finding provides additional evidence for the importance of including block-based programming in formal education, especially as it relates to goals of equity and broadening participation in the field.

Drawbacks and challenges

While the evidence presented above shows the value of block-based instruction in K-12 classrooms, this work also identified some drawbacks and challenges related to the use of block-based environments in classrooms. In analysing student feedback to identify what students found to be useful about block-based programming, we also found that students identified a series of drawbacks (Weintrop & Wilensky, 2015a). For example, some students expressed concerns related to the authenticity of block-based programming, as one student put it, *"if we actually want to program something, we wouldn't have blocks."* Other drawbacks mentioned by students included concerns that block-based programming environments were inherently less powerful than text-based programming languages and that writing programs in block-based environments was slower than authoring programs in text-based languages.

A second drawback, or at least an open question, related to block-based programming is if and how block-based programming prepares learners for future computer science instruction using text-based programming languages. In a continuation of the study discussed above, after five weeks of learning in either a block-based or text-based introductory programming environment, we followed students as they transitioned to instruction in Java. After ten weeks of learning Java, students in both conditions took another content assessment. The result of that assessment showed that there was no difference in performance on the assessment based on their introductory experiences (Weintrop & Wilensky, 2019). That is to say, students scored the same on the assessment after ten weeks of Java instruction regardless of which introductory environment they used, so the gains found after five weeks for students learning in the block-based environment were no longer present. We also found there to be no significant difference in terms of the programming practices employed while authoring programs and that students from both introductory experiences showed similar patterns in the types and frequency of syntax errors encountered (Weintrop & Wilensky, 2018). One important thing to note about this study was that the teacher who taught these classes did not employ any specific pedagogical strategies to help bridge the transition from block-based to text-based programming. In other studies where successful transfer has been documented, there are usually explicit bridging strategies employed by the instructor(s) to help learners make the transition (Dann et al., 2012). Questions related to pedagogy and how best to prepare instructors to teach computer science remains an active area of research (Franklin et al., 2020; Yadav & Berges, 2019).

Implications and recommendations

Implications

The primary implication of this research is that block-based programming has a home in computer science classrooms. However, there are still open questions that need to be answered in terms of how best to use block-based programming to help support learning, both in the classroom and beyond (Brown et al., 2016). While much work remains to be done to figure out exactly how best to utilise this programming approach, the findings cited above and reported elsewhere show block-based programming to be an effective way to introduce novices to the practice of programming and the field of computer science more broadly.

A second implication from this work stems from the finding that students who learned using a block-based programming environment did not see any significant advantage from that experience compared to their text-based peers after transitioning to a text-based language. The important takeaway from this finding is the idea that transfer does not come for free. That is to say, while there are clear conceptual links between programming in a block-based environment and programming in a text-based language, learners do not necessarily see those links and make the connections on their own. This is a place where pedagogy and the teacher play an essential role. Providing explicit instruction to help learners make the connection between blocks and specific programming keywords can help scaffold that transition and help learners build upon conceptual gains made in block-based tools. While there is some work showing this to be effective (Dann et al., 2012), more work needs to be done to more fully understand how best to support learners in making this transition.

Recommendations

So, at the end of the day, where does that leave us in terms of what is the best way to teach students to program? When I am asked by teachers if they should use a block-based environment or start with a text-based programming language, my response is: why not both? Up to this point, block-based environments and text-based languages have been presented as mutually exclusive options. However, this need not be the case. There are a growing number of programming environments that blend block-based and text-based features like BlueJ's Frame-based editor (Kölling et al., 2015) and others that support both block-based and text-based programming like Pencil code (Bau et al., 2015). I am increasingly excited about programming environments that support both block-based and text-based programming, where the learners can decide which interface they want to see and can move back and forth between the two forms. I call these dual-modality environments (Weintrop & Wilensky, 2017b) and a growing body of research is showing them to be an effective approach to support novices early in their learning while also providing scaffolds for them to transition from block-based composition to more conventional text-based programming (Blanchard et al., 2020; Matsuzawa et al., 2015; Weintrop & Holbert, 2017).

A second important question to ask when thinking about the role of block-based programming in computer science education, especially as it relates to the transition to text-based instruction, is whether or not that transition is even necessary. Do all students need to learn to program in professional text-based languages? If the goal is to prepare students for a career in computer science, then the answer is probably yes, students would need to learn to program with professional text-based languages. However, it is worth re-examining whether the goal of computer science instruction should be to prepare learners for careers in the field. While that certainly is one desirable endpoint of computer science instruction, it is important to consider alternative endpoints, such as preparing learners for careers outside of computer science, equipping students to be informed technologically-savvy citizens, and empowering learners to pursue their own goals and interests through computing (Tissenbaum et al., In Press). The idea that block-based programming may be a sufficient endpoint for computer science instruction is also bolstered by the growing number of block-based environments designed for real-world applications such as data sciences (Bart et al., 2017) and industrial robotics programming (Weintrop, Afzal, et al., 2018).

Conclusion

The goal of this article was to share findings from research investigating the role of block-based programming in computer science education. While block-based environments such as Scratch have had a significant impact on youth learning to program in informal environments, the role of block-based programming in formal classroom contexts was less clear. In this article, I presented results that show block-based programming to be an effective way to welcome learners to the field of computer science. At the same time, there are still open questions related to how best to utilize block-based environments as part of formal computer science instruction. In particular, how to address student concerns around questions of authenticity and how to effectively scaffold learners in the transition to text-based languages. In discussing these challenges, I put forward the idea of dual-modality programming environments that support both block-based and text-based forms of authorship as one potential way to address this concern. The work reviewed here, along with the growing body of research around the design of introductory programming environments, curricula, and pedagogy, collectively are poised to lay the groundwork for the infrastructure needed to prepare all learners to succeed in an increasingly computational world.

References

Bart, A. C., Tibau, J., Kafura, D., Shaffer, C. A., & Tilevich, E. (2017). Design and Evaluation of a Block-based Environment with a Data Science Context. *IEEE Transactions on Emerging Topics in Computing, PP*(99), 1–1.

Bau, D, Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. *Proc. of the 14th Int. Conference on Interaction Design and Children*, 445–448.

Bau, David, Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM*, 60(6), 72–80.

Blanchard, J., Gardner-McCune, C., & Anthony, L. (2020). Dual-Modality Instruction and Learning: A Case Study in CS1. *Proc. of the 51st ACM Technical Symposium on Computer Science Education*, 818–824.

Brown, N. C. C., Mönig, J., Bau, A., & Weintrop, D. (2016). Future Directions of Block-based Programming. *Proc. of the 47th ACM Technical Symposium on Computing Science Education*, 315–316. Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. *Proc. of the 43rd ACM Technical Symposium on Computer Science Education*, 141–146.

Franklin, D, Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., Weintrop, D., & Harlow, D. (2017). Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. *Proc. of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 231–236.

Franklin, D, Coenraad, M., Palmer, J., Eatinger, D., Zipp, A., Anaya, M., White, M., Pham, H., Gökdemir, O., & Weintrop, D. (2020). An Analysis of Use-Modify-Create Pedagogical Approach's Success in Balancing Structure and Student Agency. *Proc. of the 2020 ACM Conference on International Computing Education Research*, 14–24.

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proc. of the 2017 ACM Technical Symposium on Computer Science Education*, 267–272.

Kölling, M., Brown, N. C. C., & Altadmri, A. (2015). Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. *Proc. of the Workshop in Primary and Secondary Computing Education*, 29–38.

Lin, Y., & Weintrop, D. (2021). The Current Landscape of Block-based Programming Environments. *Paper Presented at the Annual Meeting of the American Educational Research Association* (2021.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin, 40*(1), 367–371.

Maloney, J. H, Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Trans. on Computing Education, 10*(4), 16.

Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. *Proc. of the 46th ACM Technical Symposium on Computer Science Education*, 185–190.

Price, T. W., & Barnes, T. (2015). *Comparing Textual and Block Interfaces in a Novice Programming Environment*. 91–99.

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

Tissenbaum, M., Weintrop, D., Holbert, N., & Clegg, T. (In Press). The Case for Alternative Endpoints in Computing Education. *British Journal of Educational Technology*.

Weintrop, D. (2019). Block-based Programming in Computer Science Education. Commun. *ACM*, 62(8), 22–25.

Weintrop, D, Afzal, A., Salac, J., Francis, P., Li, B., Shepherd, D. C., & Franklin, D. (2018). Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. *Proc. of the 2018 CHI Conference on Human Factors in* Computing Systems, 366:1-12.

Weintrop, D, Hansen, A. K., Harlow, D. B., & Franklin, D. (2018). Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next. Proc. of the 2018 ACM Conference on International Computing Education Research, 142–150.

Weintrop, D, & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. Proc. of the 2017 ACM Technical Symposium on Computer Science Education, 633–638.

Weintrop, D, Killen, H., & Franke, B. (2018). Blocks or Text? How programming language modality makes a difference in assessing underrepresented populations. Proc. of the International Conference on the Learning Sciences 2018, 328–335.

Weintrop, D, Killen, H., Munzar, T., & Franke, B. (2019). Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. Proc. of the 50th ACM Technical Symposium on Computer Science Education, 1218–1224.

Weintrop, D, & Wilensky, U. (2017a). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. ACM Trans. on Computing Education, 18(1), 3.

Weintrop, D, & Wilensky, U. (2018). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. International Journal of Child-Computer Interaction, 17, 83–92.

Weintrop, D, & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. Computers & Education, 142, 103646.

Weintrop, D, & Wilensky, U. (2017b). Between a Block and a Typeface: Designing and Evaluating Hybrid Programming Environments. Proc. of the 2017 Conference on Interaction Design and Children, 183–192.

Weintrop, D, & Wilensky, U. (2015a). To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. Proc. of the 14th International Conference on Interaction Design and Children, 199–208.

Weintrop, D, & Wilensky, U. (2015b). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. Proc. of the 11th Annual International Computing Education Research Conference, 101–110.

Yadav, A., & Berges, M. (2019). Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. ACM Trans. on Computing Education, 19(3), 1–24.

Section 3: Computing topics

Learning artificial intelligence at school with Scratch and LearningML

Juan David Rodríguez (Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado (INTEF), Spain)

Introduction

Digital technologies are changing so quickly that people can become overwhelmed when dealing with new artifacts, both physical (hardware) and logical (software). However, in spite of this fast development, every new digital technology is based upon and can be explained by a set of well-established computer science (CS) principles.

For this reason, learning and teaching CS fundamentals to young people is crucial in order to help them become critical citizens able to understand and live comfortably in our increasingly digital society. Indeed, countries around the world have included in their educational curricula programming, robotic, and CS related content, intended to develop computational thinking (CT) skills (Wing, 2006) and achieve digital literacy.

Thanks to the creation of educational tools such as block-based programming platforms and the development of the CT concept, learning and teaching CS fundamentals is possible even at a very early age. A key feature of these tools is that they “engage and excite students in the first place” (Malan & Leitner, 2007, p. 6) and also allow students to solve problems that are important to them and their communities. This connection with young people’s ideas and interests is a powerful incentive to learn, providing an engaging and enjoyable way for them to learn CS principles.

A top-down strategy is often taught when children are introduced to programming. First, the problem to be solved must be well understood; that is, it must be carefully analysed. Second, a set of rules able to solve the problem must be deduced, which is roughly speaking the algorithm. And finally, this set of rules must be turned into a computer program, by using a programming language to write the program code.

A broad range of problems can be solved by following this strategy. However, there are some kinds of problem which, while very easy to solve for a human, are very difficult to code as a computer program when a top-down strategy is followed. Recognising, in a set of pictures of cats and dogs, which are cats and which are dogs, is a clear example: any human can perform this recognition naturally, but obtaining by deduction a set of rules that enable you to build a computer program which solves this recognition problem is practically impossible.

Image recognition and classification, natural language understanding, sound recognition, and many other problems involving some kind of pattern extraction resist being solved using the traditional top-down strategy. Instead, a bottom-up strategy — inducing the rules that govern the problem from automatic data analysis — is followed when dealing with such tasks. And this way of attacking the problem takes us into the field of artificial intelligence (AI) techniques.

Children are used to using computer applications capable of performing these kinds of tasks. They talk to their mobile phones to ask questions or search for information, use translation applications to translate into foreign languages, unlock their devices by showing their faces to the camera, and so on. Therefore, educational tools that allow children to design AI-based applications capable of dealing with these problems will help them to have a complete perspective about what can be done with a computer, and at the same time will allow students to solve problems that are of interest to them.

Furthermore, data is a prevalent concept in today's technological and economic development. Having an awareness of the central role that data plays in our lives and knowing how it is used to extract useful knowledge is a key factor in understanding digital society and, hence, achieving the digital literacy needed to understand the world we live in.

So, how are these problems solved? Can we teach children to solve them? At what age? These are questions we tried to ask with the development of LearningML, a tool intended to teach the fundamentals of machine learning (ML), the most widely used technique today for solving problems from data.

Artificial intelligence and machine learning

ML is considered a subfield of AI, and the latter is a subfield of CS and almost as old as CS itself. In 1950, Alan Turing wrote a seminal paper entitled "Computing Machinery and Intelligence". Although the term AI does not appear in the paper, it is considered the birth of the field, since the big question it posed was: "Can machines think?" (p. 1)
A few years later, in 1956, McCarthy, Minsky, Rochester, and Shannon led a workshop at Dartmouth College which aimed to gather a selected group of scientists to work "on the basis of conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" (McCarthy, Minsky, Rochester, & Shannon, 2006, p. 1). Their proposal where the goal of the workshop was described was titled "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence", which was the first use of the term AI.

Since then, the field of AI has grown and its development has alternated between optimistic and pessimistic periods. After more than sixty years of research, many related subfields have emerged. Planning and problem solving, natural language processing, knowledge representation, expert systems, neural networks, machine learning, robotics and computer vision are some of the most successful and broadly used in current applications.

ML has existed as long as AI, although it has only been in recent years that it has flourished as the most successful subfield of AI. All ML algorithms need as much data as possible to produce useful outcomes. Data is the key. Therefore, greater computer power, together with the availability of very big storage systems, able to process and store large amounts of data, and fast and reliable network connections, have given rise to ML's recent success. The relevance of ML in AI today is so great that frequently when people say AI, they really mean ML, confusing the part with the whole.

So, what is ML in a nutshell? ML is the process of programming computers to optimise a performance criterion using example data or past experience. ML is used to create useful approximations for processes to solve tasks that we do not have algorithms for, but that we do have relevant data to learn patterns from

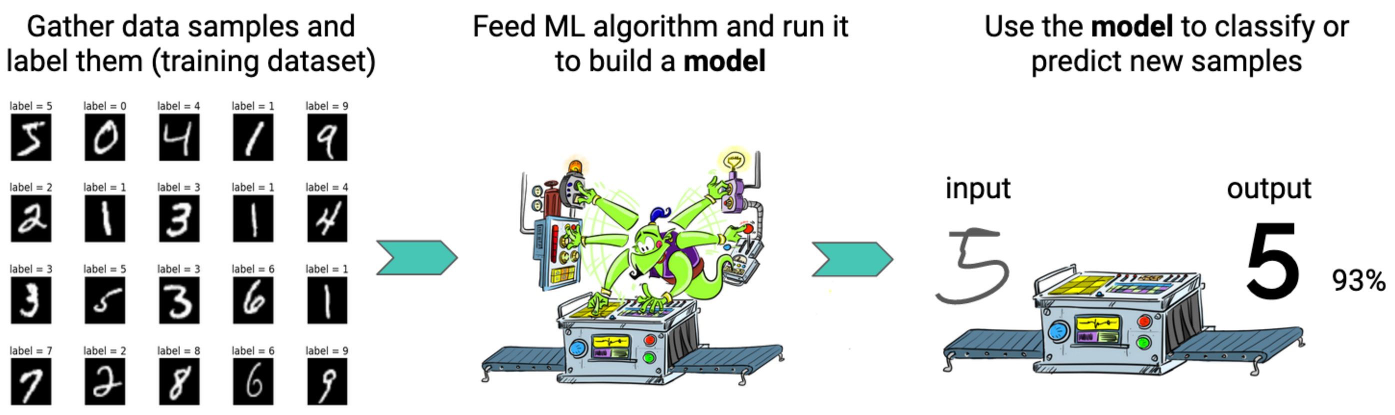


Figure 1. Supervised machine learning steps to build a ML model

(Alpaydin, 2020). The term learning is a metaphor which expresses the fact that the more data is fed in as an input to the algorithms, the more accurate their outcomes are.

Each of the numerous algorithms that make up the ML family belong to one of the following categories: supervised learning, unsupervised learning, and reinforcement learning. In all cases, the goal of ML is to build a model capable of classifying, predicting, or recognising things from a collection of data. In supervised learning, a dataset of known examples is manually labeled to build a model with which unlabeled data, similar to but different from those used in the training data set, can be recognised. In unsupervised learning, the built models are able to extract some patterns from a set of unlabeled data. Finally, in reinforcement learning, models are built by testing possible solutions; those that maximise some reward function are maintained while those that score badly according to that function are eliminated.

As an example, Figure 1 shows the steps needed to build a model aimed at recognising handwritten numbers by means of supervised ML. The model, represented by a machine, is being built by a ML algorithm, represented by a genie, which analyses the training dataset to iteratively improve the model. When the ML algorithm is finished, an independent model capable of recognising new handwritten numbers is ready to be used in a software application.

Artificial intelligence education in K12

AI education in K12 is not new. The first efforts to make AI programming tools accessible to children took place in the early 1970s, with the Logo programming language (Solomon et al., 2020), and continued through the 1980s. However, AI education suffered a cold period from the 1990s until 2012, when educators, AI researchers, and the general public changed their view about AI due to the big success achieved by ML in solving problems such as image recognition, language translation, transcription of speech, game playing, and natural language processing (Kahn & Winters, 2020).

This vigorous rebirth of AI education becomes pertinent in Moreno-Guerrero et al. (2020), where a scientific mapping of 379 publications, dated from the birth of AI in 1956 to the present was carried out to analyse the importance and the high profile that AI has acquired in the scientific literature in the Web of Science categories related to the field of education.

Why this growing interest in teaching AI at school? AI has erupted in society, creating new applications and possibilities while also introducing some ethical problems. Whether they are conscious of it or not, children use software applications based on AI on a daily basis: product recommendation systems, predictive writing, face recognition, and many more. However, few people understand how these technologies work, yet as argued earlier, this is a must if we want to educate conscientious and critical citizens of the future.

Therefore, governments around the world, worried about the benefits and risks AI poses, are developing policies, strategic plans, and other initiatives around this subject. Some policy foresight reports suggest that in the coming years AI will change learning, teaching, and education. The speed of technological change will be very fast, and it will create pressure to transform educational practices, institutions, and policies (Pedro, Subosa, Rivas, & Valverde, 2019; Tuomi, 2018). These reports also suggest that AI-related jobs are growing dramatically and hence, there is a rising demand for AI-literate workers.

Although programming and CT content has been incorporated into primary and secondary curricula in most developed countries, AI content is often not included, or is treated very superficially. CT is a cognitive ability while programming is an instrumental competence (Moreno-León, Robles, Román-Gonzalez, & Rodríguez-García, 2019). Indeed, programming, together with unplugged activities, is the activity widely used to develop CT. We propose that hands-on AI projects can also contribute to CT development (Rodríguez-García, León, González, & Robles, 2019). In fact, AI could add some new dimensions to the existing CT framework, as proposed in Van Brummelen, Shen, & Patton, (2019).

For instance, when gathering and labelling a dataset intended to build a model by means of supervised ML, students should strive to find a representative set of examples from which a good model is obtained when the ML algorithm is applied. This activity helps them to get a deeper insight into the problem being solved. For example, when a text recognition model on a given topic is to be built, a set of sentences with the necessary vocabulary for that topic must be chosen to build a sufficiently precise ML model. Furthermore, a classification task and an evaluation of the model has to be performed and, if the resulting model does not perform well enough, the training dataset must be improved by adding or removing some data. Therefore, the solution is found through iteration, which is a way to gain a progressive understanding of the problem that is being solved.

The LearningML platform

LearningML is an educational platform intended to teach ML fundamentals in an easy and enjoyable way (Rodríguez-García, Moreno-León, Román-González & Robles, 2020). It has been developed taking "low floor, high ceiling and wide walls" as the main design principle (Resnick et al, 2009, p. 63). That is, we have tried to build a tool which is very easy to get started with and allows users to get some results from the very beginning (low floor), but also allows students to build more complex projects over time (high ceiling), and is able to support different kinds of projects (wide walls).
Among these principles, the first — developing a tool that is very easy to use and get started with — was the most relevant for us. Hence,

only a standard web browser is needed to run LearningML. In addition, the use of any cloud AI service (such as Google AI or IBM Watson) has been avoided, since they require users to create an account and deal with API keys, which can be a very easy task for a software developer, but can be an obstacle for children and teachers. In addition, although there are free usage plans available for these cloud services, they are exposed to possible changes in their terms and conditions and that availability may change in the future. Therefore, all the complex ML algorithms have been built into the code of LearningML to run locally in the web browser. There is no need to register in order to start building ML models and coding applications, although some extra functionality, such as the option to save projects in the cloud and to share or reinvent shared works, can be accessed when you create a LearningML account.

Although our research has shown (Rodríguez-García, Moreno-León, Román-González & Robles, 2021) that the tool is very suitable for children between 10 and 16 years old, LearningML can also be helpful for undergraduate students and professionals who need to learn ML fundamentals because of the expansion of ML-based tools in their fields.

LearningML is composed of three elements: the website, the ML editor, and the programming editor.

The website

The website[8] is devoted to hosting the platform and offering content designed to help users learn how to use the tool, as well as learn about ML and AI. Guided activities, video tutorials, a manual, a curated list of resources about ML/AI, and a blog with ML/AI related news can be found on the website.

The ML editor

The ML editor[9] is the tool where the user can build ML models for image or text recognition. This tool demonstrates clearly how supervised ML works. The main screen is divided into three sections, one for each phase of supervised ML.

In the first section the user creates some buckets corresponding to the different classes of data that have to be recognised. Then a set of example images or text must be added to these buckets depending on the class they belong to. This process, where the user gathers and labels a dataset, is known as *training*.

Once the dataset has enough samples, the learning phase can be run. Since it is an educational application, a few examples in each bucket (>10) are sufficient to obtain a working model. In the learning phase a simple neural network is used as the ML algorithm for text recognition, while a pre-trained neural network, known as mobilenet (Howard et al., 2017), is used together with a simple neural network as ML algorithms for image recognition.

Finally, when the learning phase has finished, a ML model, able to recognise new text or images similar but different to those used in the training dataset, is available for evaluation. In the evaluation phase we can test if the model works and is able to recognise almost all the test data. If the model does not perform well enough, more data examples can be added to the training dataset and a new model can be built by running the learning phase once again. This iterative process can be repeated until a good model is obtained.

This software is released under the *GNU Affero General Public License*[10], a free software license allowing anyone to study, modify, or contribute to the project. An instance of the application is also accessible at no cost.

[8] https://learningml.org

[9] https://learningml.org/editor

[10] https://www.gnu.org/licenses/agpl-3.0.en.html

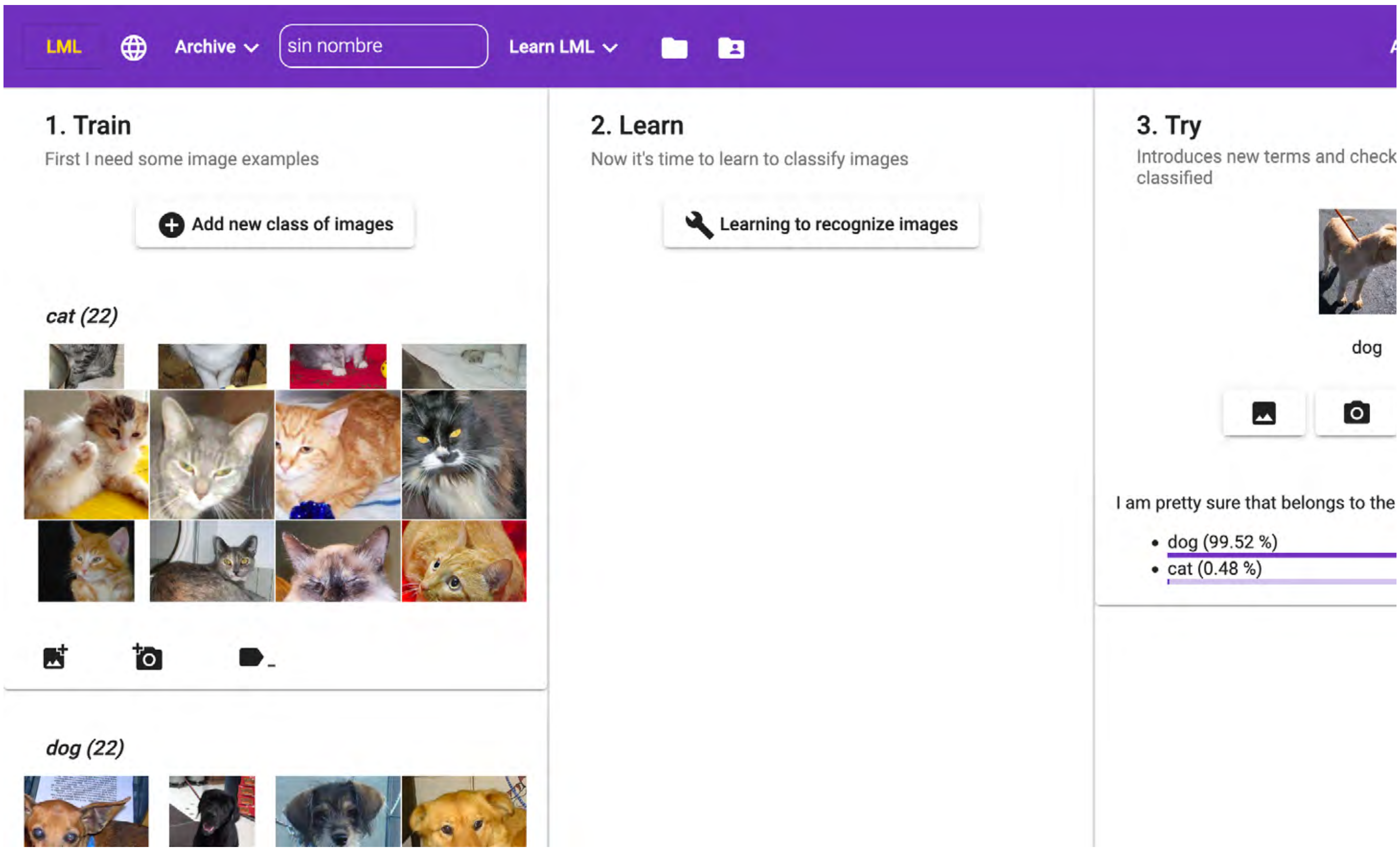

LML
Archive
sin nombre
Learn LML
1. Train
First I need some image examples
Add new class of images
cat (22)
dog (22)
2. Learn
Now it's time to learn to classify images
Learning to recognize images
3. Try
Introduces new terms and check
classified
dog
I am pretty sure that belongs to the
dog (99.52 %)
cat (0.48 %)

Figure 2. LearningML. The ML editor.

The programming editor

The programming editor[11] is a Scratch (Resnick et al., 2009) modification in which new blocks have been created which use the model built with the ML editor. Therefore, text and image recognition features can be added to Scratch programs. To achieve this, the following new blocks are available: a reporter able to classify text/images, a reporter that returns the confidence level of the classification given by the ML model, a stack block intended to add new image or text examples to the training dataset, and a stack block which allows the user to run the learning phase in order to build a new model.

We have been able to develop our ML extension of Scratch thanks to the free *Apache license 2.0*[12] under which this software is released, since it allows modification of the code under the conditions imposed by the license. Our ML Scratch fork has also been released under the same license.

Conclusion

LearningML is an educational platform aimed at teaching and learning AI fundamentals by doing. It is being developed to be as user-friendly as possible but, at the same time, to allow the creation of a wide range of AI-based applications, from the simplest to the most complete and complex applications. We aim to provide teachers and students with a

[11] https://learningml.org/scratch

[12] https://www.apache.org/licenses/LICENSE-2.0

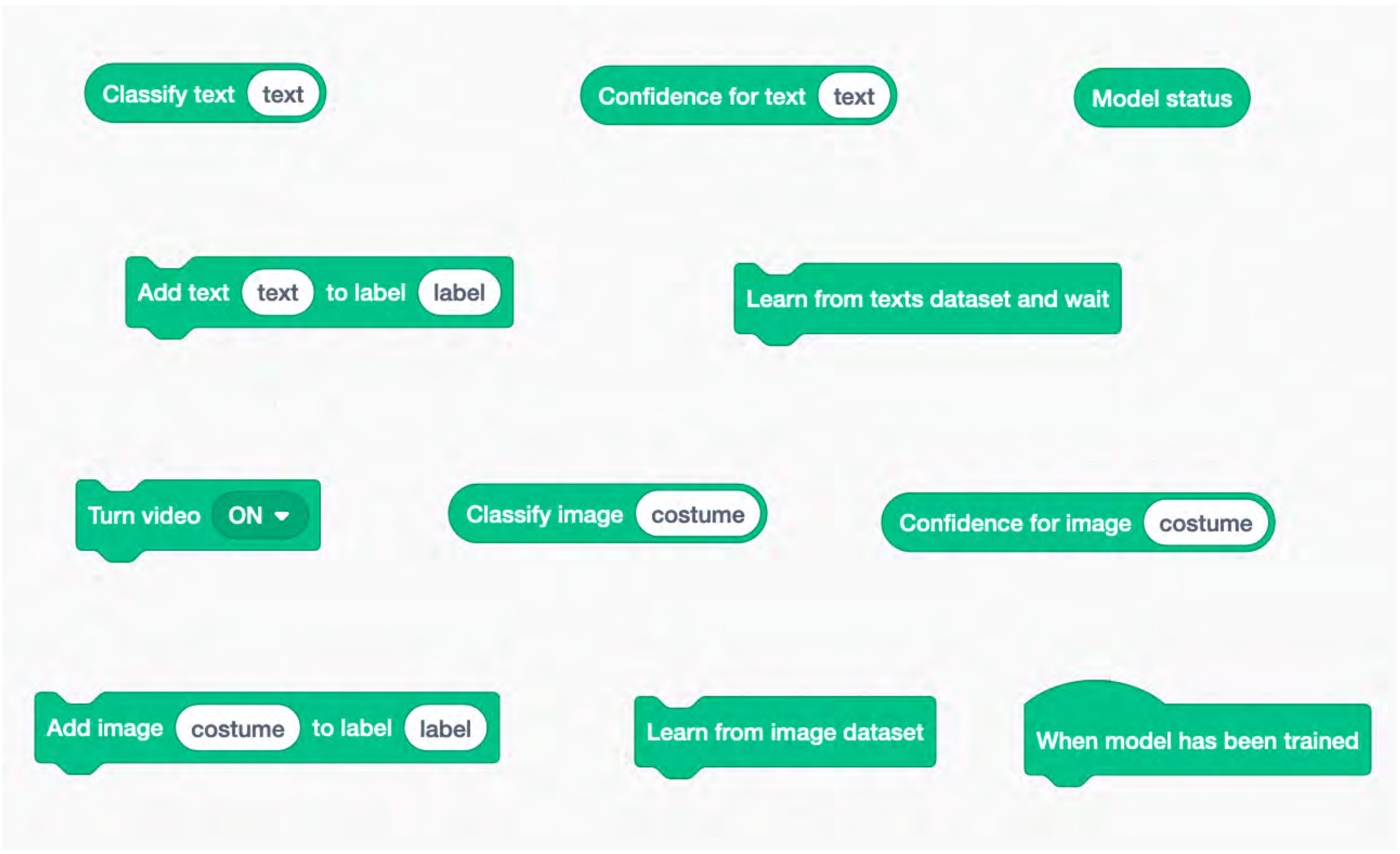


Figure 3. LearningML. ML Blocks added to Scratch in the programming editor.

powerful and engaging tool that helps them to develop CT skills by combining traditional programming activities with ML model building, to encourage the development of some new concepts, practices, and perspectives such as classification, training, and evaluating.

References

Alpaydin, E. (2020). *Introduction to machine learning*. MIT Press.

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* (Vol. 1, p. 25).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andretto, M., Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Kahn, K., & Winters, N. (2020). Constructionism and AI: A history and possible futures. Constructionism 2020, pp 238-243. (Available at: http://www.constructionismconf.org/wp-content/uploads/2020/05/C2020-Proceedings.pdf

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin, 39*(1), 223-227.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. *AI magazine, 27*(4), 12.

Moreno-Guerrero, A. J., López-Belmonte, J., Marín-Marín, J. A., & Soler-Costa, R. (2020). Scientific Development of Educational Artificial Intelligence in Web of Science. *Future Internet, 12*(8), 124.

Moreno-León, J., Robles, G., Román-González, M., & Rodríguez-García, J. D. (2019). No es lo mismo: Un análisis de red de texto sobre definiciones de pensamiento computacional para estudiar su relación con la programación informática: Not the same: a text network analysis on computational thinking definitions to study its relationship with computer programming. *Revista Interuniversitaria de Investigación En Tecnología Educativa, Nº 7*. https://doi.org/10.6018/riite.397151

Pedro, F., Subosa, M., Rivas, A., & Valverde, P. (2019). Artificial intelligence in education: Challenges and opportunities for sustainable development.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., & Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60. https://doi.org/10.1145/1592761.1592779

Rodríguez-García, J. D., León, J. M., González, M. R., & Robles, G. (2019, November). Developing computational thinking at school with machine learning: an exploration. In *2019 International Symposium on Computers in Education (SIIE)* (pp. 1-6). IEEE.

Rodríguez-García, J. D., Moreno-León, J., Román-González, M., & Robles, G. (2020). LearningML: A Tool to Foster Computational Thinking Skills Through Practical Artificial Intelligence Projects. *Revista de Educación a Distancia, 20*(63).

Rodríguez-García, J. D., Moreno-León, J., Román-González, M., & Robles, G. (2021). Evaluation of an Online Intervention to Teach Artificial Intelligence With LearningML to 10-16-Years-Old Students. In SIGCSE '21 ACM SIGCSE Technical Symposium, March 17–20, 2021, Toronto, Canada.ACM, New York, NY, USA. Preprint https://www.researchgate.net/publication/344744220_Evaluation_of_an_Online_Intervention_to_Teach_Artificial_Intelligence_With_LearningML_to_10-16-Year-Old_Students

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., Papert, A. & Silverman, B. (2020). History of Logo. *Proceedings of the ACM on Programming Languages, 4*(HOPL), 1-66.

Tuomi, I. (2018). The impact of artificial intelligence on learning, teaching, and education. *Luxembourg: Publications Office of the European Union*.

Turing, A. M. (2004). Computing machinery and intelligence (1950). The Essential Turing: The Ideas that Gave Birth to the Computer Age. Ed. B. Jack Copeland. Oxford: Oxford UP, 433-64.

Van Brummelen, J., Shen, J. H., & Patton, E. W. (2019, June). The Popstar, the Poet, and the Grinch: Relating Artificial Intelligence to the Computational Thinking Framework with Block-based Coding. In *Proceedings of International Conference on Computational Thinking Education* (pp. 160-161).

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

Section 3: Computing topics

Teaching programming with PRIMM: the importance of classroom talk

Sue Sentance (Raspberry Pi Foundation, UK)

Abstract

PRIMM is an approach to structuring programming lessons with a focus on working with extracts of code in depth to understand both structure and function and doing so in collaboration with peers, through dialogue. Previous research has shown that teaching using a PRIMM approach can improve learner outcomes. In this paper I introduce the PRIMM approach to structuring lessons and how it can impact on productive classroom talk. A qualitative study was conducted with 20 programming teachers in primary and secondary schools. Early findings indicated that in PRIMM lessons teachers' talk differs in quality and content at different stages of the lesson, and highlights the importance of students' use of programming vocabulary. A focus on language and talk could be a productive area of research in our quest to improve our understanding of effective teaching strategies for young novice programmers.

Introduction

PRIMM is an approach to teaching programming that came about because teachers who were new to teaching, or new to teaching programming, were expressing frustration that they could not effectively support young students who had difficulty with programming. Computing teachers benefit from access to proven teaching strategies and pedagogies relating to programming. Much research has been carried out in programming education, and only recently in schools, and this has not been widely translated into usable structures for teachers. Consequently, computing teachers are being called to deliver a challenging subject with insufficient knowledge of effective teaching strategies and on how to develop and enhance vital competencies to accomplish this task. To address these issues, I and my colleagues have developed and are evaluating a new pedagogical model for teaching and learning programming (PRIMM) (Sentance & Waite, 2017, Sentance, Waite and Kallia, 2019).

PRIMM stands for Predict, Run, Investigate, Modify, and Make. Using PRIMM, classroom activities can be designed that involve predicting the output of code, code comprehension, and gradually making new programs. It is a method of teaching programming that counters the known problem of novices trying to write programs before they are able to read them (Lister et al., 2004). It provides a staged and gradual approach to building an understanding of programming concepts alongside the development of confidence, with a focus on program comprehension over completed artefacts. It is an appropriate approach for young students where we need to minimise excessive cognitive load and helps teachers to engage each student when teaching large mixed-ability classes.

This paper focuses on one aspect that is a key feature of every PRIMM lesson: productive classroom talk. Despite a surge of interest in programming education in school in recent years, the use of talk and language has not been a particular focus, with little literature in computing education on this topic. Research in mathematics and science education around dialogue has increased our understanding of both the nature of productive classroom talk, and how teachers can encourage this in their classes. What is of interest here is how this work relates to the programming classroom, and whether talking together about programs can really support learning. In this paper I outline what PRIMM is, why language and talk is important to the learning of programming, and report on some of the findings from a recent study.

Teaching programming

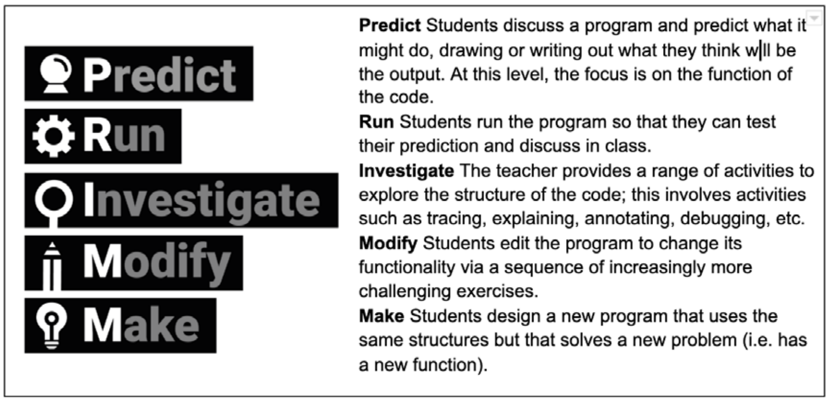
Novices can find programming difficult; research abounds on this topic. For example, it has been asserted that, beyond the syntax and semantics of particular programming concepts, novices may struggle to put these together to construct a program (Robins, Rountree, & Rountree, 2003); additionally, students have a surface knowledge of programming which is context specific and, thus, it is difficult to be applied in different contexts (Lahtinen, Ala-Mutka, & Järvinen, 2005). Actually writing code (as opposed to reading) is particularly hard for novice programmers (Denny et al., 2008; Qian & Lehman, 2017), and it is commonly believed that code tracing is easier than code writing (Denny et al., 2008). However, many students find code tracing challenging (Vainio & Sajaniemi, 2007) with particular difficulties being around single value tracing, confusion of function and structure, external representations, and levels of abstraction. The mental effort needed by learners as they embark on this complex journey of learning to program can also be viewed through cognitive load theory (van Merriënboer & Sweller, 2005). Cognitive load theory is a theory of instructional design that suggests that some instructional techniques assume a processing capacity greater than our limits and so are likely to be defective, and that students should instead engage in activities that are directed at schema acquisition and automation (Sweller, 1994). Working independently on programming has been suggested to have higher cognitive load than working collaboratively through pair programming (Tsai, Yang, & Chang, 2015). However, we may inadvertently use teaching methods which don't help this situation at all. A reliance on programming textbooks and "show me" approaches to teaching coding means that novices may end up being asked to copy in a section of code that has no meaning to them at all. Add this to the fact that younger learners will be developing their literacy and keyboard skills, the process of copying in can be incredibly frustrating and dispiriting. Another practice might be to model writing a program from the front while learners watch, and then ask learners to go ahead and write a similar program themselves: this leaves a huge chasm for the novice programmer to fill in themselves which many simply cannot manage.

What is PRIMM?

PRIMM stands for, Predict, Run, Investigate, Modify, and Make. It is based on the following five principles;

Principle 1: Read code before you write code. The excitement of writing a new program and creating something that works can mean we don't spend enough time at the beginning reading and learning from simple, well-written programs. PRIMM draws on tracing and reading code as an important principle for teaching programming (Lister et al., 2009). The predict phase of PRIMM encourages students to practise reading code and working out what it will do when executed.

Principle 2: Work collaboratively to talk about programs. Dialogue and classroom talk are an important aspect of teaching and learning.


Predict
Run
Investigate
Modify
Make

Predict Students discuss a program and predict what it might do, drawing or writing out what they think will be the output. At this level, the focus is on the function of the code.
Run Students run the program so that they can test their prediction and discuss in class.
Investigate The teacher provides a range of activities to explore the structure of the code; this involves activities such as tracing, explaining, annotating, debugging, etc.
Modify Students edit the program to change its functionality via a sequence of increasingly more challenging exercises.
Make Students design a new program that uses the same structures but that solves a new problem (i.e. has a new function).

Figure 1. The five stages of PRIMM

PRIMM particularly focuses on classroom discussion, specific questioning about code, use of vocabulary, and asking students to talk to each other about code. PRIMM draws on sociocultural theory which helps us to understand how language can support learning. Language can be seen as a central form of mediation that enables thinking and internalisation of concepts to take place (Vygotsky, 1962). In PRIMM lessons, students are encouraged to discuss with each other; a social construction of knowledge formed through collaborative, program-focused tasks.

Principle 3: Focus on code comprehension. Languages like Python (commonly used in schools in England) are often celebrated because you can write a program in a short number of lines. However, that usually means there are lots of concepts in one line. One way to unpack what the code is doing is to align comprehension exercises to the Block Model (Schulte et al., 2010; Cruz et al., 2019). The Block Model distinguishes between a novice programmer's understanding of the structural atomic detail of a program, the code, the functional goals of the program, and the problem (Schulte et al., 2010). Unpacking and focusing on understanding the code also reduces cognitive load on the learner (Sweller, 1994).

Principle 4: Use existing starter programs. Again drawing on sociocultural theory, learning can be seen as a transition from the social plane to the cognitive plane (Walqui, 2006; Sentance et al., 2019), through the use of 'starter' programs that students can work with before taking ownership themselves. A PRIMM lesson starts with an activity whereby learners examine some existing code and predict what it might do.The learner does not have responsibility for the code and does not suffer emotionally if the code has errors in. Learners can test their predictions by running the code.

Principle 5: Gradually take ownership of programs. Learners should move along a continuum from where they first use programs made by someone else to finally create their own programs. In this way, PRIMM has partly built on Use-Modify-Create (UMC) (Lee et al., 2011) to gradually transfer ownership of the program to the student. It supports the student's confidence as they are not burdened by the prospect of failure until they understand how the program works.

PRIMM provides a structure for one of a series of lessons, with the intention that teachers can develop their own PRIMM-like materials at an appropriate level for their students (Figure 1).

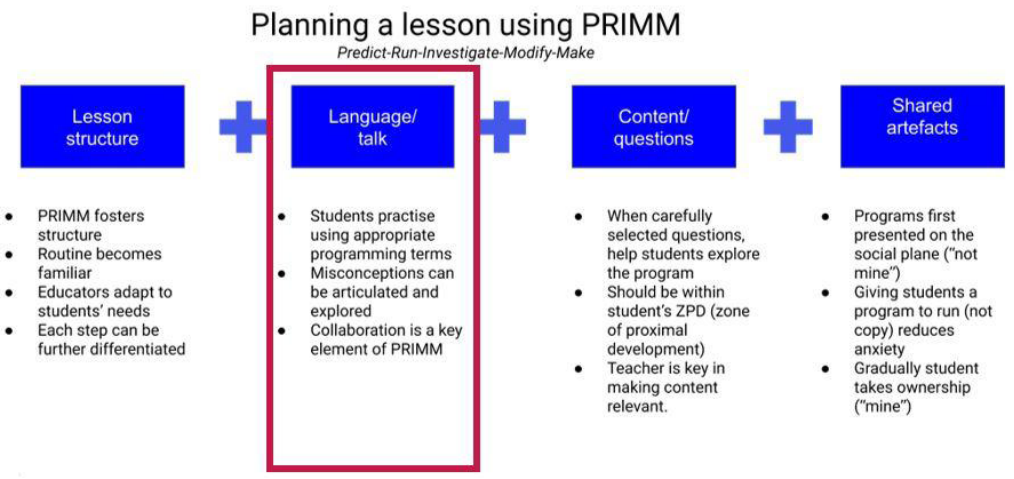

Planning a lesson using PRIMM
Predict-Run-Investigate-Modify-Make
Lesson structure
Language/ talk
Content/ questions
Shared artefacts
PRIMM fosters structure
Routine becomes familiar
Educators adapt to students' needs
Each step can be further differentiated
Students practise using appropriate programming terms
Misconceptions can be articulated and explored
Collaboration is a key element of PRIMM
When carefully selected questions, help students explore the program
Should be within student's ZPD (zone of proximal development)
Teacher is key in making content relevant.
Programs first presented on the social plane ("not mine")
Giving students a program to run (not copy) reduces anxiety
Gradually student takes ownership ("mine")

Figure 2. Planning a PRIMM lesson

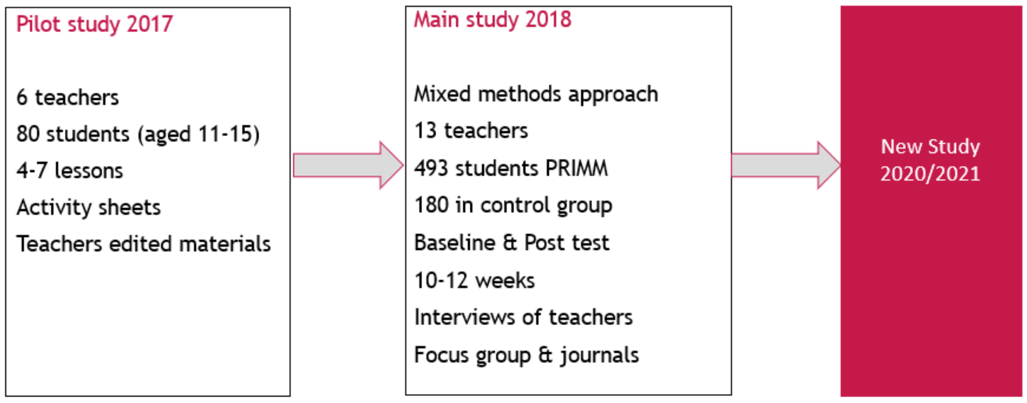


Figure 3. Research on the effectiveness of PRIMM approach

In terms of planning a PRIMM lesson, teachers will consider not only the structure of the lesson (as described in Figure 1), but also the opportunity for language and talk, the content and level of questioning, and the shared artefacts that are used in the lesson. These elements of planning are shown in Figure 2. This paper focuses on the language and talk that takes place in a PRIMM lesson.

PRIMM and learning outcomes

A number of studies have been employed to investigate the impact of PRIMM (see Figure 3). To date the largest of these was a mixed methods study conducted in 2018 involving around 500 students aged 11 to 14.

In this study, a type of quasi-experimental design known as the non-equivalent control group post-test design (Campbell & Stanley, 1963) was used to investigate the impact of a series of PRIMM-structured lessons on learner outcomes. Following this methodology, the treatment, or experimental, group were classes being taught using PRIMM materials provided by the researchers, with the control group consisting of students who were to take the same number of programming lessons, covering the same topics, but using the teaching method normally used in the school. To ensure that students did not differ significantly in their computer programming attainment, both groups were baseline tested before the start of the intervention. Teachers were given full sets of materials, including starter tasks, presentations, worksheets, starter programs, and answers, for ten lessons (including extension material) covering the basic programming constructs of sequence, selection, and iteration in Python. Teachers then delivered programming lessons using the PRIMM approach for 8 to 12 weeks. Data was collected via a combination of a baseline test, a post test to compare control and experimental groups, and teacher interviews.

The post-test score of the experimental group was compared with that of the control group. Differences between the control and experimental groups after the programming lessons were examined to see if the PRIMM lessons had had an impact on programming attainment. The results showed a statistically significant difference in the score between the control and experimental groups for all students in favour of the experimental group (see Sentance et al. (2019) for further details).

The quantitative results were further supported by the qualitative data. From interviews with nine participating teachers the research found that teachers particularly value the collaborative approach taken in PRIMM, the structure given to lessons, and the way that resources can be differentiated. This led to the assertion that PRIMM is an approach in school classrooms to improve learner outcomes in programming (Sentance et al., 2019).

PRIMM and classroom talk

In this paper I am focusing on a specific aspect of PRIMM, the role of language. According to Vygotsky, social interaction plays a critical role in children's learning (Vygotsky, 1978). Mediated activity promotes higher mental processes in three major forms of mediation: material tools, psychological tools (including language), and interaction with other human beings.

Classroom talk

Classrooms are full of talk — instructions, questions, explanations, as well as student–student and student–teacher dialogue. Teachers have an impact on the quality of the dialogue in their classroom and are an important model for pupils' use of language for reasoning (Mercer & Sams, 2006).

A range of models have been proposed to describe effective dialogue in the classroom. Dialogically organised instruction (Nystrand et al., 2003) sets out three ways the teacher can promote effective dialogue: through uptake (incorporating student ideas into subsequent questions of other students), through authentic questioning (used to explore views not test knowledge), and through high-level evaluation (where the teacher incorporates the response into elaborative comments).

This demonstrates that questioning is a key part of establishing effective dialogue, but teachers may limit their questions to the Initiation-Response-Feedback (IRF) style (Sinclair & Coulthard, 1975) to elicit answers from students where the answers to the questions are already known. Although a valid component of some lessons, these types of questions have been criticised for inhibiting classroom talk and the development of ideas (Dawes, 2004; Wilkinson, 2013). A more dialogic approach focuses on open, exploratory questions.

Mercer and colleagues developed the idea of exploratory talk (Mercer, 1995), in which partners engage critically but constructively with each other's ideas. To measure the impact of exploratory talk, a series of research projects were conducted under the banner of Thinking Together. The research involved interventions that gave both teachers and students new skills in using language for reasoning. In the context of mathematics, this was shown to enable them to use language more effectively as a tool for working on maths problems together.

A recent study found that improving the quality of children's use of language for reasoning together improves their learning and understanding of mathematics (Mercer & Sams, 2006). Another study found that three aspects of teacher–student dialogue strongly predicted the performance of pupils aged 10 to 11 in standardised assessments: elaboration (building on contributions), querying (challenging a contribution) , and student participation (Howe et al., 2019).

In computing education, most of the literature relating to language and communication as a vehicle for learning centres on pair programming and peer instruction (Vahrenhold et al., 2019), both privileging classroom talk and purposeful dialogue. Research has shown that peer instruction positively impacts learning outcomes (Porter et al., 2011; Zingaro et al., 2014). Pair programming has been shown to improve program quality and confidence (Braught et al., 2008; McDowell et al., 2006), but in the school context it may depend on the way that the collaborative work is instantiated (Lewis, 2011.) An in-depth study of six pairs of 5th grade students in the context of pair programming revealed specific dialogue strategies used by students such as 'Let me help you' or 'Make suggestion' (Tsan et al., 2018). Another study which looked at interaction mechanisms in computing students' talk identified collaborative problem solving, conversations expressing excitement, and more social conversations (Israel et al., 2017). I am not aware of studies in programming education in school that specifically focus on dialogue and programming vocabulary.

Diethelm and Goschler (2015) highlight the lack of attention to computing-specific vocabulary and consider that specific items of computing vocabulary may be ambiguous or have different meanings in everyday life from their scientific meaning. They suggest a need for a meta-discourse around language such that pupils in school can learn to distinguish between everyday and scientific meanings of terms and that teachers should be more deliberate about vocabulary (Diethelm et al., 2018). There is clearly scope for more detailed investigation into how young learners acquire and use the technical vocabulary in programming.

The current study

In a PRIMM lesson, the intention is that a teacher facilitates productive classroom talk — encouraging discussion, modelling vocabulary use, asking in-depth questions. Having a common language to talk about programming constructs is important. Talking about a program and how it works helps learners to find the right vocabulary to use to articulate their understanding. Actually verbalising out loud the steps of a program that is difficult to understand can help learners to focus on atomic or smaller elements at a time. The analysis of data in the 2018 study inspired a new phase in research around PRIMM specifically focusing on the use of talk in the classroom and how it could support a deep understanding of programming constructs.

In the current study I am focusing specifically on classroom talk in programming lessons in the context of PRIMM, seeking to investigate the quantity, quality, and content of classroom talk in programming lessons and teachers' perceptions of the impact of PRIMM on classroom talk. This work is in progress.

In the first phase of the study, I conducted interviews with 20 teachers who have been using PRIMM for different amounts of time in their classrooms. The findings are obviously impacted by the fact that much of the teaching in the last six months has been either remote or under varying degrees of social distancing in the classroom. Teachers were asked a number of questions around the following topics:

- The types of talk that take place in programming lessons
- The impact, if any, of PRIMM on the quantity and quality of talk in programming lessons
- Teachers' experience of students' use of programming terminology and vocabulary
- Approaches teachers use to foster discussion amongst students

To ensure that the study aligned to ethical guidelines (BERA, 2018) participants gave consent to the use of their data for specific purposes and full information was given. After transcription, participants were able to check their interview transcripts.

Early findings

The data was transcribed and analysed using thematic analysis (Braun & Clarke, 2006). The interviews were coded through an iterative and inductive process of coding, merging, and refining codes and re-coding (Nowell et al., 2016; Braun & Clarke, 2006).

There were some initial findings relating to the impact of PRIMM on classroom talk. Many teachers referred to the difference between 'pre-PRIMM' teaching and using the PRIMM approach. They commented that in PRIMM lessons there was less whole-class talk by the teacher, enhanced student-student dialogue, and that there was an increased focus on programming vocabulary.

Less talk by the teacher

One teacher, Teacher O[13], had found that when he initially taught programming he found that the approaches he was using were ineffective for his lower secondary school students, who were struggling. Since using PRIMM, he talks less now from the front of the class at the beginning of the lesson and gives students tasks to do that focus on the content of the code:

"In non-PRIMM lessons, I'm more talking about fundamentals and just talking through some real basics, like how to use a particular statement, and I'm talking to a whole group and then I find myself repeating myself going around the whole group. With PRIMM lessons, I'm getting kids to get onto the work and then I'm able to talk at a much higher level about what's going on in those particular programs." (Teacher O, secondary)

Both secondary and primary teachers noted the difference in the amount and nature of the whole-class talk:

"So I guess it lessens the me standing and talking at the front of the classroom because traditionally before this approach I probably would have put the code up on the board and then talked through it block by block and said, this is going to do this and this is going to do that, and so on and so forth, whereas it throws it out [and] it gets them in the driving seat straightaway..." (Teacher N, primary)

Student-student dialogue

Other teachers could specifically see the impact of the PRIMM approach in facilitating a more

[13] The 20 teachers in the study are referred to as Teacher A through to Teacher T.

questioning approach amongst students:

"And they'll go and say, but how did that work, why does that work, why is mine not doing that? And I think that PRIMM scaffolds that and allows them to have those discussions. Whereas, before, even with differentiation, they just could either do it or they couldn't do it." (Teacher C, secondary)

Several teachers highlighted the impact of verbalising on pupils' understanding. This aligns with research indicating that peer interaction improves learning:

"They are more engaged in the code itself and talking more about the code itself and what it does and that use of language definitely does aid their understanding." (Teacher L, primary)

Students' use of programming vocabulary

In the way that teachers discussed the use of programming-specific terms there was an indication that the use of PRIMM facilitated a more confident use of programming vocabulary:

"But what I have found is moving to PRIMM is the language the students are using is more improved because they know… Well, what's the variable? There's the variable… That is embedded over a period of time as well. " (Teacher G, secondary)

Other teachers were able to articulate why they thought it was important to use talk to verbalise how a program works, in that it gives learners a language with which they can express their understanding and supports the creation of a mental model. Finally, a teacher reflects on the fact that the focus on function and structure of code was enabling them to ask more advanced questions of the class or of individuals:
"I'm talking at a more advanced level to the whole group, but for less time. When I'm asking questions, they're usually much more useful and probing questions… " (Teacher O, secondary)

This study is in its early stages and I plan to report on it more in full in future publications. There are also plans to corroborate indicative findings with more research into actual classroom dialogue. However at this stage it appears that teachers believe that the use of PRIMM to structure lessons, with the collaborative, investigative exercises, gives an opportunity for more, and potentially more productive, dialogue. Teachers across the data set reflected that they have found this way of working enhances vocabulary use and a higher level of conceptual understanding.

Conclusion

Ad hoc reports indicate that the use of PRIMM to structure programming lessons has been widely adopted across schools in England, and also further afield in Australia, USA, and Malaysia. Teachers are able to create their own PRIMM materials by reworking their existing programming lessons around the PRIMM structure, or they can use or adapt resources that are being developed and shared by resource creators, including through the free, government-funded Teach Computing Curriculum[14] in England, which uses PRIMM in many of the programming units of work.

PRIMM is certainly a popular approach but further research is needed to examine what specific elements of it make a difference to learner outcomes. Variations of PRIMM are emerging which adapt the structure in different ways, some with more emphasis on keywords at the beginning (KPRIDE[15]), and others with a stage for evaluation at the end (TIME[16]).

What PRIMM has achieved for many teachers is an opportunity to reflect on, and examine, the value of the different activities that they use in the programming classroom. As all teachers know, it is being a reflective practitioner, and trying out different strategies, that improves teaching over time. To this extent it doesn't

[14] http://teachcomputing.org/curriculum

[15] https://blog.withcode.uk/2019/06/k-pride-tips-for-teaching-programming-so-everyone-can-make-progress/

[16] https://craigndave.org/programming-with-time/

really matter if every programming teacher uses a different acronym or variation on the theme, if they are able to reflect on the process of teaching and the impact on the individual students with whom they are working. Where PRIMM really comes into its own is to support new computing teachers, either new to teaching or new to computing, who are struggling with a class of young novice programmers, with varying levels of interest and engagement, where there is the potential for all the children to be "stuck" at exactly the same time and all be in need of teacher attention. The staged, gradual approach of PRIMM builds confidence and ownership of code one step at a time and focuses on understanding not completed artefacts.

In this paper, the particular focus has been on language due to the way that PRIMM promotes the practice, both by teachers and students, of talking out loud about what a program might do (function) and how it might do it (structure). The social and psychological functions of language are both drawn on to promote confidence as well as understanding, through talk and dialogue. An initial study into this aspect of PRIMM has shown some particular aspects of classroom talk that are facilitated by the PRIMM structure:

- Specific questioning about code leads to productive dialogue between students about programming code
- Teachers use whole class teaching differently at different stages of the PRIMM cycle
- Learning to use vocabulary to explain how a program works is challenging for students
- Teachers using PRIMM see part of their role as facilitating and focusing productive classroom talk

More research is needed on the way that classroom talk can support young novices learning programming, and beyond the context of PRIMM. It would be interesting to investigate whether an intervention based on exploratory talk (Mercer, 1995) would improve learning outcomes in computing as it has done in other subjects.

References

British Educational Research Association (BERA) (2018) *Ethical guidelines for educational research*. 4th edn. Available at: https://www.bera.ac.uk/researchers-resources/publications/ethical-guidelines-for-educational-research-2018 (Accessed: 20 December 2020).

Braught, G. Ebay, L. M. and Wahls, T. (2008). The effects of pair programming on individual programming skill. ACM SIGCSE Bulletin 40, 1(2008), 200–204.

Braun, V. and Clarke,V. (2006). Using thematic analysis in psychology. Qualitative research in psychology 3, 2 (2006), 77–101.

Campbell, D. T., and Stanley, J.C, Experimental and Quasi-Experimental Designs for Research on Teaching. In N. L. Gage (ed.), Handbook of Research on Teaching. Chicago: Rand McNally, 1963.

Dawes, L. (2004). Talk and Learning in classroom science. International Journal of Science Education 26, 6 (2004), 677–695.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth International Workshop on Computing Education Research*, 113-124.Diethelm, I., & Goschler, J. (2015). Questions on Spoken Language and Terminology for Teaching Computer Science. Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, 21–26.

Diethelm, I., Goschler, J., & Lampe, T. (2018). Language and Computing (In Sentance, S., Barendsen, E. and Schulte, C. Computer Science Education. Perspectives on Teaching and Learning in School. Bloomsbury Academic.. p. Chapter 15, 207-219).

Howe, C., Hennessy, S., Mercer, N., Vrikki, M., & Wheatley, L. (2019). Teacher–Student Dialogue During Classroom Teaching: Does It Really Impact on Student Outcomes? Journal of the Learning Sciences, 28(4–5), 462–512.

Israel, M., Wherfel, Q.M., Shehab, S., Melvin, O. & Lash, T. (2017). Describing Elementary Students' Interactions in K-5 Puzzle-based Computer Science Environments using the Collaborative Computing Observation Instrument (C-COI). In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17). Association for Computing Machinery*, New York, NY, USA, 110–117.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., ... & Weeda, R. (2019). Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 27-52).

Lahtinen, E., Ala-Mutka, K., J, H.-M., & rvinen. (2005). A study of the difficulties of novice programmers. Caparica, Portugal.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2(1), 32-37.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students?. *Computer Science Education*, 21(2), 105-134.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostr, J. E., Sanders, K., Sepp, O., l, Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. Leeds, United Kingdom, 119–150.

Lister, R., Fidge, C. & Teague, D. (2009). Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming. In Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09). ACM, New York, NY, USA, 161–165.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. 101–112.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90-95.

Mercer, N. (1995). The guided construction of knowledge: *Talk amongst teachers and learners*. Multilingual matters.

Mercer, N., & Sams, C. (2006). Teaching Children How to Use Language to Solve Maths Problems. Language and Education, 20(6), 507–528.

Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria. *International journal of qualitative methods, 16*(1), 1609406917733847.

Nystrand, M., Wu, L. L., Gamoran, A., Zeiser, S., & Long, D. A. (2003). Questions in Time: Investigating the Structure and Dynamics of Unfolding Classroom Discourse. Discourse Processes, 35(2), 135–198.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1-24.

Porter, L., Bailey Lee, C., Simon, B., & Zingaro, D. (2011, August). Peer instruction: Do students really learn from peer discussion in computing?. In *Proceedings of the seventh international workshop on Computing education research* (pp. 45-52).

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137–172.

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports* (pp. 65-86).

Sentance, S. and Waite, J. (2017). PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. In Proceedings of the 12th Workshop in Primary and Secondary Computing Education. ACM. https://dl.acm.org/doi/10.1145/3137065.3137084

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. Computer Science Education, 29(2–3), 136–176.

Sinclair, J.M. & Coulthard, M. (1975). Towards an analysis of discourse: The English used by teachers and pupils. Oxford Univ Press.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction, 4*(4), 295-312.

Tsai, C.-Y., Yang, Y.-F., & Chang, C.-K. (2015). Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language. *Educational Innovation through Technology (EITT), 2015 International Conference Of*, 143–146.Tsan, J., Lynch, C. F., & Boyer, K. E. (2018). "Alright, what do we need?": A study of young coders' collaborative dialogue. International Journal of Child-Computer Interaction, 17, 61–71

Vahrenhold, J., Cutts, Q. & Falkner, K. (2019). Schools (K–12). In The Cambridge Handbook of Computing Education Research, Fincher, S.A & Robins, A.V. (Eds.). Cambridge University Press, 547–583

Vainio, V., & Sajaniemi, J. (2007). Factors in Novice Programmers' Poor Tracing Skills. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 236–240.

van Merriënboer, J. J. G., & Sweller, J. (2005). Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review*, 17(2), 147–177. https://doi.org/10.1007/s10648-005-3951-0

Vygotsky, L.S. (1962). Thought and word. In Studies in communication. Thought and Language, Vygotsky,L.S., Hanfmann, £. & Vakar, G. (Eds.). MIT Press, 119–153.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes* Cambridge, Mass.: Harvard University Press.

Walqui, A. (2006). Scaffolding Instruction for English Language Learners: A Conceptual Framework. International Journal of Bilingual Education and Bilingualism, 9(2), 159–180. https://doi.org/10.1080/13670050608668639

Wilkinson, I. & Nelson, K. (2019). Role of Discussion in Reading Comprehension. In Hattie, J., Anderman, E.M. Visible Learning Guide to Student Achievement: Schools Edition. Routledge

Zingaro, D. (2014, March). Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM technical symposium on Computer Science Education* (pp. 373-378).

Raspberry Pi

www.raspberrypi.org @raspberrypi @raspberrypi @Raspberry_Pi raspberrypi