Fifteen Years of Introductory Programming in Schools: A Global Overview of K-12 Initiatives

Conference Paper · November 2019

DOI: 10.1145/3364510.3364513

CITATIONS
32

READS
560

6 authors, including:

Claudia Szabo
University of Adelaide
78 PUBLICATIONS 1,899 CITATIONS
SEE PROFILE

Judy Sheard
Monash University (Australia)
115 PUBLICATIONS 2,651 CITATIONS
SEE PROFILE

Andrew Luxton-Reilly
University of Auckland
137 PUBLICATIONS 3,414 CITATIONS
SEE PROFILE

Beth Simon
University of California, San Diego
209 PUBLICATIONS 6,411 CITATIONS
SEE PROFILE

Fifteen Years of Introductory Programming in Schools: A Global Overview of K–12 Initiatives

Claudia Szabo
University of Adelaide
Australia
claudia.szabo@adelaide.edu.au

Judy Sheard
Monash University
Australia
judy.sheard@monash.edu

Andrew Luxton-Reilly
University of Auckland
New Zealand
andrew@cs.auckland.ac.nz

Simon
University of Newcastle
Australia
simon@newcastle.edu.au

Brett A. Becker
University College Dublin
Ireland
brett.becker@ucd.ie

Linda Ott
Michigan Technological University
United States of America
linda@mtu.edu

ABSTRACT

Computing education and outreach in the K–12 school sector have shown significant growth over recent decades, resulting in a large body of literature focused on the teaching and learning of computing. Despite this extensive literature, we are not aware of global overviews on teaching and learning *programming* as opposed to computing or computational thinking in K–12. We conducted a systematic review of the literature on introductory programming from 2003 to 2017. In this paper we review the papers that were set in the K–12 context with the aim of exploring developments that have been made in teaching introductory K–12 programming during this period. These include new programming languages, tools, teaching methods, and outreach programs. The impetus for these innovations was often a desire to provide interesting and engaging learning experiences and to ensure an appropriate level of instruction for a particular age group. Many initiatives were driven by changes to national curricula to mandate the teaching of programming. We find that there is a need for long-term studies to identify the most effective pedagogical approaches. We also identify a major need faced by many countries for training and resources to support teachers through the curriculum changes.

CCS CONCEPTS

• **Social and professional topics → Computing education**;

KEYWORDS

K12; K–12; introductory programming; novice programming; systematic literature review; review; school

ACM Reference Format:
Claudia Szabo, Judy Sheard, Andrew Luxton-Reilly, Simon, Brett A. Becker, and Linda Ott. 2019. Fifteen Years of Introductory Programming in Schools: A Global Overview of K–12 Initiatives. In *19th Koli Calling International Conference on Computing Education Research (Koli Calling '19), November 21–24, 2019, Koli, Finland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3364510.3364513

1 INTRODUCTION

K–12 computing education is a broad topic that has received extensive research interest in the last years [4, 14, 16, 22, 58], with many governments across the world requiring the teaching of computing from the beginning of formal schooling. For example, in England [9] children learn computing from kindergarten, in Australia [3] the digital technologies curriculum starts in the first year of school, and in Finland [20, 43] computing is integrated with other subjects throughout the curriculum from grade 1.

Research has found a variety of terms used to refer to K–12 computing education internationally, including computing, computational thinking, digital technologies, and introductory programming. In addition, there are a variety of curricula. For example, in Australia [3], the Digital Technologies curriculum is focused on the design and evaluation of digital solutions, the use of computational thinking and the key concepts of abstraction, data management, algorithms, and the application of systems thinking among others. While there are many definitions of computational thinking in the literature [9, 12, 53, 76, 77], a consesus is that computational thinking is a thought process that utilizes the elements of abstraction, generalization, decomposition, algorithmic thinking, and debugging [73].

Despite a plethora of work focusing on the definition and implementation of a K–12 curriculum that covers computational thinking, computing, or digital technologies more broadly [1, 6, 22, 57], few works look at introductory programming courses or modules within K–12. The content and delivery of K–12 introductory programming courses influences K–12 students' views of computing as a future study area and career, and also potentially affects what is taught in introductory programming courses at the tertiary level. Teaching students to program is a complex process, with many studies providing evidence that learning to program is difficult [59]. However, recent studies have found that dropout rates among tertiary computing students are not alarmingly high [5, 72], and it has been suggested that the difficulties faced by novices may be a consequence of unrealistic expectations rather than intrinsic subject complexity [42], thus suggesting the need to teach programming in earlier, pre-tertiary years.

Designing and delivering introductory programming courses is challenging, especially when the audience comprises middle- and primary-school children with limited or no literacy skills. The challenge is exacerbated by a lack of identity for computing as a discipline in the primary and secondary school curricula [13, 69] and the fact that computing modules can be interdisciplinary, taught by a variety of teachers, and spread throughout or otherwise embedded in the curriculum [11]. In addition, teachers may not have the necessary knowledge or access to appropriate training and professional development [11]. A further problem is that the schools may have limited infrastructure [13].

In 2018 we performed a systematic literature review of introductory programming education papers published from 2003-2017 [2]. One of our classifications of the identified papers was based on the target audience of the introductory programming course or activity, separating the papers into tertiary education, K–12, and informal courses. Our earlier report [2] focuses on tertiary education. In this paper we focus on introductory programming in K–12 education. We present an analysis of 72 papers that cover K–12 activities relating to the teaching and learning of programming and meet our selection criteria. While we have analysed 72 papers, in our reference list we include only those that we have explicitly mentioned in this paper.

2 RELATED WORK

Despite a large corpus of publications on the topic of K–12 computing over the past fifteen years, to the best of our knowledge there have been no reviews of the literature of K–12 introductory programming. A 2019 study by Rich et al. [58] snowball surveyed over 300 K–8 teachers across 23 countries about their classroom experiences in teaching coding. The study showed that Scratch was by far the most popular programming language used and that computing was taught integrated mostly with mathematics and sciences. The study also showed more than 55% of teachers had no or very little training with computing or coding prior to deciding to teach it in their classrooms and that they had little experience (average four years) at the time of the of the survey.

A systematic review by Garneli et al. [14] covers the literature from 2009 to 2013 on the broader topic of computing education in K–12, focusing on instructional methods, contexts, and tools for teaching programming. Grover and Pea [15] present a review of computational thinking in K–12 and highlight the need for teaching computational thinking concepts as a prerequisite for teaching programming in K–12. Benitti's systematic literature review [4] analyses how robotics are used in K–12 teaching, identifying ten papers published between 2000 and 2009. Of these, only three used robotics as a means to introduce students to programming, and all three had positive effects on student engagement and learning of programming concepts.

Menekse [46] presents an overview of computing teacher training programs in the USA, reviewing papers published on the topic from 2004 to 2014. The analysis identifies that the majority of programs focused on high-school teachers, and particularly on training in-service teachers to incorporate computational thinking into their courses, to broaden their computing knowledge, and to provide them with tools for increasing diversity and accessibility to computing education. An earlier study [41] discusses programs from 11 universities. The analysis shows that the programs were between three days and three weeks long, similar to the findings of Menekse [46], and that they focused mostly on introducing various computing environments and tools to teachers and on empowering them to follow existing lesson plans and create new ones.

The Computer Science Education Research repository, which is a curated collection of research papers specifically on the topic of K–12 computing education from 2012-2017 [45], reveals only 17 papers when searched for introductory programming / novice programming, and none of those 17 are reviews or surveys.

3 METHODOLOGY

We review papers published between 2003 and 2017 inclusive. In selecting papers for review, we make a clear distinction between papers involving introductory programming in K–12 and those on other aspects of introductory computing, such as computational thinking, except where they intersect with introductory programming. We aim to answer the following research questions:

RQ1 What developments have been made in K–12 introductory programming education between 2003 and 2017?
RQ2 What evidence has been reported for different aspects of K–12 introductory programming education?

3.1 Data Sources

A 2018 ITiCSE Working Group [2] conducted a systematic review of introductory programming literature, based on the guidelines proposed by Kitchenham et al. [35]. The search term used was:

> "introductory programming" OR "introduction to programming" OR "novice programming" OR "novice programmers" OR "CS1" OR "CS 1" OR "learn programming" OR "learning to program" OR "teach programming"

This search term was applied to titles, abstracts, and keywords in the ACM Full Text Collection, IEEE Xplore, ScienceDirect, SpringerLink and Scopus databases. The search, conducted on 27 May 2018, resulted in an initial candidate list of 5056 papers.

Examination of the papers by the researchers resulted in the elimination of papers that were not in English, were less than four pages long, or did not explicitly discuss and evaluate introductory programming courses or initiatives. After this step, 1844 papers remained. Each paper was then classified based on when introductory programming was taught, resulting in 108 papers being classified as relating to introductory programming in the K–12 sector. These were not included in the initial report, which focused on papers situated in the tertiary level; instead they form the initial data set for this paper. These papers were checked against the Computer Science Education Research repository, a curated collection of research papers specifically on the topic of K–12 education from 2012-2017 [45]. A search in that repository for 'introductory' or 'novice programming' reveals 17 papers, all of which are part of our set.

3.2 Paper Classification

Once the 108 papers were identified, the team collectively defined a set of topics that they might cover. Through reading the papers and brainstorming, the list of topics was refined, grouped, and rationalised. The process resulted in four high-level groups: *curriculum*, *programming languages and environments*, *students*, and *teachers*, with multiple classifications possible.

The 108 papers were then divided among the authors for classification. During this phase 30 further papers were excluded because they did not match the selection criteria, and eight because they were inaccessible to the research team. This resulted in 72 papers, which form the primary set of papers for this study. After an initial calibration round, where a set of ten papers was read and classified by all authors until a satisfactory inter-rater reliability was achieved, all 72 papers were classified into at least one, and sometimes several, of the groups defined above. Additional information, including the age range of students, country in which the initiative/study took place, and funding source, was also collected.

4 DESCRIPTIVE STATISTICS

We found an even spread of papers across the three main year groups of K–6, 7–10 and 11–12 (Table 1). Most initiatives took place in the USA, followed by Greece (Table 2). Most reported initiatives (63%) were organised by single institutions. The most frequent funding source was the university of the initiative organisers (64%), with the other sources being government (28%), companies (4%), and schools (1%).

Table 1: Number of papers for each age range (n = 72); note that there is some overlap in the ranges

Student age range	Papers
All groups	5
K–6	20
Years 7–10	20
Years 7–12	2
Years 11–12	17
N/A	8

Bearing in mind that a paper can be classified into more than one group, we found that 50 (69%) papers deal with *students*, 46 (63%) with *programming languages and environments*, and 19 (26%) with *teachers*, while only eight (11%) deal explicitly with *curriculum*. In addition, we found that 30 (41%) papers presented one-off initiatives, while 24 (33%) of papers presented an overview of a program.

Figure 1 shows the frequency of publications about introductory programming in the K–12 context across the period of our review (no papers from the set are from 2003-2005). As can be seen, there is a growing interest in this space. Notwithstanding publication bias, this could be a natural consequence of the maturity of K–12 general computing outreach activities, but also of several initiatives such as national CS K–12 curricula and CS4All-type initiatives.

5 CURRICULUM

A key concern in recent years is the formal incorporation of computing into K–12 curricula. We identified several papers about broad initiatives to design, develop, and incorporate programming components into school curricula. The most extensive programs have been undertaken in countries whose K–12 curricula include a mandatory computing and programming component. In some cases this amounts to a revision of already existing K–12 computing standards. An early paper describing such a change is by Blonskis and Dagienė [7]. In recognition of the importance of programming to the Lithuanian economy, the exams given to high-school students were revised to include programming questions in the hope of fostering interest in the subject.

Table 2: Country of initiative (n = 72); note that some papers covered more than one country

Country of initiative	Papers
USA	18
Greece	11
Finland	5
Germany, Japan, Lithuania, Sweden, Switzerland, UK	3 each
China, Hong Kong, Slovenia, South Africa, Taiwan	2 each
Argentina, Australia, Belgium, Bulgaria, Canada, Czech Republic, Israel, Korea, Malaysia, Mexico, New Zealand, Oman, Slovak Republic, South Korea, Spain, Poland	1 each

In Slovenia, high-school computing course materials were developed by master teachers in a workshop. Brodnik et al. [10] discuss the creation of a community of practice for the teachers who are teaching the new curriculum. At the lower levels, Heintz et al. [21, 22] discuss computing curriculum revisions in Sweden using an integrated approach which introduces computing through already required subject areas. The new program was approved by the government in March 2017 to be implemented in 2018.

Not all countries have a nationally mandated curriculum. In Switzerland, for example, school curriculum is the responsibility of each canton, but a continuing discussion on common curricula for primary and lower secondary schools does address the inclusion of computing as a mandatory subject. Serafini [68] discusses the introduction of computational thinking into Swiss primary schools. This early work is further developed by Hromkovič et al. [26], who provide a comprehensive discussion of educational goals, curriculum support and their experience using Python and Logo for primary and high-school students. They also discuss their efforts to support the incorporation of computing into K–12 teacher training.

In the United States, another country in which educational standards are the purview of individual states, a number of programs are under way to incorporate programming into K–12 education. Xu et al. [79] discuss the incorporation of Creative Computation into computing courses in two very different high schools, taught by teachers with quite divergent pedagogical approaches. Funding from the National Science Foundation is providing support for some teacher training and the development of materials.

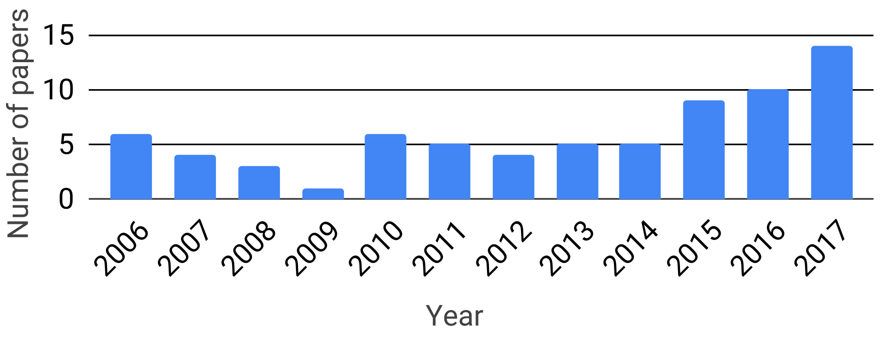

Figure 1: Number of papers by year of publication

6 PROGRAMMING LANGUAGES AND ENVIRONMENTS

A variety of programming languages and environments are used to deliver introductory programming in schools. We categorise these according to their *modality* (the method by which the user creates the program — as visual blocks or text), and their *intended output* (the contextualised output of the programming tasks — 2D graphics, 3D graphics, text, tangible objects, or simulated environments).

6.1 Visual

Most of the visual modality papers in our study focus on students in middle school and early high school (aged approximately 10–15), and some environments, such as App Inventor for Android, are more frequently associated with high-school programming. Table 3 summarises the papers on visual environments, their output types, and the school contexts in which they have been reported. Of all programming environments used, Scratch and variants was the most prominent [38, 47, 49, 52, 55, 56, 62, 70].

Table 3: Distribution of papers on languages and environments in the visual modality and their output types

Output	Primary school	Middle school	High school
2D Graphics			
- LaPlaya	[24]		
- Pilas Bloques		[63]	
- Scratch		[47, 62]	[49, 52, 70]
- Snap!	[30]	[30]	[74, 75]
3D Graphics			
- Alice			[19]
- Storytelling Alice		[34]	
- LookingGlass		[17, 18, 25, 28]	
- Scratch4SL			[56]
Tangibles			
- Lego NXT-G	[32]	[32, 47]	[32]
- ModKit/Arduino		[47]	
- BrickLayer/Arduino		[39]	
Simulation			
- App Inventor			[49, 52, 61]
Kinesthetic			
- Dance	[51]		

6.1.1 *2D Graphics.*

Although most visual programming environments are targeted at a middle school audience, LaPlaya, based on Snap!, is notable for supporting students in the fourth to sixth grades [24]. The Snap! environment was also extended to add Speech and Image Recognition blocks for children in mid–late primary [30]. Unfortunately, while both initiatives have been piloted, neither had been used in a formal classroom situation at the time of reporting.

A block-based environment called Pilas Bloques was developed to support students 9–15 years old in Argentina who have limited interest or ability in mathematics and who are living in rural areas with poor internet access and computers with low specifications [63]. The environment needed to fit with the national curriculum while supporting students from impoverished backgrounds who had limited interest or ability in mathematical concepts [63].

A large number of studies focus on determining the most suitable programming environment. A comparison of Scratch and a text-based mini-language, Karel the Robot, used to introduce programming to seventh-grade students in Germany suggested that the environments were similar, but there were some indications that Scratch was more effective [62]. Comparisons between Scratch and App Inventor for teaching high-school students found that App Inventor was more effective [49, 52]. This provides support for the intuition that Scratch is better suited to younger students, while older students prefer a more sophisticated visual programming environment.

A comparison between block-based and text-based languages was conducted with students in grades 8–12 in a high-school programming course [74, 75]. The use of Snap! was compared with two modified versions of Snap! that allowed students to read or to read and edit the JavaScript corresponding to their created blocks. In the second half of the course, the students moved to a Java environment. Student performance appeared to be impacted by the modality, with indications that block-based environments helped students to understand programs with more complex structure.

6.1.2 3D Graphics. Alice was used in a small study in Oman with students in grades 9–11 who worked independently for a couple of hours to complete a task. The students enjoyed the process, but could not engage more creatively due to English comprehension challenges [19]. Storytelling Alice was found to be popular among girls aged 11–15 who developed an animated sequence with programmed interactions [34].

Looking Glass, another Alice variant, uses a block-based language to interact with a 3D storytelling environment. Students aged 10–15 found the environment engaging to use [17, 18], with its examples and suggestions identified as the most helpful resources [25, 28]. Exercises in the style of Parson's Puzzles were found to be more helpful than tutorials, although students saw value in both [17, 18].

A variant of Scratch called Scratch4SL, designed for programming visual objects within the SecondLife virtual world, was used in Greece to teach programming to students aged 14–15 who had previous unsuccessful experiences with Storytelling Alice [56]. The Second Life environment was found to facilitate collaboration, making it easy for students to work together outside class, and increased overall engagement.

6.1.3 Tangibles. A study compared the use of block-based languages by student aged 12–13 to generate equivalent programs in three different contexts (Lego Mindstorms robots using NXT-G, LilyPad wearables using ModKit, and standard Desktop PC using Scratch). Its findings suggest that the robot environment was more engaging than the other contexts [47].

A five-day workshop introduced middle-school students to programming using a graphical environment that controlled Lilypad Arduino components in order to create wearable computing garments. Although the workshop was interesting, students found it difficult to program the Arduino [39].

While wearable devices are being investigated, they appear to pose technical challenges that are not present in more mature software development environments. The Lego NXT-G environment seems to be the easiest environment to use, resulting in the highest levels of student engagement.

6.1.4 Simulation. App Inventor for Android (AIA), which allows development of applications for mobile devices using a visual language, was used in week-long high-school summer camps in the USA, with positive reception from the teachers delivering the camps [61]. A comparison between Scratch and AIA found that students aged 15–17 in Greece were more motivated by AIA than by Scratch [49], while another study found that students using AIA had higher performance than those using Scratch, and both had higher performance than a control group taught using Pascal [52]. The ability to create applications for mobile devices is highly motivating for many students and results in high levels of engagement that may explain greater levels of learning.

6.2 Text-based

In the text-based modality, environments at the primary-school level are focused on Logo, while a variety of languages, similar to those used in typical introductory programming courses at tertiary level, are used at the high-school level. Table 4 summarizes text-based environments, output, and school contexts in which they have been reported.

Table 4: Distribution of papers on languages and environments in the text-based modality and their output types

Output	Primary school	Middle school	High school
2D Graphics			
- Logo	[27, 40, 68]	[26]	
- Java			[48]
- Jypeli		[29]	
- Python			[26]
- Squeak Smalltalk		[60]	[60]
Text			
- Python			[37, 44]
- Pascal			[7]
Tangibles			
- Java			[31, 78]

6.2.1 2D Graphics. The Logo environment is used to introduce students to text-based programming using an engaging visual output, and has been selected as the environment of choice in Swiss primary and middle schools. XLogo was used in Swiss primary schools to deliver introduction to programming workshops of 3–4 half-day blocks. The students seemed to enjoy programming and solution sharing and did not appear to struggle with the language syntax [68]. A variant was used to introduce programming concepts and develop algorithmic thinking skills in students aged 11–12 [26], and a simplified browser-based version with reduced cognitive load has been deployed for use in Swiss schools [27]. A 2007 Taiwanese study found that guided collaboration resulted in significantly better understanding than self-directed learning for sixth-grade students using MSWLogo over a 14-week course [40].

Sixth grade Taiwanese students used *MSWLogo* in a study comparing the performance of students who followed guided worksheets with those engaged in self-directed learning over a 14 week long course. Guided collaboration resulted in significantly better understanding than self-directed learning [40]. A more interactive alternative, Jypeli, was used in a week-long game programming course held at the University of Jyväskylä, Finland, for middle-school students. Results showed an improvement in the students' perception of programming [29].

When Squeak Smalltalk was used to teach programming to students aged 10–18, the younger children had difficulty following the instructions while the more mature students were better able to learn independently [60].

Python and Java were used in Swiss [26] and Bulgarian [48] high schools in limited contexts (turtle graphics and using pre-defined 2D graphics libraries respectively) before the introduction of programming concepts such as variables, functions, and lists.

6.2.2 Text. Python, Pascal, and Java are the three main text-based programming languages used. Despite concerns about its appeal to high-school audiences [37], Python is used in introductory high-school programming courses in Finland [44] and Sweden [26]. Pascal has been used to teach data structures and algorithms courses to senior high-school students (grades 11–12) in preparation for national paper- and computer-based exams [7]. The effectiveness of the language itself is studied by Kaila et al. [31], whose results show that Java used within the ViLLE environment resulted in improved performance in exams, suggesting that the visualisation helped students build more accurate mental models.

6.2.3 Tangibles. A study in Taiwan compared physical Lego Mindstorms robots and a simulated equivalent environment for teaching Java programming to tenth-grade students. The study found no difference in learning outcomes, but the group using the physical robots reported a greater level of enjoyment [78]. In another study, primary-school children aged 5–12 used two different approaches to control a Lego Mindstorms robot [64, 65]. The use of tangible blocks was compared with graphical images of the same blocks embedded in a draggable interface. Students preferred the tangible interface, younger children were able to produce programs faster with the tangible devices than with the graphical counterparts, and older children were more willing to experiment with alternative algorithms in the tangible environment. However, since the tangibles themselves were limited and the graphical components were an exact representation of the same tangible blocks, the results cannot be generalised to environments where these are not identical. Nevertheless, this in contrast to the Wu et al. high school study [78] and warrants further research.

Two day workshops were held in Slovenia to introduce programming to in-service high school teachers using an eProDas microcontroller module that facilitates data collection from sensors in real-time environments. The modules were variously controlled by Delphi, LabVIEW, Visual Basic, BASCOM. The teachers were encouraged to use their experience in the classroom, but no data on classroom activities were reported [36].

7 TEACHERS

The literature that focuses on teachers [8, 10, 22, 23, 26, 32, 33, 36, 48, 50, 52, 54, 60, 66, 67, 71, 73, 79] mainly concerns the challenges they face with incorporation of introductory programming into the curriculum and the requirement that they teach this course.

In the UK in 2014, Sentance and Csizmadia [67] carried out a survey of over 300 teachers preparing for the mandatory introduction of computing to the K–12 curriculum to elicit their perspectives on challenges and strategies. They found that of the issues raised, 40% related to challenges directly experienced by the teacher, 38% to difficulties experienced by the student, and 16% to resources. They also identified five key approach themes in terms of supporting students: unplugged type activities, contextualisation of tasks, collaborative learning, developing computational thinking, and scaffolding programming tasks. In Spain in 2017, Hijón-Neira et al. [23] sent a survey to 318 educational centres in Madrid, where computing is also mandatory. They found that all schools considered it useful to teach programming in schools, but that 39% were unable to teach the subject due to timetabling or teacher training issues and that most teachers lacked adequate methods appropriate for teaching programming.

In a study of schools in Sweden, Finland, and Lithuania, Pears et al. [54] also found that teacher training initiatives were critical for the success of incorporating introductory programming in the curriculum. Heintz et al. [21] confirmed that the need for teacher training and in-service professional development is a 'huge' issue in the implementation of the Swedish K–12 computing curriculum. In a Japanese survey of 309 teachers covering 44 of 47 prefectures, Ohashi [50] found that many teachers lacked confidence in implementing a 2020 introduction of programming in elementary school programs due to the low level of ICT use in teaching, teacher scepticism, the use of conventional teaching methods, and teacher workloads. Concluding that teachers were not technically or emotionally prepared for the planned introduction, Ohashi proposed improved resources and training, student-centred teaching, and reduced teacher workloads. Webb et al. [73] analysed curriculum developments in Australia, Israel, New Zealand, Poland, and the UK, and found teacher professional development to be critical for supporting curriculum change.

The above initiatives have provided the impetus for programs to support teachers. Kay and Moss [32] describe a three-day summer workshop in the US developed to teach K–12 teachers programming with Lego NXT-G robots. A post-workshop survey found that the teachers' confidence had increased dramatically and many intended to use the materials in their teaching. A follow-up survey nine months later found that many had indeed incorporated the material into their teaching programs. Kocijancic and Kušar [36] developed a course to teach programming to teachers in secondary technical schools. Rodhouse et al. [60] describe a set of Squeak Smalltalk tutorials and report on their successful use in teaching programming to middle- and high-school students.

After being taught Scratch, Czech pre-service teachers delivered 15 lessons to high-school students using an inquiry-based approach [70]. The pre-service teachers found it challenging to deliver the material effectively, suggesting that regardless of the ease of use of visual programming languages, substantial pedagogic content knowledge is still required for delivery of lessons. An identified research gap is the need for a similar study using text-based programming languages.

8 STUDENTS

The literature that focuses on students in K–12 introductory programming can be divided into three broad areas focused on student motivation and engagement, performance, and learning approaches. While a number of papers discussed in this section have been reported in Section 6. We focus here on the aspects of the work relevant to students in the identified areas. When teaching programming, the need to motivate students often guides the choice of technologies, activities, and resources. We found a number of papers that investigated student motivation and attitudes towards their programming environments or languages, many in the form of comparative studies.

Student engagement was found to be positively influenced by the use of physical devices and interfaces for learning programming. For example, tenth-grade students showed a preference for learning activities with physical rather than simulated robots [78], children aged 5–12 preferred a tangible rather than graphical interface for programming robots [64], and those aged 12–13 found it more engaging to program robots than wearable devices or standard desktop PC environments [47].

A number of comparative studies have found that students' level of motivation and engagement is strongly influenced by the programming environment. Ruf et al. [62] explored secondary school students' use of Scratch and Karel the Robot and found that the students using Karel the Robot showed higher self-regulation, but those using Scratch had higher intrinsic motivation. Nikou and Economides [49] explored changes in student motivation during two courses, one using Scratch and the other using App Inventor for Android. They found that intrinsic goal orientation, task value, control of learning beliefs, and self-efficacy increased from the beginning to the middle of the course regardless of the programming environment. No effect was found on extrinsic motivation.

We found a number of papers that compared student performance in different programming environments. Higher performance was found for high-school students using App Inventor than for those using Scratch, and both groups had higher performance than those using Pascal [52]. Block-based rather than text-based languages were found to help students in grades 8–12 to understand programs with complex structure [74, 75]. ViLLE, a programming visualisation environment used in high schools, was found to improve performance in learning Java compared with a standard environment [31]. However, no significant difference in performance was found between high-school students using physical and simulated robots [31].

Several studies have investigated students' approaches to learning to program. In an exploratory study, Ichinco and Kelleher [28] investigated how novices approach programming tasks, identifying hurdles, strategies, and types of task behaviour. Harms et al. [17] explored the strategic decisions that students make regarding instructional format when learning to program, and found that enjoyment, challenge, and perceived value all play important roles in students' choice of instructional format. A guided collaboration approach to programming tasks was investigated by Lin et al. [40] and found to enhance the performance of sixth-grade students learning to program in MSWLogo. Working with children aged 10–15, Hnin et al. [25] explored open-ended use of four different learning supports: tutorials, code puzzles, in-application documentation, and code suggestions; they found that participants leveraged in-context forms of help most frequently, but valued documentation for question-answering, and suggestions for opportunistic learning. While tutorials and puzzles were used less frequently than documentation and suggestions, they actually had the highest use-to-access ratio.

9 CONCLUSION AND SUGGESTED FUTURE WORK

Our review of literature on introductory programming in K–12 from 2003–2017 has shown a recent strong growth in research in this area across all age groups. This is perhaps unsurprising considering the initiatives worldwide to incorporate the study of programming into K–12 curricula, either as a standalone discipline or as an inter-disciplinary module woven into activities in other disciplines such as science, history, or mathematics.

As with introductory programming courses at the tertiary level, much of the work has focused on comparing programming environments, media, or languages to establish their effectiveness at engaging and motivating students and ensuring that learning outcomes are attained. However, few studies are large-scale or long-term and only one study in our set entailed a controlled experiment, highlighting a critical need for research in this space. Nevertheless, there seems to be consensus that Scratch is most appealing to children of primary-school age but less appealing to middle- and high-school students, who tend to prefer environments that permit more advanced development, such as App Inventor for Android. Students also seem to be engaged by the use of Lego NXT or similar tangible bricks, but graphical alternatives might be more valuable as they allow students to focus on programming rather than robot building. Further research is required to establish the relative effectiveness of these approaches.

We found a wide variety of introductory programming initiatives offered in a range of programming languages and environments, with most initiatives targeted at middle- and high-school students. However, studies show that the perception of programming and computing is formed as early as early primary school, especially for underrepresented groups such as women. Recent work has attempted to modify existing environments for primary-school children, with promising initial results, highlighting the need for further development in this area.

We also found a large number of studies that identify a need to support teachers in teaching the programming curriculum and in overcoming other barriers such as timetabling, teacher workload, and lack of confidence. It is concerning that we did not find evidence of systematic or widespread initiatives to train and support teachers in technology skills and appropriate pedagogy for teaching introductory programming in K–12. This is clearly an area for future work, with critical implications for future computing students.

REFERENCES

[1] Charoula Angeli, Joke Voogt, Andrew Fluck, Mary Webb, Margaret Cox, Joyce Malyn-Smith, and Jason Zagami. 2016. A K-6 computational thinking curriculum framework: Implications for teacher knowledge. Journal of Educational Technology & Society 19, 3 (2016).

[2] Anon. 20nn. Omitted for blind review.

[3] Australian Curriculum. 2019. Australian Curriculum - Digital Technologies. https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/digital-technologies/aims/. (2019).

[4] F.B.V. Benitti. 2012. Exploring the educational potential of robotics in schools: A systematic review. Computers & Education 58, 3 (2012), 978–988.

[5] Jens Bennedsen and Michael E. Caspersen. 2007. Failure Rates in Introductory Programming. SIGCSE Bull. 39, 2 (June 2007), 32–36. https://doi.org/10.1145/1272848.1272879

[6] Marina Umaschi Bers, Louise Flannery, Elizabeth R Kazakoff, and Amanda Sullivan. 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Computers & Education 72 (2014), 145–157.

[7] J. Blonskis and V. Dagienė. 2006. Evolution of Informatics Maturity Exams and Challenge for Learning Programming. In Informatics Education – The Bridge between Using and Understanding Computers. 220–229.

[8] Torsten Brinda. 2006. Discovery learning of object-oriented modelling with exploration modules in secondary Informatics education. Education and Information Technologies 11, 2 (apr 2006), 105–119.

[9] Royal Society (Great Britain). 2012. Shut down or restart?: The way forward for computing in UK schools. Royal Society.

[10] A. Brodnik, M. Lokar, and N. Mori. 2017. Activation of Computer Science Teachers in Slovenia. In IFIP Advances in Information and Communication Technology. 658–662.

[11] Neil C.C. Brown, S. Sentance, T. Crick, and S. Humphreys. 2014. Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education 14, 2 (2014), 9.
[12] National Research Council et al. 2010. Committee for the Workshops on Computational Thinking. In Report of a workshop on the scope and nature of computational thinking, Natl Academy Pr.
[13] J. Gal-Ezer and C. Stephenson. 2014. A tale of two countries: Successes and challenges in K-12 computer science education in Israel and the United States. ACM Transactions on Computing Education 14, 2 (2014), 8.
[14] V. Garneli, M. N. Giannakos, and K. Chorianopoulos. 2015. Computing Education in K-12 Schools: A review of the Literature. In 2015 IEEE Global Engineering Conference (EDUCON). IEEE, 543–551.
[15] Shuchi Grover and Roy Pea. 2013. Computational thinking in K–12: A review of the state of the field. Educational Researcher 42, 1 (2013), 38–43.
[16] Shuchi Grover and Roy Pea. 2013. Using a Discourse-intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13). ACM, New York, NY, USA, 723–728.
[17] K. J. Harms, E. Balzuweit, J. Chen, and C. Kelleher. 2016. Learning programming from tutorials and code puzzles: Children's perceptions of value. In 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 59–67.
[18] K. J. Harms, D. Cosgrove, S. Gray, and C. Kelleher. 2013. Automatically Generating Tutorials to Enable Middle School Children to Learn Programming Independently. In Proceedings of the 12th International Conference on Interaction Design and Children. 11–19.
[19] K. Hayat, N. Ali Al-Shukaili, and K. Sultan. 2016. Alice in Oman. Education and Information Technologies 22, 4 (2016), 1553–1569.
[20] F. Heintz, L. Mannila, and T. FÃďrnqvist. 2016. A review of models for introducing computational thinking, computer science and computing in K-12 education. In 2016 IEEE Frontiers in Education Conference (FIE). 1–9. https://doi.org/10.1109/FIE.2016.7757410
[21] F. Heintz, L. Mannila, L. Nordén, P. Parnes, and B. Regnell. 2017. Introducing Programming and Digital Competence in Swedish K-9 Education. In Informatics in Schools: Focus on Learning Programming. 117–128.
[22] F. Heintz, L. Mannila, K. Nygårds, P. Parnes, and B. Regnell. 2015. Computing at School in Sweden – Experiences from Introducing Computer Science within Existing Subjects. In Lecture Notes in Computer Science. 118–130.
[23] R. Hijón-Neira, L. Santacruz-Valencia, D. Pérez-Marín, and M. Gómez-Gómez. 2017. An analysis of the current situation of teaching programming in Primary Education. In 2017 International Symposium on Computers in Education (SIIE). IEEE, 1–6.
[24] C. Hill, H. A. Dwyer, T. Martinez, D. Harlow, and D. Franklin. 2015. Floors and Flexibility: Designing a Programming Environment for 4th-6th Grade Classrooms. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). ACM, New York, NY, USA, 546–551.
[25] W. Hnin, M. Ichinco, and C. Kelleher. 2017. An exploratory study of the usage of different educational resources in an independent context. In 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 181–189.
[26] J. Hromkovič, T. Kohn, D. Komm, and G. Serafini. 2016. Combining the Power of Python with the Simplicity of Logo for a Sustainable Computer Science Education. In Informatics in Schools: Improvement of Informatics Knowledge and Perception. Springer International Publishing, 155–166.
[27] J. Hromkovič, G. Serafini, and J. Staub. 2017. XLogoOnline: A Single-Page, Browser-Based Programming Environment for Schools Aiming at Reducing Cognitive Load on Pupils. In Informatics in Schools: Focus on Learning Programming. 219–231.
[28] M. Ichinco and C. Kelleher. 2015. Exploring novice programmer example use. In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 63–71.
[29] V. Isomöttönen, A.I. Lakanen, and V. Lappalainen. 2011. K-12 Game Programming Course Concept Using Textual Programming. In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. 459–464.
[30] K. Kahn and N. Winters. 2017. Child-Friendly Programming Interfaces to AI Cloud Services. In Data Driven Approaches in Digital Education. 566–570.
[31] E. Kaila, T. Rajala, M.-J. Laakso, and T. Salakoski. 2010. Effects of Course-long Use of a Program Visualization Tool. In Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103. 97–106.
[32] J. S. Kay and J. G. Moss. 2012. Using robots to teach programming to K-12 teachers. In 2012 Frontiers in Education Conference Proceedings. IEEE, 1–6.
[33] Mizue Kayama, Takao Futagami, Atsushi Konno, Takeharu Tasaki, and Cortland Starrett. 2010. Let's Go! Magical Spoons: A High School Learning Program for Information Coding Fundamentals. In Proceedings of the 2010 ACM Conference on Information Technology Education (SIGITE '10). ACM, New York, NY, USA, 95–104.
[34] Caitlin Kelleher and Randy Pausch. 2007. Using Storytelling to Motivate Programming. Commun. ACM 50, 7 (July 2007), 58–64.
[35] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering – A systematic literature review. Information and Software Technology 51, 1 (2009), 7–15.
[36] S. Kocijancic and T. Kušar. 2008. Introducing programming languages through data acquisition examples. MIPRO 2008 - 31st International Convention Proceedings: Computers in Education 4 (2008), 258–261.
[37] Eleni Konidari and Panos Louridas. 2010. When Students Are Not Programmers. ACM Inroads 1, 1 (March 2010), 55–60.
[38] Maria Kordaki. 2010. A drawing and multi-representational computer environment for beginnersâĂŹ learning of programming using C: Design and pilot formative evaluation. Computers & Education 54, 1 (2010), 69 – 87.
[39] W. W.Y. Lau, G Ngai, Stephen C.F. Chan, and J. C.Y. Cheung. 2009. Learning Programming Through Fashion and Design: A Pilot Summer Course in Wearable Computing for Middle School Students. In Proceedings of the 40th ACM Technical Symposium on Computer Science Education. 504–508.
[40] J. Lin, Y. Lie, R. Ho, and C. Lie. 2007. Effects of guided collaboration on sixth graders' performance in logo programming. In 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports. 11–16.
[41] Jiangjiang Liu, Ethan Philip Hasson, Zebulun David Barnett, and Peng Zhang. 2011. A survey on computer science K-12 outreach: Teacher training programs. In Frontiers in Education Conference (FIE), 2011. IEEE, T4F–1.
[42] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16). ACM, New York, NY, USA, 284–289. https://doi.org/10.1145/2899415.2899432
[43] Linda Mannila. 2006. Progress Reports and Novices' Understanding of Program Code. In Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006 (Baltic Sea '06). ACM, New York, NY, USA, 27–31.
[44] L. Mannila. 2007. Novices' progress in introductory programming courses. Informatics in Education 6, 1 (2007), 139–152.
[45] Monica McGill and Adrienne Decker. 2017. Computer Science Education Repository. (2017). https://csedresearch.org
[46] Muhsin Menekse. 2015. Computer science teacher professional development in the United States: a review of studies published between 2004 and 2014. Computer Science Education 25, 4 (2015), 325–350.
[47] A. Merkouris, K. Chorianopoulos, and A. Kameas. 2017. Teaching Programming in Secondary Education Through Embodied Computing Platforms: Robotics and Wearables. Trans. Comput. Educ. 17, 2 (2017), 9:1–9:22.
[48] N. Nikolova and E. Stefanova. 2014. Inquiry-Based Science Education in Secondary School Informatics – Challenges and Rewards. In Information Technology and Open Source: Applications for Education, Innovation, and Sustainability. 17–34.
[49] S. A. Nikou and A. A. Economides. 2014. Transition in student motivation during a Scratch and an App Inventor course. In 2014 IEEE Global Engineering Education Conference (EDUCON). IEEE, 1042–1045.
[50] Yutaro Ohashi. 2017. Preparedness of Japan's Elementary School Teachers for the Introduction of Computer Programming Education. In Informatics in Schools: Focus on Learning Programming. Springer International Publishing, 129–140.
[51] C. B. Owen, L. Dillon, A. Dobbins, N. Keppers, M. Levinson, and M. Rhodes. 2016. Dancing Computer: Computer Literacy Though Dance. In Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media. 174–180.
[52] S. Papadakis, M. Kalogiannakis, N. Zaranis, and V. Orfanakis. 2016. Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study. International Journal of Technology Enhanced Learning 8, 3-4 (2016), 217–233.
[53] Seymour Papert. 1980. Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.
[54] A. Pears, V. Dagienė, and E. Jasute. 2017. Baltic and Nordic K-12 Teacher Perspectives on Computational Thinking and Computing. In Informatics in Schools: Focus on Learning Programming. 141–152.
[55] N. Pellas and E. Peroutseas. 2016. Gaming in Second Life via Scratch4SL: Engaging High School Students in Programming Courses. Journal of Educational Computing Research 54, 1 (2016), 108–143.
[56] N. Pellas and E. Peroutseas. 2017. Leveraging Scratch4SL and Second Life to motivate high school students' participation in introductory programming courses: findings from a case study. New Review of Hypermedia and Multimedia 23, 1 (2017), 51–79.
[57] Ljubomir Perković, Amber Settle, Sungsoon Hwang, and Joshua Jones. 2010. A framework for computational thinking across the curriculum. In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education. ACM, 123–127.
[58] Peter J Rich, Samuel F Browning, McKay Perkins, Timothy Shoop, Emily Yoshikawa, and Olga M Belikov. 2019. Coding in K-8: International Trends in Teaching Elementary/Primary Computing. TechTrends 63, 3 (2019), 311–329.

[59] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. Computer Science Education 13, 2 (2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200
[60] K.N. Rodhouse, B. Cooper, and S.E. Watkins. 2011. Programming for pre-college education using Squeak Smalltalk. Computers in Education Journal 21, 2 (2011), 101–111.
[61] K. Roy. 2012. App Inventor for Android: Report from a Summer Camp. In Proceedings of the ACM Technical Symposium on Computer Science Education. 283–288.
[62] A. Ruf, A. Mühling, and P. Hubwieser. 2014. Scratch vs. Karel: Impact on Learning Outcomes and Motivation. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education. 50–59.
[63] A. Sanzo, F. Schapachnik, P. Factorovich, and F. S. O’Connor. 2017. Pilas bloques: A scenario-based children learning platform. In 2017 Twelfth Latin American Conference on Learning Technologies (LACLO). IEEE, 1–6.
[64] T. Sapounidis and S. Demetriadis. 2013. Tangible Versus Graphical User Interfaces for Robot Programming: Exploring Cross-age Children’s Preferences. Personal Ubiquitous Comput. 17, 8 (2013), 1775–1786.
[65] T. Sapounidis, S. Demetriadis, and I. Stamelos. 2015. Evaluating Children Performance with Graphical and Tangible Robot Programming Tools. Personal Ubiquitous Comput. 19, 1 (2015), 225–237.
[66] C.D. Seals and L.O. Tripp. 2007. A study of science teachers utilizing visual programming techniques. IMSCI 2007 - International Multi-Conference on Society, Cybernetics and Informatics, Proceedings 2 (2007), 207–212. cited By 0.
[67] S. Sentance and A. Csizmadia. 2016. Computing in the curriculum: Challenges and strategies from a teacher’s perspective. Education and Information Technologies 22, 2 (2016), 469–495.
[68] G. Serafini. 2011. Teaching Programming at Primary Schools: Visions, Experiences, and Long-Term Research Prospects. In Lecture Notes in Computer Science. 143–154.
[69] F. Tort and B. Drot-Delange. 2013. Informatics in the French secondary curricula: Recent moves and perspectives. In International Conference on Informatics in Schools: Situation, Evolution, and Perspectives. 31–42.
[70] Jiří Vaníček. 2015. Programming in Scratch Using Inquiry-Based Approach. In Lecture Notes in Computer Science. Springer International Publishing, 82–93.
[71] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success. In Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER ’14). 19–26.
[72] Christopher Watson and Frederick W.B. Li. 2014. Failure Rates in Introductory Programming Revisited. In Proceedings of the 2014 Conference on Innovation &#38; Technology in Computer Science Education (ITiCSE ’14). 39–44.
[73] M. Webb, N. Davis, T. Bell, Y. J. Katz, N. Reynolds, D. P. Chambers, and M. M. Sysło. 2016. Computer science in K-12 school curricula of the 2lst century: Why, what and when? Education and Information Technologies 22, 2 (2016), 445–468.
[74] D. Weintrop and U. Wilensky. 2015. To Block or Not to Block, That is the Question: Students’ Perceptions of Blocks-based Programming. In Proceedings of the 14th International Conference on Interaction Design and Children. 199–208.
[75] D. Weintrop and U. Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research. 101–110.
[76] Jeanette Wing. 2011. Research notebook: Computational thinkingâĂŤWhat and why. The Link Magazine (2011), 20–23.
[77] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.
[78] C. Wu, I. Tseng, and S. Huang. 2008. Visualization of Program Behaviors: Physical Robots Versus Robot Simulators. In Lecture Notes in Computer Science. 53–62.
[79] D. Xu, A. Cadle, D. Thompson, U. Wolz, I. Greenberg, and D. Kumar. 2016. Creative Computation in High School. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education. 273–278.