Computing Education: An Overview of Research in the Field

Tom Crick

April 2017

Background

In May 2016 the Royal Society announced it was commissioning a study[1], funded by Google and Microsoft, to understand the challenges faced by educators delivering computing and computer science and share best practice which can be adopted more widely. The research will establish the progress that has been made since the introduction of the new English computing curriculum (Department for Education, 2013) in September 2014 – as well as recognising related curriculum and qualifications reforms across the rest of the UK – identify areas that still need to be addressed, and will be used as the basis of a wider action plan to transform computing in schools. This new project builds upon the Royal Society's impactful work in this space, *Shut Down or Restart?* (Royal Society, 2012), a review of ICT and computing education in schools in the United Kingdom (UK). The introduction of a new computing curriculum for 5-16 year-olds in England has, for the first time anywhere, established computer science and computational thinking as foundational subjects alongside English, mathematics and the sciences. England is in effect pioneering a brand new school subject from the age of five[2], with the rest of the world watching to see the impact. This is alongside a number of national and international initiatives to reinforce computer science's position as a mainstream Science, Technology, Engineering and Mathematics (STEM) discipline (Guzdial and Morrison, 2016)

At the same time, the government is standing back from guiding everyone about how to implement this seismic curriculum change in England, and inviting, employers, universities, professional societies, and educational establishments themselves to play the leading role on how to deliver this curriculum. Academia, industry, parents and educators have responded enthusiastically with a range of initiatives such as code clubs and informal networks for professional development. But there are limits to informal activity; a subject in its infancy needs high-quality teacher training and development, best practice in the classroom, and reliable materials for students; it also needs solid, evidence-based research about what works, proper co-ordination of activity, and effective dissemination of best practice.

Thus, the key objectives which the Royal Society is seeking to understand by commissioning this research project are:

[1]https://royalsociety.org/topics-policy/projects/computing-education/

[2]However, it is important to note that the new national curriculum in England is only mandatory for state-maintained primary and secondary schools; free schools and academies have more freedom and flexibility to decide the curriculum they follow; see here for more information on the types of schools in England: https://www.gov.uk/types-of-school/overview

- What current literature says about computing education, including effective pedagogy and assessment methods;
- Current practice in educational establishments with reference to computing education, including pedagogy; learning goals and content and assessment;
- The attitudes of school leaders and teaching staff towards computing education, including timetabling, teacher confidence, qualifications and motivation as well as engagement of the senior leadership team.

This is a two-stage programme by the Royal Society designed to improve the quality and scale of computing education in English educational establishments, with transferability across the UK and internationally. Stage one will identify and prioritise goals for stage two, which will achieve those goals through a series of separate projects. Stage two projects may include areas such as:

- Producing classroom resources, teacher guidance, and continuing professional development (CPD) programmes;
- Developing effective assessment tools that educators can use to understand and guide progress;
- Publishing guidance about how to address gender imbalance in the uptake of computing;
- Identifying opportunities for project work in schools, perhaps with corporate partners.

This literature review covers the first part of Work Package 1 of the project (*What do we know about pedagogy and assessment in computing?*), providing a comprehensive literature review on effective computing pedagogy.

Methodology

There is an emerging corpus of academic and pedagogic literature in computing education, including assessment, attainment and baselining of core digital and technology skills, contextualised by substantial policy reports that transcends education, skills and wider digital economy agendas.

Much of the recent UK-focused literature has been driven by the curricula reforms (and emerging professional practice) in England and Scotland, alongside more recent developments in Wales. There exists active international networks of computer science education researchers (for example, in Germany, US, Israel and New Zealand), providing quantitative and qualitative educational research insight into effective pedagogies, assessment, models for cascading best practices for in-service training and professional development, as well as curriculum changes and wider education policy reform.

The proposed methodology for this literature review was initially based on a systematic review (Gough et al., 2012) of the existing corpus of research and policy contributions in this space to identify, appraise, select and synthesise all high-quality research evidence and arguments relevant to the project aims and objectives. Furthermore, it aimed to evaluate the existing "grey literature" and semi-formal evaluations of practice and experience produced in the UK and internationally, which has required careful categorisation and evaluation of the methodologies and cohort sizes presented. However, during the preliminary phases of the review process, it became apparent that due to the significant quantities of references to pedagogy in computer science in the academic literature, conducting a rigorous systematic review in the relevant time frame would have been problematic. Therefore, multiple methods have been employed to search the literature to ensure a thorough coverage and to minimise the possibilities of omitting any promising research from the review. A number of key academic databases have been targeted, including Google Scholar, ACM Digital Library and IEEE eXplore, followed by searching individual leading international journals and conferences in computer science/ICT/digital/technology education. This traditional literature search of recent English-language literature in refereed educational research journals, using key words and then selecting for relevance to the detailed research questions, produced a selection of some hundreds of papers: these were organised into themes, and in each of these we have attempted to identify a recent meta-analysis or review that we could have confidence in. This survey then suggested some important gaps which we endeavoured to fill through wider searches. Finally, certain areas have been covered to add context to the wider overview of research in this field, but are not comprehensively covered; for example, the section on gender and diversity.

Effective Pedagogies in Computing

Introduction

There is significant international focus on recent and ongoing computing curriculum reform in the UK. A number of audits and studies of national-level curricula models have been conducted over the past five years (CAS, 2011; Hazzan et al., 2008; Hubwieser et al., 2011; Snyder, 2012; Sturman and Sizmur, 2011), with numerous nations and states engaged in efforts to revamp their compulsory-level computer science curricula (Hubwieser et al., 2015b; Webb et al., 2017). Relevant examples include the USA (both nationally (ACM et al., 2016) and at the state level (Ericson et al., 2016; Guzdial et al., 2014)), France (Baron et al., 2014), Italy (Bellettini et al., 2014), India (Raman et al., 2015), Israel (Armonia and Gal-Ezer, 2014; Gal-Ezer and Stephenson, 2014), New Zealand (Bell, 2014; Bell et al., 2012), Russia (Khenner and Semakin, 2014) and Sweden (Rolandsson and Skogh, 2014), with each country having different issues to address in implementing a concepts-rich computer science curriculum (along with ensuring they have the expertise in the teaching profession to deliver it effectively). It is worth noting that recommendations for academic computer science curricula have a long pedigree (Atchison et al., 1968). Despite the increasing number of success reports from several countries – with establishing computer science as a worthwhile and high-value subject a frequently-named educational goal – it is one that represents significant challenge in terms of research, assessment and teacher training (Vahrenhold, 2012), especially for early years education (Beauchamp, 2016; Bird et al., 2014; Manches and Plowman, 2017) and in the wider context of rethinking effective pedagogies for the digital age (Beetham and Sharpe, 2013). Due to the substantial differences of preconditions, circumstances and influence factors, it is often difficult to compare or transfer research results in the field of computer science education in schools from one country to another; a framework has been developed – the "Darmstadt model" (Hubwieser, 2013) – to reflect all factors that might be relevant for computer science education in schools (Hubwieser et al., 2014, 2015a; Passey, 2017). Furthermore, throughout all of these reforms, there is a clear imperative to provide high-quality professional service to upskill existing and new educators to be able to effectively deliver the curriculum (Ericson et al., 2014; Hazzan et al., 2015; Sentance et al., 2012, 2013, 2014), reflecting a variety of challenges (Ni, 2009; Sentance and Csizmadia, 2017).

Policy and practice in the higher education sector is also of importance, particularly from a curriculum standards and guidelines perspective; for example from the UK's Quality Assurance Agency and its Subject Benchmark Statement for Computing (QAA, 2016) and the US Association of Computing Machinery (ACM) model curricula for undergraduate degree programmes (ACM, 2010, 2013, 2014, 2016), especially from a pedagogic perspective (Dziallas and Fincher, 2015). For universities across the UK offering computer science degrees, the ongoing school curriculum reform has had uncertain (and emerging) impact on the delivery (and thus pedagogy) of their undergraduate programmes, with the diversity of the educational background of applicants likely to increase before it narrows: it is certainly possible now for prospective students to have anywhere from zero to four or five years experience (and potentially two school qualifications) in computer science with some knowledge of programming.

Theories of Learning

There are long-established groundings in the literature to key educational research themes and how they support effective pedagogies and underpin high-quality learning and teaching in ICT and computing (Cox et al., 2004).

Constructivism is a philosophical viewpoint about the nature of knowledge, representing an epistemological stance. Constructivists interpret learning as the development of personalised knowledge frameworks that are continually refined, with learners actively constructing mental models to understand the world around them, rather than have understanding passively "dumped into their brains" (Esper et al., 2012). According to this theory, to learn, a student must actively construct knowledge, rather than simply absorbing it from textbooks and lectures (Davis et al., 1990; Meyer et al., 2010). There is an important distinction to note between *social constructivism* – which primarily focuses on the development of an individual's understanding – and *cognitive constructivism* – which focuses on knowledge constructed through interaction with others. The influential idea of cognitive apprenticeship (Brown et al., 1989), which holds that masters of a skill often fail to take into account the implicit processes involved in carrying out complex skills when they are teaching novices. To combat these tendencies, cognitive apprenticeships are designed, among other things, to bring these tacit processes into the open, where students can observe, enact, and practice them with help from the teacher, which honors the situated nature of knowledge. Further work in social constructivism includes communities of practice (Lave, 1991), with a primary focus on learning as social participation – that is, an individual as an active participant in the practices of social communities, and in the construction of his or her identity through these communities (Wenger, 2000).

The work of Piaget (1950) and Vygotsky (1978) is prominent in this space. Piaget described constructivism in education as being the process whereby students constructed their own unique systems of knowing (Piaget, 1950), in consequence of which the teacher should focus on this individual process of internal construction rather than standing at the front and dictating their own models. Students develop their own self-constructed rules, or "alternative frameworks" (Ben-Ari, 2001). This appears to be a common theme in programming, in which these alternative frameworks "naturally occur as part of the transfer and linking process" (Clancy, 2004); they represent the prior knowledge essential to the construction of new knowledge. When learning, the student modifies or expands his or her framework in order to incorporate new knowledge. Piaget's theory of constructivist learning has had wide-ranging impact on learning theories and teaching methods in education, and has been an underlying theme of many education reform movements across the world; it has also been proposed as a suitable pedagogy for information and computing sciences in the past (Boyle, 2000), particularly for computer programming (Lister, 2016). This constructivist approach informs the US National Research Council's recommendations for the adoption of active learning pedagogies in the classroom (Eberlein et al., 2008), particularly focused around the perceived utility of the "classroom lecture" and exploratory homework, a mechanism to support active learning for teaching programming languages by seeking to develop a model for students of how to explore and understand key constructs and concepts (Esper et al., 2012).

Constructivism does not refer to a specific pedagogy, although it is often confused with constructionism, an educational theory developed by Seymour Papert, inspired by constructivist and experiential learning ideas of Piaget. There are thus strong pedagogical links to the work of Papert (1993) and social constructionism, where learners work together to construct solutions (Kafai and Resnick, 1996), potentially through distributed means (Resnick, 1996), as well as how constructionist learning in science and mathematics can be applied to computing and the digital world (Kafai, 2006; Kafai and Resnick, 1996). Pedagogical approaches that emphasise constructive and collaborative learning in CS1 classrooms, based on empirical results, have been conducted (Van Gorp and Grissom, 2001), with potential application to schools. This leads into the pedagogic approaches around designing and making artefacts, particularly on the idea of "computational participation" (Kafai et al., 2014), as well as using robotics (Barreto and Benitti, 2012), although there are tensions with "technocentrism" and taking a constructionist approach to learning (Brennan, 2015).

While there have been criticism of Bloom's taxonomy (Bloom, 1965) – particularly as the classification was not a properly constructed taxonomy, as it lacked a systemic rationale of construction – work has been done on whether the taxonomy is appropriate for computer science, especially with the structure of observed learning outcomes (SOLO) taxonomy (a general educational taxonomy that describes levels of increasing complexity in student's understanding of subjects). Bloom's taxonomy of the cognitive domain and the SOLO taxonomy are being increasingly widely used in the design and assessment of courses, but there are some drawbacks to their use in computer science (Anderson et al., 2013); researchers have advocated its use for specifying learning outcomes in computer science prior to assessment (Starr et al., 2008), for computational thinking and teaching programming (Selby, 2015), as well as attempting to 'benchmark' the content of a computing degree (Fuller et al., 2007; Johnson and Fuller, 2006).

There is also extensive work in threshold concepts (Land et al., 2008; Meyer et al., 2010) – as a subset of the core concepts in a discipline – for computing, and specifically in programming (Khalife, 2006); while not necessarily pedagogy, identifying and understanding which concepts are transformative provides valuable insight into key parts of the curriculum, as these are the building blocks that must be understood. Threshold concepts must be:

- *transformative:* they change the way a student looks at things in the discipline;
- *integrative:* they tie together concepts in ways that were previously unknown to the

student;

- *irreversible:* they are difficult for the student to un-learn;
- *potentially troublesome for students:* they are conceptually difficult, alien, and/or counter-intuitive;
- *often boundary markers:* they indicate the limits of a conceptual area or the discipline itself.

Students who have mastered these threshold concepts have, at least in part, "crossed over from being outsiders to belonging to the field they are studying" (Eckerdal et al., 2006), although there is some dispute on how they apply to computer science (Boustedt et al., 2007). Furthermore, perceptions (Lonati et al., 2011) and issues with domain terminology is still a barrier to many – both practitioners and policymakers – although there is ongoing work in the UK to address this (Simon et al., 2015; UKForCE, 2016).

There are a number of other pedagogical approaches that have been used to identify key introductory computing topics;Delphi processes – a structured multi-step process that uses a group of experts to achieve a consensus opinion (Clayton, 1997) – have been used to identify topics that are important and perceived to be difficult in each of three introductory computing subjects: discrete math, programming fundamentals, and logic design. These topic rankings can then be used to guide both the coverage of standardised tests of student learning (i.e. concept inventories) and can be used by educators to identify which topics merit emphasis (Goldman et al., 2008).

This naturally leads into the use of concept inventories, specialised assessment instruments that enable educational researchers to investigate student (mis)understandings of concepts in a particular domain i.e. how a student's conceptual framework matches the accepted conceptual framework of a discipline. While students experience a concept inventory as a set of multiple-choice items taken as a test, this belies its purpose, its careful development, and its validation (Almstrum et al., 2006). A concept inventory is not intended to be a comprehensive instrument, but rather a tool that probes a student's comprehension of a carefully-selected subset of concepts that give rise to the most common and pervasive mis-modellings; concept inventories have been developed and used in a number of STEM fields, with application to computing, for example digital logic (Herman et al., 2010), algorithms and data structures (Danielsiek et al., 2012).

Peer instruction (PI) is an effective active learning method that supports student-centric classrooms, where students construct their own understanding through a structured approach featuring questions with peer discussions. PI has been shown to increase learning in STEM disciplines such as physics and biology (Crouch and Mazur, 2001), as well as being an indicator of student success. The potential implications of widespread adoption of PI in computing has also been discussed (Porter et al., 2011, 2013a), with the general focus of PI research been on the in-class portion of PI: multiple choice questions and group discussion (Zingaro et al., 2013). It is generally assumed that early success in CS1 is crucial for success on the examination and course as a whole. Particularities of students, densely-connected CS1 content, and recurring core topics each suggest that it is difficult to rebound from early misunderstandings. PI data, in addition to examination data, has been used to explore relationships between in-class assessments and performance on the end of term assessment; early course performance very quickly and strongly predicts performance on the final examination and that subsequent weeks provide no major increase in that predictive power (Porter and Zingaro, 2014). Peer instruction has also been used to increase engagement in lectures, for example with students answering a multiple choice question typically using hand-held remote devices, discuss the question with their peers, and then answer the question again; studies have reflected on this approach, understanding the value for the teacher, and reporting the attitudes and opinions of the students (Simon et al., 2010). Furthermore work on the benefits of lab-centric instruction, a collection of pedagogical techniques enabled by converting class time in lecture to time in a supervised closed lab, with the intermediate goal of supporting students' in developing skills of self-assessment (Lewis et al., 2011).

Finally, there is a body of pedagogic approaches on the use games-based learning to support the acquisition of knowledge in computing education (Egenfeldt-Nielsen, 2007; Giannakos, 2013; Schmitz et al., 2011; Theodoropoulos et al., 2017), with a variety of factors affecting learner perception, engagement and performance Bourgonjon et al. (2010); Farrell and Moffat (2014); Jiau et al. (2009); Lee and Hammer (2011). For example, enhancing self-motivation to learn programming using game-based simulation and metrics, primarily to avoid tedious trial-and-error refinement processes by providing helpful clues on how the student might reprogram the strategy to improve the result (Jiau et al., 2009).

Computer Science Fundamentals

There are a range of pedagogic approaches to developing knowledge and understanding of key fundamental topics in computer science, for example understanding automata (Isayama et al., 2017), finite state machines and Turing machines (Korte et al., 2007), as well as key issues such as concurrency (Kolikant, 2004). A wide range of work has been conducted in algorithm design strategies (Levitin, 1999, 2016) and in particular, the use of analogies (Cao et al., 2016; Chee, 1993; Repenning and Perrone, 2000) and metaphors (Forišek and Steinová, 2012; Hidalgo-Céspedes et al., 2014). This is linked to students' misconceptions relating to algorithms and data structures, building in work on concept inventories (Coffey, 2013; Danielsiek et al., 2012). Tracing – to understand the notional machine provides insight into the process a computer goes through when executing a section of code is manually stepped through by the learner in order to understand how it will work and what the expected output should be (Sorva, 2013). Just showing a visualisation of a piece of code or an algorithm does not appear to improve learners' depth of understanding; annotation on, and next to, the code provides a visual guide of what actually happens when it executes. This makes the abstract concepts and hidden mechanisms visible and therefore easier to think and reason about (Sorva, 2013). However, just viewing program or algorithm visualisations does not necessarily lead to improved understanding: how students use the technology has a greater impact on effectiveness than what the technology shows them (Hundhausen et al., 2002). Instead, active engagement with the visualisation is the key factor either by predicting the next step, choosing suitable input data to achieve a particular output of learners constructing the visualisation themselves (Grissom et al., 2003; Petre and Blackwell, 1999).

Various studies have been conducted to determine factors that promote success in introductory college computer science courses, identifying possible predictive factors including: math background, attribution for success/failure (luck, effort, difficulty of task, and ability), domain specific self-efficacy, encouragement, comfort level in the course, work style preference, previous programming experience, previous non-programming computer experience, and gender (Cantwell Wilson and Shrock, 2001). Studies have revealed three predictive factors: comfort level, math, and attribution to luck for success/failure; comfort level and math background were found to have a positive influence on success, whereas attribution to luck had a negative influence. Furthermore, by considering the different types of previous computer experiences (including formal programming class, self-initiated programming, internet use, game playing, and productivity software use) that both a formal class in programming and game playing were predictive of success, as well as outreach programs (Franklin et al., 2013). Formal training had a positive influence and games a negative influence on class grade (Cantwell Wilson and Shrock, 2001). Much work has focused on planning and assessment of CS1 level topics – especially programming – and how this impacted on attainment of students (Matthasdóttir and Arnalds, 2015). Many university computing academics report bimodal grade distributions in their CS1 classes, believing that such a distribution is due to there being an innate talent for programming, a so-called "geek gene" (Ahadi and Lister, 2013). Robins (2010) introduced the concept of learning edge momentum, which offers an alternative explanation for the purported bimodal grade distribution, with studies analysing empirical data from introductory programming class, looking for evidence of geek genes, learning edge momentum and other possible factors (Ahadi and Lister, 2013). Further work in this area in discussed later on in this chapter, particularly with how developing conceptual understanding of theoretical concepts apply to programming and computational thinking.

Programming and Coding

Since the launch of the curriculum in England, there has been significant media and public attention on this new "coding curriculum"[3], with programming dominating the focus on what is a much broader discipline; even the terminology used can be divisive: "coding" or "programming"? However, as part of the initial search of the wider computing education literature, it is clear there is a substantial corpus of work on developing and understanding effective pedagogies on how to teach programming.

For many years – and increasingly at all levels of compulsory and post-compulsory education – the choice of programming language to introduce the "art" (Knuth, 2011), "science" (Gries, 1981) and "discipline" (Dijkstra, 1976) of computer programming via key principles, constructs, syntax and semantics has been regularly revisited. So what is a good first programming language? The issues surrounding choosing a first language (Gupta, 2004; Kaplan, 2010) – and a Google Scholar search identified a number of papers of the form "*X as a first programming language*", going as far back as the 1970s (Gries, 1974) – appear to be legion, especially with discussions of what precisely we aim to achieve from teaching programming (Fincher, 1999; Schulte and Bennedsen, 2006), to psychological approaches (Winslow, 1996), gender gaps (Angel Rubio et al., 2015), adults learning programming (Guo, 2017), attitudes (Fesakis and Serafeim, 2009) and a new focus on developing transferable computational thinking and problem solving skills (Tedre and Denning, 2016; Wing, 2008). While in the past, research on the teaching of introductory programming had limited effect on classroom practice (Pears et al., 2007), increasingly relevant research exists across several disciplines including education and cognitive science, disciplinary differences have made this material inaccessible to many computing educators. Furthermore, computer science educators have not had access to comprehensive surveys of research in this area (McCracken et al., 2001; Pears et al., 2007).

The topics addressed in the literature span a wide range of problems and solutions as-

[3]e.g. https://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming

sociated with the teaching of programming such as introductory programming courses, exposition of the programming process, apprentice-based learning, functional programming first, problem-based learning, the use of on-line tutorials, object-oriented programming and Java, environments to introduce programming, model-driven programming as opposed to the prevailing language-driven approach, teaching software engineering, testing, extreme programming, frameworks, feedback and assessment, active learning, technology-based individual feedback, and mini project programming examinations (Bennedsen et al., 2008).

Mastery of basic syntactic and logical constructs is an essential part of learning to program. Unfortunately, practice exercises for programming basics can often be tedious (or perceived to be tedious), making it difficult to motivate students (Dasgupta and Resnick, 2014; Dasgupta et al., 2015). The idea of using "problets" – problem solving software assistants for learning, reinforcement and assessment of programming concepts – was introduced by Kumar (2005). They are designed to help students learn programming concepts through small-scale problem-solving e.g. code-tracing problems (Kumar, 2015), and as a supplement to large-scale programming traditionally used in introductory programming courses. Other simple "programming puzzles" was introduced by Parsons and Haden (2006): automated, interactive tools that provides practice with basic programming principles in an entertaining puzzle-like format. Careful design of the items in the puzzles allows the tutor to highlight particular topics and common programming errors. Since each puzzle solution is a complete sample of well-written code, use of the tool thus exposes students to good programming practices, particularly when coupled with using sub-goals (Morrison et al., 2016).

This links with work focusing on issues with students who are unable to "trace" code, implying they are unable to explain its syntax and function. Students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code (Lister et al., 2009) – the performance of students on code tracing tasks correlates with their performance on code writing tasks. A correlation was also found between performance on "explain in plain English" tasks and code writing. Further work in this space suggests the possibility of a hierarchy of programming-related tasks: knowledge of programming constructs forms the bottom of the hierarchy, with "explain in English", Parson's puzzles, and the tracing of iterative code forming one or more intermediate levels in the hierarchy (Lopez et al., 2008). The structure of observed learning outcomes (SOLO) taxonomy – a general educational taxonomy that describes levels of increasing complexity in student's understanding of subjects – has also been used to describe differences in the way students (particularly novice programmers) and educators solve small code reading exercises (Lister et al., 2006).

Programming is a hard craft to master and its teaching is challenging. An apprentice model, where students learn their craft from a master is an approach that can lead to improved student engagement (Astrachan and Reed, 1994; Vihavainen et al., 2011). Although traditionally applied to physical and vocational skills, the apprenticeship model can also be applied to the acquisition of cognitive skills such as those required for programming, as discussed in the *Theories of Learning* section. In this context, the master is required to focus on the programming process and demonstrates it through writing, debugging and running 'live' programs. This takes place whilst being observed by the student cohort; scaffolding is provided through the provision of regular practical exercises with good quality formative feedback (Crick et al., 2015).

There is a belief that programming – as opposed to, say analysis of algorithms, a closely related theoretical skill – is fundamentally a craft that needs immersion and practice (Fincher, 1999; Milne and Rowe, 2016). This is related to both general and discipline-specific pedagogy, with some positions implying that minimal guidance during instruction is less effective and efficient than guidance specifically designed to support the cognitive processing necessary for learning (Kirschner et al., 2006). Connected to this general point is a curious paradox in the teaching of programming: students are generally taught to write programs, and not particularly to read them (essentially overlooking the importance of code literacy), whereas in natural languages, be it the mother tongue or foreign language instruction, students are taught to read before they are taught how to write (Crick et al., 2015). Approaches such as discussion classes (Hagan and Sheard, 1998) and live-coding – defined as "the process of designing and implementing a [coding] project in front of class during lecture period" – can be effective in teaching introductory programming, with experimental data indicating that teaching via live-coding is as good as if not better than using static code examples (Rubin, 2013).

Pair programming (Williams and Kessler, 2000) is another common theme; prior research on pair programming has found that compared to students who work alone, students who pair have shown increased confidence in their work, greater success in CS1, and greater retention in computer-related majors (Hanks et al., 2004). A systematic review by Salleh et al. (2011) showed that students' skill level was the factor that affected pair programming's effectiveness the most. The most common measure used to gauge pair programming's effectiveness was time spent on programming; in addition, students' satisfaction when using pair programming was overall higher than when working solo and was effective in improving students' grades on assignments. Furthermore, pairing students were more likely to turn in working programs, and these programs correctly implemented more required features. An unexpected but significant finding was that pairing students were more likely to submit solutions to their programming assignments (Hanks et al., 2004). From a qualitative, student-focused approach, how do students define, experience, and value the pair programming experience; especially how do they experience and value it compared to solo programming? Students get stuck less and explore more ideas while pairing, and believe that pair programming helped them in CS1; however, students reported that when solo programming they were more confident and better understood their programs. Many students also said that they started work on their assignments earlier when soloing. Students also continue to use other students as resources even when working solo (Simon and Hanks, 2008). Finally, pair programming produces more proficient, confident programmers – and may help increase female representation in the field (McDowell et al., 2006).

As expected, there is a broad corpus on work on innovative and effective assessment mechanisms for programming, asking fundamental questions about what are we aiming to do when we teach programming (Fincher, 1999), through to misconceptions, attitudes and perceptions (Clancy, 2004). There have been multi-national, multi-institutional studies of assessment of programming skills of first-year computer science students in higher education institution (Lister et al., 2004; McCartney et al., 2013; McCracken et al., 2001), looking at effective pedagogies (Pears et al., 2007; Schulte and Bennedsen, 2006), the potential impact of programming expertise on learning motivation and academic achievement (Kori et al., 2016) – particularly failure rates (Bennedsen and Caspersen, 2007) – as well as the choice of introductory programming language and the potential impact on students' grades and attainment (Bergin and Reilly, 2006; Ivanović et al., 2015; Porter et al., 2013b; Simon et al., 2006). Research into effective study habits has provided insight into preparation for assessment, as well as resilience (Nam Liao, 2016; Willman et al., 2015); in many instances, high-quality exemplification is a key feature of successful initiatives (Scott, 2013). Recent international surveys of coding practices in K-8 have provided insight into demographics, experience, teaching practices and teacher confidence[4].

From a higher education perspective, there is a substantial body of work around effective pedagogies to support high-quality learning, teaching and assessment of computing. In the UK, much work has been supported by the Higher Education Academy – an independent non-profit organisation committed to world-class teaching in higher education; they work in partnership with institutions and individuals in higher education supporting student success through collaboration and share teaching strategies and practice. This has been primarily done through the HEA's STEM disciplinary theme (HEA, 2017). As the national body for promoting high-quality learning and teaching in higher education institutions in the UK, they run a programme of discipline-specific events, including STEM teaching and learning conferences, funding pedagogic research on innovative pedagogies, learning & teaching resources and case studies to inform and improve practice (for example, looking at retention and attainment in computer science (Gordon, 2016), a key sector issue). In the context of what are perceived to be the most difficult introductory topics in computer science degrees, numerous key themes frequently appear (Dale, 2006), with innovative approaches to addressing them (Hazzan et al., 2011).

Furthermore, the teaching of introductory programming in many higher education institutions has started to move away from focusing primarily on syntax to developing a deeper understanding of principles of programming, transferable language semantics, underlying constructs and structures, as well as developing useful and usable software artefacts: in summary, building upon the craft of programming via software carpentry (Wilson et al., 2014), codemanship (Crick et al., 2015) and teamwork (Martinez et al., 2014). Although controversial (Guzdial, 2009), this practice has a dual focus: firstly, it develops a high-level appreciation for why programming is being taught – essentially to solve real-world problems, using the most appropriate languages, tools and environments; secondly, it allows embedding the use of tools, methodologies and techniques so as to start to develop best practice for real-world software development. While the aim of this approach is not to create industry-level programmers at the end of the degree programme, by fostering and supporting the development of a particular culture around creating useful and usable software artefacts, underpinned by rigorous knowledge and theory, ensures that students understand how software is designed, developed and maintained in industry, and have the internal framework for developing knowledge and understanding in new languages, tools and environments and methodologies when required to do so (Davenport et al., 2016). However, it is important to note that university educators have the freedom and flexibility to select whichever programming languages, tools and environments they wish; educators in schools and colleges may be constrained in their choice of programming languages by informal requirements of the UK awarding bodies (especially in regards to their specific GCSE Computer Science qualification).

In this evolving national and international environment of emerging policy and curricula, as well as the demands of developing innovative pedagogies and high-quality learning and teaching for computer science degree programmes, national surveys of introductory programming languages provide valuable insight into pedagogy and practice. Longitudinal studies has been conducted in Australia and New Zealand over the past 15 years; in

[4]http://peterjrich.com/blog/2016/10/19/coding-in-k-8-international-survey-initial-results/

2001 and 2003, and censuses were conducted on Australian and New Zealand universities to examine trends in the programming languages and environments used in introductory programming courses (Raadt et al., 2004). In 2010 and 2013 (Mason and Cooper, 2014; Mason et al., 2012) similar surveys were conducted online, providing further data on the types of programming languages used (the prevalence of Python), along with some insight into pedagogies and environments. A related national-scale survey conducted in the USA in 2011 also provided some insight into the state of computing education in that country (Davies et al., 2011), along with more recent surveying (Guo, 2014) (again, an increasingly preference for Python). An inaugural survey of UK universities was conducted in 2016 (Murphy et al., 2017), mirroring the Australasian questions and structure; across the UK there is a diversity of practice, with Java appearing to have most traction – perhaps from perceived industry usefulness – but with Python also on the rise, most likely due to school curriculum reforms. Further to this large corpus of work on computer science – and in particular, programming – pedagogies, there is an opportunity to identify best practice that can be adapted for use in schools and colleges (Tangney et al., 2010).

Computational Thinking

Computational thinking (Wing, 2008) is a key theme of the new computing curriculum in England – appearing in the first sentence of the purpose of study (Department for Education, 2013):

> "*A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world.*".

It refers to a collection of computational ideas and habits of mind that people in computing disciplines acquire through their work in designing programs, software, simulations, and computations performed by machinery. Known in the 1950s and 1960s as "algorithmic thinking", it meant a mental orientation to formulating problems as conversions of some input to an output and looking for algorithms to perform the conversion (Denning, 2009). An early definition of computational thinking as presented in Wing (2010):

> "*Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.*"

While increasingly being discussed in the context of being a "21st century skill" (Bocconi et al., 2016b,c), computational thinking is perhaps perceived to be one of the more challenging concepts to teach and assess, partly due to lack of clarity about its precise definition and core components. Over the past decade, computational thinking and related concepts have received increasing attention across the educational field, which has given rise to an increasing amount of academic and grey literature[5] (Lockwood and Mooney, 2017), but also numerous collaborative initiatives (Meerbaum-Salant et al., 2015). Despite this widespread interest and policy focus, successful computational thinking integration in compulsory education faces unresolved issues and challenges (Bocconi et al., 2016a). A 2014 working group report on computational thinking in K-9 education (Mannila et al., 2014) aiming to revealing to what extent different aspects of computational thinking are

[5]See: `https://idleclicks.wordpress.com/computational-thinking/`

already part of teachers' classroom practice. Well-established initiatives such as CSUnplugged[6], Barefoot Computing[7], as well as aspects of CAS Tenderfoot[8] in the UK, attempt to build teacher confidence in teaching computer science through algorithmic and computational thinking principles (Csizmadia et al., 2015).

Considerable evidence indicates that domain specific knowledge in the form of schemas is the primary factor distinguishing experts from novices in problem-solving skill. Evidence that conventional problem-solving activity is not effective in schema acquisition is also accumulating. It is suggested that a major reason for the ineffectiveness of problem solving as a learning device, is that the cognitive processes required by the two activities overlap insufficiently, and that conventional problem solving in the form of means-ends analysis requires a relatively large amount of cognitive processing capacity which is consequently unavailable for schema acquisition (Sweller, 1988). This links with the "use-modify-create" framework (Lee et al., 2011), representing three phases of students' cognitive and practical activity in computational thinking, as well as the use of sub-goals (Morrison et al., 2015).

Linking back to the previous section on effective pedagogies for programming, a study by an ITiCSE working group (McCracken et al., 2001) established that many students do not know how to program at the conclusion of their introductory courses; although this study from 2001 may be dated by recent reforms, it is clear more research needs to be done in this space. A popular explanation for this incapacity is that the students lack the ability to problem-solve; that is, they lack the ability to take a problem description, decompose it into sub-problems and implement them, then assemble the pieces into a complete solution (Lister et al., 2004). While this could also be attributed to the fact that many students have a fragile grasp of both basic programming principles and the ability to systematically carry out routine programming tasks (such as tracing through code), there appears to be a strong perceived link between programming and computational thinking (Davies, 2008; Hu, 2011).

As the term became more accepted and the 'for everyone' manifesto generated interest, the focus shifted to curricula (ACM et al., 2016; Barr and Stephenson, 2011; Bell et al., 2010; Brinda et al., 2009; CAS, 2012; Gal-Ezer et al., 1995; Hubwieser et al., 2015b; Iyer et al., 2010), classroom experiences (Ater-Kranov et al., 2010; Cooper et al., 2010; Demšar and Demšar, 2016; Lee et al., 2011; Rodriguez et al., 2017; Yadav et al., 2014) and cross-curricular activities (Basawapatna et al., 2011; Curzon et al., 2009; Eisenberg, 2010; Goldberg et al., 2013; Kubica, 2012; Kules, 2016; Perković et al., 2010); at this time, other fields were also exploring their connections to computational thinking (Eisenberg, 2010; Gal-Ezer and Zur, 2002; L'Heureux et al., 2012; Lu and Fletcher, 2009).

However, modern computational thinking initiatives should be well aware of the broad and deep history of computational thinking, or risk repeating already refuted claims, past mistakes, and already solved problems, or losing some of the richest and most ambitious ideas in computational thinking (Denning, 2007). Since Wing (2006), there have been attempts to provide a formal and comprehensive definition of computational thinking by authoritative individuals and groups (Aho, 2012; Barr and Stephenson, 2011; Denning, 2007; Google, 2013; Grover and Pea, 2013; Guzdial, 2008, 2012; National Research Council, 2010, 2011; Wing, 2008). Recent work by Tedre and Denning (2016) has examined a number of threats to computational thinking initiatives: lack of ambition, dogmatism,

[6]http://csunplugged.org

[7]http://www.barefootcas.org.uk

[8]http://www.computingatschool.org.uk/tenderfoot

knowing versus doing, exaggerated claims, narrow views of computing, overemphasis on formulation, and lost sight of computational models. Two interesting questions that have been raised in the literature: i) *computational thinking a unique and distinctive characterisation of computer science?* and ii) *is computational thinking an adequate characterisation of computer science?* (Denning, 2009).

Policy Context

Overview

With significant international focus on recent computing education policy reform (and emerging practice) in the UK, it is useful to briefly frame the wider policy context, particularly disaggregating the educational, societal and economic drivers. Since the development of ICT as a new curriculum subject in the late 1990s (McKinsey & Co., 1997; Stevenson, 1997), there has been scrutiny of its breadth, depth and impact on students' learning and attainment (Ofsted, 2001, 2002, 2004). The publication of the Nesta *Next Gen.* report in 2011 (Livingstone and Hope, 2011) and the Royal Society's *Shut Down or Restart?* report (Royal Society, 2012), coupled with the disapplication of the ICT programme of study in England and scrutiny of the quality of a range of vocational qualifications (Royal Academy of Engineering, 2012; Wolf, 2011), left many educational establishments uncertain about the direction of this subject area, as well as poor wider perceptions of the discipline (Quinlan, 2015). However, a new computing curriculum (and associated qualifications) was starting to emerge (Brown et al., 2013, 2014), being published in 2013 for first delivery in September 2014 (Department for Education, 2013) from aged five (Manches and Plowman, 2017) to qualifications at 16 and 18. Alongside reforms in Scotland as part of their Curriculum for Excellence (Scottish Government, 2008), with a clear strand of computing science, as well as emerging education and skills reform in Wales (Arthur et al., 2013; Crick and Moller, 2015; Donaldson, 2015) and Northern Ireland (Perry, 2015). This has been supported by a strong European theme (Informatics Europe and ACM Europe Working Group, 2013), particularly as part of the wider Digital Agenda for Europe[9] initiative to promote digital literacy, skills and inclusion. Throughout all of these reforms, there is a clear imperative on providing high-quality professional service to upskill existing and new educators to be able to effectively deliver the curriculum (Cutts et al., 2017; Sentance and Csizmadia, 2017; Sentance et al., 2012, 2013, 2014).

From a societal and economic policy context, the "digital skills gap" has been highlighted by a number of high-profile policy reports, including by the UK Digital Skills Taskforce in 2014 (UK Digital Skills Taskforce, 2014), the House of Lords Select Committee on Digital Skills in 2015 (House of Lords Select Committee on Digital Skills, 2015), as well as the House of Commons Select Committee on Science & Technology in 2016 (House of Commons Select Committee on Science & Technology, 2016). From a wider societal context, a 2017 report from the Children's Commissioner for England's Growing Up Digital Taskforce stated that: "*The current Computing curriculum sets out in detail the technical skills and some of the legal knowledge a child should have at different ages. The Childrens Commissioner however believes this is too narrow, and often too late; your data protection rights, for instance, are not taught until GCSE level, and GCSE Computing is*

[9] `https://ec.europa.eu/digital-single-market/digital-agenda-europe`

not compulsory." (Children's Commissioner for England, 2017).

Policy and practice in the higher education sector is also of relevance to this project, particular from a curriculum standards and guidelines perspective, for example from the QAA in the UK (QAA, 2016) and the ACM in the USA (ACM, 2010, 2013, 2014, 2016). For universities across the UK offering computer science degrees, the school curriculum reform has had uncertain (and emerging) impact on the delivery of their undergraduate programmes, with the diversity of the educational background of applicants likely to increase before it narrows: it is certainly possible now for prospective students to have anywhere from zero to four or five years experience (and potentially two school qualifications) in computer science with some knowledge of procedural, object-oriented and/or functional programming. Furthermore, over the past three years, there has been increasing scrutiny of the quality of teaching in UK universities, partly linked to the current levels – and potential future increases – of tuition fees (generally paid by the student through government-supported loans), as well as relative levels of graduate employability and the perceived value of professional body accreditation by industry. In February 2015, the Department of Business, Innovation & Skills initiated independent reviews of STEM degree accreditation and graduate employability[10], with a specific focus – the Shadbolt review – on computer science degree accreditation and graduate employability, reporting back in May 2016 (Shadbolt, 2016). A number of recommendations were made to address the relatively high unemployment rates of computer sciences graduates, particular on the quality of data, course types, gender and demographics.

Qualifications

The uptake of computing/computer science qualifications at GCSE and A-level in the UK can obviously be made by looking at the educational establishments that offer the qualification and the students sitting it. Not all schools and colleges offer computing qualifications at GCSE or A-level and not all students sit qualifications in computing. This is further complicated by the increasing divergence of the four education systems in the UK, two of whom are still undergoing curriculum reform. However, all of the major examination boards – AQA, OCR, Edexcel, CCEA and WJEC/Qualifications Wales) – offer GCSEs and/or A-Levels in computing/computer science. Even where a qualification is taught by a school, subject requirements might limit the type of student who is able to take the course. Whilst at A-level, computing is a well-established subject, it is only offered by a minority of centres, with some areas having no provision. Until recently, the number of students taking A-level computing has been in decline, but Joint Council for Qualifications figures show that since 2014 numbers have been increasing year-on-year (Joint Council for Qualifications, 2014, 2015a, 2016a). A new computing GCSE was introduced by OCR in 2011[11] with the first cohort of students sitting examinations in 2013. Understandably, not all schools and colleges adopted this qualification immediately, and whilst the number of centres and students have been increasing, the numbers have not yet matched those of ICT (Joint Council for Qualifications, 2016b). A similar picture has been observed at A-level with numbers of computing students rising 50% in five years but still well below ICT (Joint Council for Qualifications, 2011, 2015b). Additionally, with recent school funding changes at A-level, from a per subject to per student system

[10] https://www.gov.uk/government/collections/graduate-employment-and-accreditation-in-stem-independent-reviews

[11] http://www.ocr.org.uk/qualifications/gcse-computing-j275-from-2012/

in England, the computing cohort size of A-level providers now becomes a greater concern for the ongoing financial viability of the subject, with smaller subject cohorts potentially making the course too expensive for smaller providers (Kemp et al., 2016).

Gender and Diversity

While a comprehensive analysis of the history and challenges of gender and diversity in computer science is out of scope for this review, it is worth highlighting some of the key issues. A number of different strategies have been proposed for involving females in computing – particularly in higher education – but there is a clear and recognised issue with engaging and retaining females in computing from the 1990s onwards. Despite the current growing popularity of the computer science major in the USA, women remain sorely underrepresented in the field, continuing to earn only 18% of bachelors degrees; understanding womens low rates of participation in computer science is important given that the demand for individuals with computer science expertise has grown sharply in recent years (Lehman et al., 2017). Numerous studies (Adam et al., 2004; Bunderson and Christensen, 1995; Clarke, 1992; Wilson, 2003) have explored some of the reasons that may underlie the gender segregation and declining levels of female participation within the field of computing, including gender differences and differences in computer science majors vs. non-majors in ability in quantitative areas, educational goals and interests, experience with computers, stereotypes, as well as domain knowledge, confidence, personality, support and encouragement, stress and financial issues, gender discrimination, and attitudes toward the academic environment in computer science. Three themes often appear: communicative processes, social networks and legitimising claims to knowledge, overlaid by gendered-power relations (Robertson et al., 2001). One of the challenges is to conceptualise women's computer skills as real computing and to instead ask what is wrong with computing rather than what is wrong with women (Clegg and Trayhurn, 2000). Finally, it is worth noting that there is growing evidence that instructors in computer science tend to believe in the importance of innate ability (Lewis, 2007) and thus the resulting negative effects this has on diversity (Murphy and Thomas, 2008) and thus performance (Patitsas et al., 2016). Addressing these gender stereotypes about intellectual ability from an early age (Bian et al., 2017), as well as improving these fixed mindsets in computing courses can have a positive effect on learning outcomes (Cutts et al., 2010).

Men had more confidence in using computers than did women even when statistically-controlling quantitative ability; in fact, female CS majors had less computer confidence than did male non-majors (Beyer et al., 2003). Studies have been conducted to determine factors that promote success in introductory college computer science courses – particularly programming (Angel Rubio et al., 2015) – and what differences appear between genders on those factors (Clarke and Chambers, 1989), including gender-equitable pedagogical practices (Vekiri, 2013); three predictive factors have appeared: comfort level (with a positive influence), mathematics background (with a positive influence), and attribution to luck (with a negative influence) (Cantwell Wilson, 2002).

While a number of bodies have been active in promoting diversity in computing, especially addressing the poor gender balance. e.g. BCSWomen[12], CAS #include[13] and ACM-W[14], more recent work has been conducted in England to better understand the

[12] http://bcswomen.bcs.org

[13] http://casinclude.org.uk

[14] https://women.acm.org/

landscape – particularly, the A-level and GCSE computing cohorts – beyond the widely publicised disparity in gender (Joint Council for Qualifications, 2015b), looking at provider type, provider location, subject mix, the ethnicity and socio-economic status of students (Kemp et al., 2016).

Conclusions & Recommendations

There are a number of recognised issues with the existing corpus of work associated with effective pedagogies and assessment mechanisms for computing. In the first instance, much of the existing research has been conducted in post-compulsory or higher education, addressing some of the specific issues associated with degree-level study. While there is certainly transferability from higher education to compulsory education, we have to make sure that the approaches and pedagogies are relevant and appropriate to the age group and topics being covered. Furthermore, we have to consider the quality and transferability of some of the studies presented, especially with regards to cohort size, sampling, statistical significant and claimed impacts, particularly when studies are based on single-institution, non-sampled interventions.

Building upon the previous section, in which we discussed some of the limitations of the existing research base, in this section we will summarise some of the key points from the literature with an aim of capturing future research priorities and requirements to best support the ongoing implementation of the computing curriculum in England.

- **Identifying a UK Research "Grand Challenge" Theme:** there is a significant opportunity to identify and develop a long-term strategic research programme (and associated funding models) for the UK to "*develop effective pedagogies for computing education in schools*". However, it is important to recognise that this is a significant task that may take a number of years; it will not be possible to do everything. This could consist of a number of sub-strands around some of the key themes identified in this literature review, particularly focused on concepts, skills and pedagogies. For example, it could be framed around "big ideas in computing", in a similar way to work done on the *Big Ideas in Science* (Association for Science Education, 2017). It should aim to establish cross-cutting understanding of a set of "big ideas" in computing which include key strategic themes and concepts, as well as and appreciation of computing's role and impact on society.

- **Identifying research with immediate applicability to schools**: a clear short-term priority exists in identifying "low-hanging fruit" that could be quickly transferred and translated for rapid adoption in schools and colleges by educators, especially from the higher education sector. For example, there are key areas of pedagogy that have a strong empirical evidence base and could be applied to schools, as well as calling for more school-specific research; for example, tracing, peer instruction and pair programming, as well as work on effective pedagogies of programming and the apprenticeship model from higher education.

- **Improving the strength of the UK research base:** it is a known issue that there is a lack of UK capacity for computer science education research, with small clusters at a few institutions across the UK. It is clear that the researchers and groups at these few institutions – namely, King's College London, University of

Kent, Queen Mary University of London, University of Glasgow, as well as individuals at a number of other institutions across the UK – do not currently provide the critical mass necessary to drive forward computer science education research, from both a pedagogical and practice perspective. This is contrasted by international strengths in STEM education, but is also an artefact of clear (if any) funding mechanisms to support computer science education research in the UK. Therefore, a priority recommendation is to identify funding models and schemes to support existing researchers, as well as develop researchers and groups at other institutions across the UK. Funding models could support doctoral and/or post-doctoral research, as well as identifying strategic partners (e.g. Nuffield, UK Research Councils, Leverhulme, etc) for co-funded schemes. It is imperative that the level of research and number of researchers increases in the UK to support the curriculum and engage with and contribute to the international research domain. Precisely what these models of funding look like are somewhat dependent on the partner organisations – particularly if this includes UK Research Councils, charities and/or industry support. This literature review has provided the start of a gap-analysis to enable a prospective research programme to be developed.

- **Adopting interdisciplinary approaches:** addressing some of the cognitive issues with learning some of the key topics, how can we best expand our research collaborations to ensure that we have the educational/cognitive/psychological expertise to achieve what we need to achieve? For example, research themes in programming, computational thinking, creativity, diversity, perceptions, etc.

- **Addressing the long-standing terminological diversions**: it is clear that domain terminology is still a barrier to many, both practitioners and policymakers; while there is existing work to address this (UKForCE, 2016), as a community we will need to agree on consistent terminology and nomenclature going forward so as to be able define and deliver a coherent research programme.

- **Relationship with digital literacy/competency**: further to the recommendations in Royal Society (2012) – and the resulting curriculum reform in England – digital literacy has been largely sidelined during the expansion of computer science. While there are active national and international research communities in this space, there is a need to better understand how digital literacy/competency fits in the wider "computing" landscape (especially across the four nations of the UK).

- **Defining and evidencing the value and impact of computational thinking**: for example, how do you teach computational thinking effectively? The idea is frequently espoused that it is something you get from learning computing (or programming. Teaching it specifically, do you teach the separate CT elements separately, or do you teach them holistically as an overall "CT package"? Also, teaching skills vs. teaching the concepts – how do you teach them to students and how do you teach them to educators? Finally, how does "unplugged" work and how best to teach it in the context of CT?

- **Effective programming pedagogies**: in particular, tools for teaching programming; software tools, hardware gadgets and pedagogies for teaching effective programming. We have been asking similar research questions for nearly 30 years, but we do not yet have the rigorous evidence base; some of this work clearly requires more empirical evidence, contextualised to the specific school or university environment.

- **Improving the perceptions of young people towards computer science**: students' perceptions, wider public perception (potentially linking with the existing Wellcome science education work), as well as looking at gender/class/ethnicity/inclusion.
- **Local, regional and national mapping**: what is actually being taught in schools and colleges e.g. published schemes of work/curricula on school websites, aggregating at the local, regional and national level to better understand the landscape, as well as being able to contribute to international initiatives.

In summary, we need to identify and sustainably fund long-term research focused on supporting students' learning, so that they are able to obtain secure, resilient and transferable knowledge of the subject of computing in the UK.

References

ACM. Information Systems 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. Technical report, Association for Computing Machinery, 2010. http://www.acm.org/education/curricula/IS%202010%20ACM%20final.pdf.

ACM. Computer Science 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Technical report, Association for Computing Machinery, 2013. http://www.acm.org/education/CS2013-final-report.pdf.

ACM. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Technical report, Association for Computing Machinery, 2014. http://www.acm.org/education/se2014.pdf.

ACM. Computer Engineering 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Technical report, Association for Computing Machinery, 2016. http://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf.

ACM, Code.org, CSTA, Cyber Innovation Center, and National Math and Science Initiative. K-12 Computer Science Framework. https://k12cs.org, 2016.

A. Adam, D. Howcroft, and H. Richardson. A decade of neglect: reflecting on gender and IS. *New Technology, Work and Employment*, 19(3):222–240, 2004.

A. Ahadi and R. Lister. Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant? In *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research (ICER '13)*, pages 123–128, 2013.

A. V. Aho. Computation and Computational Thinking. *The Computer Journal*, 55(7): 832–835, 2012.

V. L. Almstrum, P. B. Henderson, V. Harvey, C. Heeren, W. Marion, C. Riedesel, L.-K. Soh, and A. Elliott Tew. Concept inventories in computer science for the topic discrete mathematics. *ACM SIGCSE Bulletin*, 38(4):132–145, 2006.

L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson, 2013.

M. Angel Rubio, R. Romero-Zaliz, C. M. noso, and J. del Rosal. Closing the gender gap in an introductory programming course. *Computers & Education*, 82(C):409–420, 2015.

M. Armonia and J. Gal-Ezer. High school computer science education paves the way for higher education: the Israeli case. *Computer Science Education*, 24(2-3), 2014.

S. Arthur, T. Crick, and J. Hayward. The ICT Steering Group's Report to the Welsh Government. http://learning.wales.gov.uk/docs/learningwales/publications/131003-ict-steering-group-report-en.pdf, September 2013.

Association for Science Education. Big Ideas in Science. http://www.ase.org.uk/resources/big-ideas, 2017.

O. Astrachan and D. Reed. AAA and CS 1 The Applied Apprenticeship Approach to CS 1. Technical report, Duke University, 1994.

W. F. Atchison, S. D. Conte, J. W. Hamblen, T. E. Hull, T. A. Keenan, W. B. Kehl, E. J. McCluskey, S. O. Navarro, W. C. Rheinboldt, E. J. Schweppe, W. Viavant, and D. M. Young, Jr. Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science. *Communications of the ACM*, 11(3):151–197, 1968.

A. Ater-Kranov, R. Bryant, G. Orr, S. Wallace, and M. Zhang. Developing a community definition and teaching modules for computational thinking: accomplishments and challenges. In *Proceedings of the 2010 ACM Conference on Information Technology Education (SIGITE'10)*, pages 143–148, 2010.

G.-L. Baron, B. Drot-Delange, M. Grandbastien, and F. Tort. Computer Science Education in French Secondary Schools: Historical and Didactical Perspectives. *ACM Transactions on Computer Science Education*, 14(2)(11), 2014.

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

F. Barreto and V. Benitti. Exploring the educational potential of robotics in schools. *Computers & Education*, 58(3):978–988, 2012.

A. Basawapatna, K. Han Koh, A. Repenning, D. C. Webb, and K. Marshall. Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'11)*, pages 245–250, 2011.

G. Beauchamp. *Computing and ICT in the Primary School: From pedagogy to practice.* Routledge, 2nd edition, 2016.

H. Beetham and R. Sharpe. *Rethinking Pedagogy for a Digital Age: Designing for 21st Century Learning.* Routledge, 2nd edition, 2013.

T. Bell. Establishing a nationwide CS curriculum in New Zealand high schools. *Communications of the ACM*, 57(2):28–30, 2014.

T. Bell, P. Andreae, and L. Lambert. Computer Science in New Zealand high schools. In *Proceedings of the 12th Australasian Conference on Computing Education (ACE'10)*, pages 15–22, 2010.

T. Bell, P. Andreae, and A. Robins. Computer science in NZ high schools: the first year of the new standards. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*, pages 343–348, 2012.

C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, M. Torelli, and L. Zecca. Informatics Education in Italian Secondary Schools. *ACM Transactions on Computer Science Education*, 14(2)(15), 2014.

M. Ben-Ari. Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.

J. Bennedsen and M. E. Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

J. Bennedsen, M. E. Caspersen, and M. Kölling, editors. *Reflections on the Teaching of Programming: Methods and Implementations*. Springer, 2008.

S. Bergin and R. Reilly. Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, 16(4):303–323, 2006.

S. Beyer, K. Rynes, J. Perrault, K. Hay, and S. Haller. Gender differences in computer science students. *ACM SIGCSE Bulletin*, 35(1):49–53, 2003.

L. Bian, S.-J. Leslie, and A. Cimpian. Gender stereotypes about intellectual ability emerge early and influence childrens interests. *Science*, 355(6323):389–391, 2017.

J. Bird, H. Caldwell, and P. Mayne. *Lessons in Teaching Computing in Primary Schools*. Learning Matters, 2014.

B. S. Bloom. *he Taxonomy of Educational Objectives*. Longman Higher Education, 1965.

S. Bocconi, A. Chioccariello, G. Dettori, A. Ferrari, K. Engelhardt, P. Kampylis, and Y. Punie. Developing Computational Thinking in Compulsory Education – Implications for policy and practice. Technical report, European Union Scientific and Technical Research Reports, 2016a. EUR 28295 EN.

S. Bocconi, A. Chioccariello, G. Dettori, A. Ferrari, K. Engelhardt, P. Kampylis, and Y. Punie. Developing Computational Thinking: Approaches and Orientations in K-12 Education. In *Proceedings of EdMedia 2016*, 2016b.

S. Bocconi, A. Chioccariello, G. Dettori, A. Ferrari, K. Engelhardt, P. Kampylis, and Y. Punie. Exploring the Field of Computational Thinking as a 21st Century Skill. In *Proceedings of the 8th International Conference on Education and New Learning Technologies (EDULEARN16)*, 2016c.

J. Bourgonjon, M. Valcke, R. Soetaert, and T. Schellens. Students' perceptions about the use of video games in the classroom. *Computers & Education*, 54(4):1145–1156, 2010.

J. Boustedt, A. Eckerdal, R. McCartney, J. Moström, M. Ratcliffe, K. Sanders, and C. Zander. Threshold Concepts in Computer Science: Do They Exist and Are They Useful? In *Proceedings of the 38th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'07)*, pages 504–508, 2007.

T. Boyle. Constructivism: A Suitable Pedagogy for Information and Computing Sciences? http://users.ecs.soton.ac.uk/sysapl/www.ics.ltsn.ac.uk/pub/conf2000/Papers/tboyle.htm, 2000.

K. Brennan. eyond technocentrism: Supporting Constructionism in the Classroom. *Constructivist Foundations*, 10(3):289–296, 2015.

T. Brinda, H. Puhlmann, and C. Schulte. Bridging ICT and CS: educational standards for computer science in lower secondary education. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'09)*, pages 288–292, 2009.

J. Brown, A. Collins, and P. Duguid. Situated Cognition and the Culture of Learning. *Educational Researcher*, 18(1):32–42, 1989.

N. C. C. Brown, M. Kölling, T. Crick, S. Peyton Jones, S. Humphreys, and S. Sentance. Bringing Computer Science Back Into Schools: Lessons from the UK. In *Proceedings of the 44th ACM Techical Symposium on Computer Science Education*, pages 269–274, 2013.

N. C. C. Brown, S. Sentance, T. Crick, and S. Humphreys. Restart: The Resurgence of Computer Science in UK Schools. *ACM Transactions on Computer Science Education*, 14(2):1–22, 2014.

E. D. Bunderson and M. E. Christensen. An analysis of retention problems for female students in university computer science programs. *Journal of Research on Computing in Education*, 28(1):1–15, 1995.

B. Cantwell Wilson. A Study of Factors Promoting Success in Computer Science Including Gender Differences. *Computer Science Education*, 12(1-2):141–164, 2002.

B. Cantwell Wilson and S. Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin*, 33(1):184–188, 2001.

Y. Cao, L. Porter, and D. Zingaro. Examining the Value of Analogies in Introductory Computing. In *Proceedings of 2016 ACM Conference on International Computing Education Research (ICER'16)*, pages 231–239, 2016.

CAS. International Comparisons. Technical report, Computing At School, November 2011. http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf.

CAS. Computer Science: A curriculum for schools. Technical report, Computing At School, March 2012. https://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf.

Y. Chee. Applying Gentner's theory of analogy to the teaching of computer programming. *International Journal of Man-Machine Studies*, 38(3):347–368, 1993.

Children's Commissioner for England. Growing up Digital. https://www.childrenscommissioner.gov.uk/publications/growing-digital, January 2017. Growing Up Digital Taskforce.

M. Clancy. *Computer Science Education Research*, chapter Misconceptions and Attitudes that Interfere with Learning to Program, pages 85–100. Routledge, 2004.

V. Clarke. *In Search of Gender Free Paradigms for Computer Science Education*, chapter Strategies for involving girls in computer science. International Society for Technology in Education, 1992.

V. A. Clarke and S. M. Chambers. Gender-based factors in computing enrolments and achievement: Evidence from a study of tertiary students. *Journal of Educational Computing Research*, 5(4):409–429, 1989.

M. J. Clayton. Delphi: A Technique to Harness Expert Opinion for Critical Decision-Making Task in Education. *Educational Psychology*, 17:373–386, 1997.

S. Clegg and D. Trayhurn. Gender and Computing: Not the same old problem. *British Educational Research Journal*, 26(1):75–89, 2000.

J. W. Coffey. Integrating theoretical and empirical computer science in a data structures course. In *Proceedings of the 44th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'13)*, pages 23–28, 2013.

S. Cooper, L. C. Pèrez, and D. Rainey. K-12 computational learning. *Communications of the ACM*, 53(11):27–29, 2010.

M. Cox, M. Webb, C. Abbott, B. Blakeley, T. Beauchamp, and V. Rhodes. ICT and pedagogy: A review of the research literature. ICT in Schools Research and Evaluation Series 18, Department for Education and Skills, UK Government, 2004.

T. Crick and F. Moller. Technocamps: Advancing Computer Science Education in Wales. In *Proceedings of the 10th International Workshop in Primary and Secondary Computing Education*, pages 121–126, 2015.

T. Crick, J. H. Davenport, and A. Hayes. Innovative Pedagogical Practices in the Craft of Computing. Innovative Pedagogies in the Disciplines. Higher Education Academy, 2015. https://www.heacademy.ac.uk/innovative-pedagogical-practices-craft-computing.

C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69:970–977, 2001.

A. Csizmadia, P. Curzon, M. Dorling, S. Humphreys, T. Ng, C. Selby, and J. Woollard. *Computational thinking: A guide for teachers*. Computing At School, 2015.

P. Curzon, J. Peckham, H. Taylor, A. Settle, and E. Roberts. Computational thinking (CT): on weaving it in. *ACM SIGCSE Bulletin*, 41(3):201–202, 2009.

Q. Cutts, E. Cutts, S. Draper, P. O'Donnell, and P. Saffrey. Manipulating mindset to positively influence introductory programming performance. In *Proceedings of the 41st ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'10)*, pages 431–435, 2010.

Q. Cutts, J. Robertson, P. Donaldson, and L. O'Donnell. An evaluation of a professional learning network for computer science teachers. *Computer Science Education*, pages 1–24, 2017.

N. B. Dale. Most difficult topics in CS1: results of an online survey of educators. *ACM SIGCSE Bulletin*, 38(2):49–53, 2006.

H. Danielsiek, W. Paul, and J. Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*, pages 21–26, 2012.

S. Dasgupta and M. Resnick. Engaging novices in programming, experimenting, and learning with data. *ACM Inroads*, 5(4):72–75, 2014.

S. Dasgupta, S. M. Clements, A. Y. Idlbi, C. Willis-Ford, and M. Resnick. Extending Scratch: New pathways into programming. In *Proceedings of the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2015.

J. H. Davenport, A. Hayes, R. Hourizi, and T. Crick. Innovative Pedagogical Practices in the Craft of Computing. In *Proceedings of the 4th International Conference on Learning & Teaching in Computing and Engineering*, 2016.

S. Davies. The effects of emphasizing computational thinking in an introductory programming course. In *Proceedings of the 38th Annual Frontiers in Education Conference (FIE 2008)*, 2008.

S. Davies, J. A. Polack-Wahl, and K. Anewalt. A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 625–630, 2011.

R. B. Davis, C. A. Maher, and N. Noddings. Introduction: Constructivist views on the teaching and learning of mathematics. In *Constructivist views of the teaching and learning of mathematics*, Journal for Research in Mathematics Education – Monograph 4, pages 1–3. 1990.

I. Demšar and J. Demšar. CS Unplugged: Experiences and Extensions. In *Proceedings of 9th International Conference on Informatics in Schools (ISSEP'16)*, volume 9378 of *Lecture Notes in Computer Science*, 2016.

P. J. Denning. Computing is a natural science. *Communications of the ACM*, 50(7): 13–18, 2007.

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

Department for Education. National curriculum in England: computing programmes of study. https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study, September 2013.

E. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1st edition, 1976.

G. Donaldson. Successful Futures: Independent Review of Curriculum and Assessment Arrangements in Wales. http://gov.wales/docs/dcells/publications/150317-successful-futures-en.pdf, February 2015.

S. Dziallas and S. Fincher. ACM Curriculum Reports: A Pedagogic Perspective. In *Proceedings of 11th Annual International Conference on Computing Education Research (ICER'15)*, pages 81–89, 2015.

T. Eberlein, J. Kampmeier, V. Minderhout, R. S. Moog, T. Platt, P. Varma-Nelson, and H. B. White. Pedagogies of engagement in science. *Biochemistry and Molecular Biology Education*, 36:262–273, 2008.

A. Eckerdal, R. McCartney, J. Moström, M. Ratcliffe, K. Sanders, and C. Zander. Putting Threshold Concepts into Context in Computer Science Education. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE'06)*, pages 103–107, 2006.

S. Egenfeldt-Nielsen. *Educational Potential of Computer Games.* Bloomsbury 3PL, 2007.

M. Eisenberg. Bead Games, or, Getting Started in Computational Thinking Without a Computer. *International Journal of Computers for Mathematical Learning*, 15(2): 161–166, 2010.

B. Ericson, W. R. Adrion, R. Fall, and M. Guzdial. State-Based Progress Towards Computer Science for All. *ACM Inroads*, 7(4):57–60, 2016.

B. J. Ericson, M. Guzdial, and T. McKlin. Preparing secondary computer science teachers through an iterative development process. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE'14)*, pages 116–119, 2014.

S. Esper, B. Simon, and Q. Cutts. Exploratory homeworks: an active learning tool for textbook reading. In *Proceedings of the 9th Annual International Conference on International Computing Education Research (ICER '12)*, pages 105–110, 2012.

D. Farrell and D. C. Moffat. Adapting Cognitive Walkthrough to Support Game Based Learning Design. *International Journal of Game-Based Learning*, 4(3):23–34, 2014.

G. Fesakis and K. Serafeim. Influence of the familiarization with "scratch" on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE Bulletin*, 41(3):258–262, 2009.

S. Fincher. What are we doing when we teach programming? In *Proceedings of the 29th Annual Frontiers in Education Conference*, 1999.

M. Forišek and M. Steinová. Metaphors and analogies for teaching algorithms. In *Proceedings of the 43rd ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'12)*, pages 15–20, 2012.

D. Franklin, P. Conrad, B. Boe, K. Nilsen, C. Hill, M. Len, G. Dreschler, G. Aldana, P. Almeida-Tanaka, B. Kiefer, C. Laird, F. Lopez, C. Pham, J. Suarez, and R. Waite. Assessment of Computer Science Learning in a Scratch-Based Outreach Program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'13)*, pages 371–376, 2013.

U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. McGee Thompson, C. Riedesel, and E. Thompson. Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin*, 39(4): 152–170, 2007.

J. Gal-Ezer and C. Stephenson. A Tale of Two Countries: Successes and Challenges in K-12 Computer Science Education in Israel and the United States. *ACM Transactions on Computer Science Education*, 14(2)(8), 2014.

J. Gal-Ezer and E. Zur. The concept of 'algorithm efficiency' in the high school curriculum. In *Proceedings of the 32th Annual Frontiers in Education Conference (FIE 2002)*, 2002.

J. Gal-Ezer, C. Beeri, D. Harel, and A. Yehudai. A high school program in computer science. *IEEE Computer*, 28(10):73–80, 1995.

M. N. Giannakos. Enjoy and learn with educational games: Examining factors affecting learning performance. *Computers & Education*, 68:429–439, 2013.

D. Goldberg, D. Grunwald, C. Lewis, J. Feld, K. Donley, and O. Edbrooke. Addressing 21st century skills by embedding computer science in K-12 classes. In *Proceedings of the 44th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'13)*, pages 637–638, 2013.

K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin*, 40(1):256–260, 2008.

Google. Exploring Computational Thinking. https://edu.google.com/resources/programs/exploring-computational-thinking/#!ct-overview, 2013.

N. Gordon. Issues in retention and attainment in Computer Science. Technical report, Higher Education Academy, March 2016. https://www.heacademy.ac.uk/resource/issues-retention-and-attainment-computer-science.

D. Gough, S. Oliver, and J. Thomas. *An Introduction to Systematic Reviews*. Sage, 1st edition, 2012.

D. Gries. What should we teach in an introductory programming course? *ACM SIGCSE Bulletin*, 6(1):81–89, 1974.

D. Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, 1981.

S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in CS education: comparing levels of student engagement. In *Proceedings of the ACM Symposium on Software visualization (SoftVis'03)*, pages 87–94, 2003.

S. Grover and R. Pea. Computational Thinking in K12: A Review of the State of the Field. *Educational Researcher*, 42(1):38–43, 2013.

P. Guo. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities, July 2014.

P. J. Guo. Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI 2017)*, 2017.

D. Gupta. What is a good first programming language? *Crossroads*, 10(4), 2004.

M. Guzdial. Education: Paving the way for computational thinking. *Communications of the ACM*, 51(8):25–27, 2008.

M. Guzdial. How we Teach Introductory Computer Science is Wrong. http://cacm.acm.org/blogs/blog-cacm/45725-how-we-teach-introductory-computer-science-is-wrong/fulltext, October 2009.

M. Guzdial. A nice definition of computational thinking, including risks and cyber-security. https://computinged.wordpress.com/2012/04/06/a-nice-definition-of-computational-thinking-including-risks-and-cyber-security/, April 2012.

M. Guzdial and B. Morrison. Growing computer science education into a STEM education discipline. *Communications of the ACM*, 59(11):31–33, 2016.

M. Guzdial, B. Ericson, T. Mcklin, and S. Engelman. Georgia Computes! An Intervention in a US State, with Formal and Informal Education in a Policy Context. *ACM Transactions on Computer Science Education*, 14(2)(13), 2014.

D. Hagan and J. Sheard. The value of discussion classes for teaching introductory programming. *ACM SIGCSE Bulletin*, 30(3):108–111, 1998.

B. Hanks, C. McDowell, D. Draper, and M. Krnjajic. Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04)*, pages 176–180, 2004.

O. Hazzan, J. Gal-Ezer, and L. Blum. A model for high school computer science education: the four key elements that make it! *ACM SIGCSE Bulletin*, 40(1):281–285, 2008.

O. Hazzan, T. Lapidot, and N. Ragonis. *Guide to Teaching Computer Science: An Activity-Based Approach*. Springer, 2011.

O. Hazzan, T. Lapidot, and N. Ragonis. *Guide to Teaching Computer Science: An Activity-Based Approach*, chapter Teaching Methods in Computer Science Education, pages 105–135. 2015.

HEA. Science, Technology, Engineering and Mathematics (STEM). https://www.heacademy.ac.uk/disciplines/science-technology-engineering-and-mathematics-stem, 2017.

G. L. Herman, M. C. Loui, and C. Zilles. Creating the digital logic concept inventory. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*, pages 102–106, 2010.

J. Hidalgo-Céspedes, G. Marín-Raventós, and V. Lara-Villagrán. Playing with metaphors: a methodology to design video games for learning abstract programming concepts. In *Proceedings of the 2014 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'14)*, pages 348–348, 2014.

House of Commons Select Committee on Science & Technology. Digital Skills Crisis. http://www.publications.parliament.uk/pa/cm201617/cmselect/cmsctech/270/270.pdf, June 2016.

House of Lords Select Committee on Digital Skills. Make or Break: The UK's Digital Future. http://www.publications.parliament.uk/pa/ld201415/ldselect/lddigital/111/111.pdf, February 2015.

C. Hu. Computational thinking: what it might mean and what we might do about it. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE'11)*, pages 223–227, 2011.

P. Hubwieser. The Darmstadt model: a first step towards a research framework for computer science education in schools. In *Proceedings of the 6th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP'13)*, pages 1–14, 2013.

P. Hubwieser, M. Armoni, T. Brinda, V. Dagiene, I. Diethelm, M. N. Giannakos, M. Knobelsdorf, J. Magenheim, R. Mittermeir, and S. Schubert. Computer science/informatics in secondary education. In *Proceedings of the 2011 Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR'11)*, pages 19–38, 2011.

P. Hubwieser, M. Armoni, M. N. Giannakos, and R. T. Mittermeir. Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools. *ACM Transactions on Computer Science Education*, 14(2)(7), 2014.

P. Hubwieser, M. Armoni, and M. N. Giannakos. How to Implement Rigorous Computer Science Education in K-12 Schools? Some Answers and Many Questions. *ACM Transactions on Computer Science Education*, 15(2)(5), 2015a.

P. Hubwieser, M. N. Giannakos, M. Berges, T. Brinda, I. Diethelm, J. Magenheim, Y. Pal, J. Jackova, and E. Jasute. A Global Snapshot of Computer Science Education in K-12 Schools. In *Proceedings of the 2015 Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR'15)*, pages 65–83, 2015b.

C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

Informatics Europe and ACM Europe Working Group. Informatics education: Europe cannot afford to miss the boat. http://www.informatics-europe.org/images/documents/informatics-education-acm-ie.pdf, 2013.

D. Isayama, M. Ishiyama, R. Relator, and K. Yamazaki. Computer Science Education for Primary and Lower Secondary School Students: Teaching the Concept of Automata. *ACM Transactions on Computing Education*, 17(1)(2), 2017.

M. Ivanović, Z. Budimac, M. Radovanović, and M. Savić. Does the choice of the first programming language influence students' grades? In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, pages 305–312, 2015.

S. Iyer, M. Baru, Vijayalakshmi, Chitta, F. Khan, and U. Vishwanathan. Model Computer Science Curriculum for Schools. Technical report, March 2010.

H. C. Jiau, J. C. Chen, and K.-F. Ssu. Enhancing Self-Motivation in Learning Programming Using Game-Based Simulation and Metricsm. *IEEE Transactions on Education*, 52(4):555–562, 2009.

C. G. Johnson and U. Fuller. Is Bloom's taxonomy appropriate for computer science? In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pages 120–123, 2006.

Joint Council for Qualifications. A, AS and AEA Results Summer 2011. http://www.jcq.org.uk/examination-results/a-levels/2011/a-as-and-aea-results-summer-2011, 2011.

Joint Council for Qualifications. GCE Trends 2014. http://www.jcq.org.uk/examination-results/a-levels/2014/gce-trends-2014, 2014.

Joint Council for Qualifications. GCE Trends 2015. http://www.jcq.org.uk/examination-results/a-levels/2015/gce-trends-2015, 2015a.

Joint Council for Qualifications. GCSE gender, entry trends and regional charts 2015. http://www.jcq.org.uk/examination-results/gcses/2015/gcse-gender-entry-trends-and-regional-charts-2015, 2015b.

Joint Council for Qualifications. GCE Trends 2016. http://www.jcq.org.uk/examination-results/a-levels/2016/gce-trends-2016, 2016a.

Joint Council for Qualifications. GCSE and Entry Level Certificate Results Summer 2016. http://www.jcq.org.uk/examination-results/gcses/2016/gcse-and-entry-level-certificate-results-summer-2016, 2016b.

Y. B. Kafai. *The Cambridge Handbook of the Learning Sciences*, chapter Constructionism. Cambridge University Press, 2006.

Y. B. Kafai and M. Resnick. *Constructionism in Practice: Designing, Thinking, and Learning in A Digital World*. Routledge, 1996.

Y. B. Kafai, E. Lee, K. Searle, D. Fields, E. Kaplan, and D. Lui. A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles. *ACM Transactions on Computer Science Education*, 14(1)(1), 2014.

R. M. Kaplan. Choosing a first programming language. In *Proceedings of the ACM Conference on Information Technology Education*, pages 163–164, 2010.

P. Kemp, B. Wong, and M. Berry. The Roehampton Annual Computing Education Report: 2015 data from England. Technical report, University of Roehampton, December 2016. http://dx.doi.org/10.13140/RG.2.2.34832.81926.

J. Khalife. Threshold for the introduction of programming: providing learners with a simple computer model. In *Proceedings of 28th International Conference on Information Technology Interfaces*, pages 71–76, 2006.

E. Khenner and I. Semakin. School Subject Informatics (Computer Science) in Russia: Educational Relevant Areas. *ACM Transactions on Computer Science Education*, 14(2)(14), 2014.

P. A. Kirschner, J. Sweller, and R. E. Clark. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*, 41(2):75–86, 2006.

D. E. Knuth. *The Art of Computer Programming*. Addison Wesley, 3rd edition, 2011.

Y. Kolikant. Learning concurrency: evolution of students understanding of synchronization. *International Journal of Human-Computer Studies*, 60(2):243–268, 2004.

K. Kori, M. Pedaste, Äli Leijen, , and E. Tönisson. The Role of Programming Experience in ICT Students Learning Motivation and Academic Achievement. *International Journal of Information and Education Technology*, 6(5):331–337, 2016.

L. Korte, S. Anderson, H. Pain, and J. Good. Learning by game-building: a novel approach to theoretical computer science education. *ACM SIGCSE Bulletin*, 39(3): 53–57, 2007.

J. Kubica. *Computational Fairy Tales*. CreateSpace Independent Publishing Platform, 2012.

B. Kules. Computational thinking is critical thinking: Connecting to university discourse, goals, and learning outcomes. *Proceedings of the Association for Information Science and Technology*, 53(1):1–6, 2016.

A. N. Kumar. Generation of problems, answers, grade, and feedback – case study of a fully automated tutor. *Journal on Educational Resources in Computing*, 5(3)(3), 2005.

A. N. Kumar. Solving Code-tracing Problems and its Effect on Code-writing Skills Pertaining to Program Semantics. In *Proceedings of ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'15)*, 2015.

R. Land, J. H. Meyer, and J. Smith, editors. *Threshold Concepts and Transformational Learning*, volume 16 of *Educational Futures: Rethinking Theory and Practice*. Sense Publishers, 2008.

J. Lave. *Perspectives on Socially Shared Cognition*, chapter Situating learning in communities of practice. American Psychological Association, 1991.

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

J. J. Lee and J. Hammer. Gamification in Education: What, How, Why Bother? *Academic Exchange Quarterly*, 2011.

K. J. Lehman, L. J. Sax, and H. B. Zimmerman. Women planning to major in computer science: Who are they and what makes them unique? *Computer Science Education*, 26(4):277–298, 2017.

A. Levitin. Do we teach the right algorithm design techniques? *ACM SIGCSE Bulletin*, 31(1):179–183, 1999.

A. Levitin. Algorithm design strategies in CS curricula 2013: hits and misses. *Journal of Computing Sciences in Colleges*, 31(3):78–84, 2016.

C. Lewis. Attitudes and beliefs about computer science among students and faculty. *ACM SIGCSE Bulletin*, 39(2):37–41, 2007.

C. M. Lewis, N. Titterton, and M. Clancy. Developing students' self-assessment skills using lab-centric instruction. *Journal of Computing Sciences in Colleges*, 26(4):173–180, 2011.

J. L'Heureux, D. Boisvert, R. Cohen, and K. Sanghera. IT problem solving: an implementation of computational thinking in information technology. In *Proceedings of the 13th Annual Conference on Information Technology Education (SIGITE'12)*, pages 183–188, 2012.

R. Lister. Toward a Developmental Epistemology of Computer Programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (WiPSCE 2016)*, pages 5–16, 2016.

R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from Innovation and Technology in Computer Science Education (ITiCSE-WGR '04)*, pages 119–150, 2004.

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

R. Lister, C. Fidge, and D. Teague. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin*, 41(3): 161–165, 2009.

I. Livingstone and A. Hope. *Next Gen.* http://www.nesta.org.uk/publications/next-gen, 2011. Nesta.

J. Lockwood and A. Mooney. Computational Thinking in Education: Where does it fit? A systematic literary review. Technical report, National University of Ireland Maynooth, March 2017. https://arxiv.org/pdf/1703.07659.pdf.

V. Lonati, M. Monga, A. Morpurgo, and M. Torelli. What's the fun in informatics? working to capture children and teachers into the pleasure of computing. In *Proceedings of the 5th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP'11)*, pages 213–224, 2011.

M. Lopez, J. Whalley, P. Robbins, and R. Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the 4th international Workshop on Computing Education Research (ICER '08)*, pages 101–112, 2008.

J. J. Lu and G. H. L. Fletcher. Thinking about computational thinking. *ACM SIGCSE Bulletin*, 41(1):260–264, 2009.

A. Manches and L. Plowman. Computing education in childrens early years: A call for debate. *British Journal of Educational Technology*, 48(1):191–201, 2017.

L. Mannila, V. Dagiene, B. Demo, N. Grgurina, C. Mirolo, L. Rolandsson, and A. Settle. Computer science/informatics in secondary education. In *Proceedings of the 2014 Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR'14)*, pages 1–29, 2014.

J. Martinez, J. Martin, and A. Alonso. Teamwork competence and academic motivation in computer science engineering studies. In *Proceedings of the 2014 IEEE Global Engineering Education Conference*, 2014.

R. Mason and G. Cooper. Introductory programming courses in Australia and New Zealand in 2013 – trends and reasons. In *Proceedings of the 16th Australasian Computing Education Conference*, pages 139–147, 2014.

R. Mason, G. Cooper, and M. Raadt. Trends in introductory programming courses in Australian universities: languages, environments and pedagogy. In *Proceedings of the 14th Australasian Computing Education Conference*, pages 33–42, 2012.

Ásrún. Matthasdóttir and H. Arnalds. Rethinking teaching and assessing in a programming course a case study. In *Proceedings of the 16th International Conference on Computer Systems and Technologies (CompSysTech '15)*, pages 313–318, 2015.

R. McCartney, J. Boustedt, A. Eckerdal, K. Sanders, and C. Zander. Can first-year students program yet?: a study revisited. In *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research (ICER '13)*, pages 91–98, 2013.

M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33 (4):125–180, 2001.

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

McKinsey & Co. The future of Information Technology in UK schools: An independent inquiry. Technical report, 1997.

O. Meerbaum-Salant, B. Haberman, and S. Pollack. "Computer Science, Academia and Industry" as pedagogical model to enhance Computational thinking. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'15)*, pages 341–341, 2015.

J. H. Meyer, R. Land, and C. Baillie, editors. *Threshold Concepts and Transformational Learning*, volume 42 of *Educational Futures: Rethinking Theory and Practice*. Sense Publishers, 2010.

I. Milne and G. Rowe. Difficulties in Learning and Teaching Programming – Views of Students and Tutors. *Education and Information Technologies*, 7(1):55–66, 2016.

B. B. Morrison, L. E. Margulieux, and M. Guzdial. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of 2015 ACM Conference on International Computing Education Research (ICER'15)*, pages 21–29, 2015.

B. B. Morrison, L. E. Margulieux, B. Ericson, and M. Guzdial. Subgoals Help Students Solve Parsons Problems? In *Proceedings of the 47th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'16)*, pages 42–47, 2016.

E. Murphy, T. Crick, and J. H. Davenport. An Analysis of Introductory Programming Courses at UK Universities. *The Art, Science, and Engineering of Programming*, 1(2), 2017.

L. Murphy and L. Thomas. Dangers of a fixed mindset: implications of self-theories research for computer science education. In *Proceedings of 13th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'08)*, pages 271–275, 2008.

S. Nam Liao. Identify and Help At-Risk Students Before It Is Late. In *Proceedings of 2016 ACM Conference on International Computing Education Research (ICER'16)*, pages 295–296, 2016.

National Research Council. Report of a Workshop on the Scope and Nature of Computational Thinking. Technical report, The National Academies Press, 2010. https://doi.org/10.17226/12840.

National Research Council. Report of a Workshop on the Pedagogical Aspects of Computational Thinking. Technical report, The National Academies Press, 2011. https://doi.org/10.17226/13170.

L. Ni. What makes CS teachers change?: factors influencing CS teachers' adoption of curriculum innovations. *ACM SIGCSE Bulletin*, 41(1):544–548, 2009.

Ofsted. ICT in schools: The impact of government initiatives; an interim report. Technical Report HMI 264, Office for Standards in Education (Ofsted), April 2001.

Ofsted. ICT in schools: Effect of government initiatives; progress report. Technical Report HMI 423, Office for Standards in Education (Ofsted), April 2002.

Ofsted. ICT in schools: The impact of government initiatives five years on. Technical Report HMI 2050, Office for Standards in Education (Ofsted), May 2004.

S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, 1993.

D. Parsons and P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing (ACE '06)*, pages 157–163, 2006.

D. Passey. Computer science (CS) in the compulsory education curriculum: Implications for future research. *Education and Information Technologies*, 22(2):421–443, 2017.

E. Patitsas, J. Berlin, M. Craig, and S. Easterbrook. Evidence That Computer Science Grades Are Not Bimodal. In *Proceedings of 2016 ACM Conference on International Computing Education Research (ICER'16)*, pages 113–121, 2016.

A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin*, 39(4):204–223, 2007.

L. Perković, A. Settle, S. Hwang, and J. Jones. A framework for computational thinking across the curriculum. In *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'10)*, pages 123–127, 2010.

C. Perry. Coding in schools. http://www.niassembly.gov.uk/globalassets/documents/raise/publications/2015/education/3715.pdf, February 2015. Research and Information Service Briefing Paper, Northern Ireland Assembly.

M. Petre and A. F. Blackwell. Mental imagery in program design and visual programming. *International Journal of Human-Computer Studies*, 51(1):7–30, 1999.

J. Piaget. *The Psychology of Intelligence*. Routledge, 1950.

L. Porter and D. Zingaro. Importance of early performance in CS1: two conflicting assessment stories. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*, pages 295–300, 2014.

L. Porter, C. Bailey Lee, B. Simon, and D. Zingaro. Peer instruction: do students really learn from peer discussion in computing? In *Proceedings of 7th ACM Conference on International Computing Education Research (ICER'11)*, pages 45–52, 2011.

L. Porter, C. Bailey Lee, and B. Simon. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*, pages 177–182, 2013a.

L. Porter, M. Guzdial, C. McDowell, and B. Simon. Success in introductory programming: what works? *Communications of the ACM*, 56(8):34–36, 2013b.

QAA. *Subject Benchmark Statement: Computing*. Quality Assurance Agency, February 2016.

O. Quinlan. Young Digital Makers: Surveying attitudes and opportunities for digital creativity across the UK. http://www.nesta.org.uk/sites/default/files/young-digital-makers-march-2015.pdf, March 2015. Nesta.

M. Raadt, R. Watson, and M. Toleman. Trends in introductory programming courses in Australian universities: languages, environments and pedagogy. In *Proceedings of the 6th Australasian Computing Education Conference*, pages 277–282, 2004.

R. Raman, S. Venkatasubramanian, K. Achuthan, and P. Nedungadi. Computer Science (CS) Education in Indian Schools: Situation Analysis using Darmstadt Model. *ACM Transactions on Computer Science Education*, 15(2)(7), 2015.

A. Repenning and C. Perrone. Programming by example: programming by analogous examples. *Communications of the ACM*, 43(3):90–97, 2000.

M. Resnick. Distributed constructionism. In *Proceedings of the International Conference on Learning Sciences (ICLS'96)*, pages 280–284, 1996.

M. Robertson, S. Newell, J. Swan, L. Mathiassen, and G. Bjerknes. The issue of gender within computing: reflections from the UK and Scandinavia. *Information Systems Journal*, 11(2):111–126, 2001.

A. Robins. Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, 20(1):37–71, 2010.

B. Rodriguez, S. Kennicutt, C. Rader, and T. Camp. Assessing Computational Thinking in CS Unplugged Activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'17)*, pages 501–506, 2017.

L. Rolandsson and I.-B. Skogh. Programming in School: Look Back to Move Forward. *ACM Transactions on Computer Science Education*, 14(2)(12), 2014.

Royal Academy of Engineering. *Computing qualifications included in the 2014 Key Stage 4 Performance Tables: a guide for schools.* 2012.

Royal Society. Shutdown or restart? The way forward for computing in UK schools. https://royalsociety.org/topics-policy/projects/computing-in-schools/report/, January 2012.

M. J. Rubin. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*, pages 651–656, 2013.

N. Salleh, E. Mendes, and J. Grundy. Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 37(4):509–525, 2011.

B. Schmitz, A. Czauderna, R. Klemke, and M. Specht. Game based learning for computer science education. In *Proceedings of the Computer Science Education Research Conference (CSERC'11)*, pages 81–86, 2011.

C. Schulte and J. Bennedsen. What do teachers teach in introductory programming? In *Proceedings of the 2nd International Workshop on Computing Education Research*, pages 17–28, 2006.

J. Scott. The Royal Society of Edinburgh/British Computer Society Computer Science Exemplification Project. In *Proceedings of 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*, 2013.

Scottish Government. Curriculum for Excellence: A Framework for Learning and Teaching. http://www.gov.scot/resource/doc/226155/0061245.pdf, June 2008.

C. C. Selby. Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. In *Proceedings of the 10th Workshop in Primary and Secondary Computing Education (WiPSCE'15)*, pages 80–87, 2015.

S. Sentance and A. Csizmadia. Computing in the curriculum: Challenges and strategies from a teachers perspective. *Education and Information Technologies*, 22(2):469–495, 2017.

S. Sentance, M. Dorling, A. McNicol, and T. Crick. Grand Challenges for the UK: Upskilling Teachers to Teach Computer Science Within the Secondary Curriculum. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE 2012)*, pages 82–85, 2012.

S. Sentance, M. Dorling, and A. McNicol. Computer science in secondary schools in the UK: ways to empower teachers. In *Proceedings of the 6th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP'13)*, pages 15–30, 2013.

S. Sentance, S. Humphreys, and M. Dorling. The Network of Teaching Excellence in Computer Science and Master Teachers. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE 2014)*, pages 80–88, 2014.

N. Shadbolt. *Computer science degree accreditation and graduate employability: Shadbolt review*. Department for Business, Innovation & Skills, UK Government, May 2016.

Simon, S. Fincher, A. Robins, B. Baker, I. Box, Q. Cutts, M. de Raadt, P. Haden, J. Hamer, M. Hamilton, R. Lister, M. Petre, K. Sutton, D. Tolhurst, and J. Tutty. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Conference on Computing Education*, pages 189–196, 2006.

Simon, A. Clear, J. Carter, G. Cross, A. Radenski, L. Tudor, and E. T. onisson. What's in a Name?: International Interpretations of Computing Education Terminology. In *Proceedings of the 2015 Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR'15)*, pages 173–186, 2015.

B. Simon and B. Hanks. First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing*, 7(4), 2008.

B. Simon, M. Kohanfars, J. Lee, K. Tamayo, and Q. Cutts. Experience report: peer instruction in introductory computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*, pages 341–345, 2010.

L. Snyder. Status update: high school CS internationally. *ACM Inroads*, 3(2):82–85, 2012.

J. Sorva. Notional machines and introductory programming education. *ACM Transactions on Computer Science Education*, 13(2):1–31, 2013.

C. W. Starr, B. Manaris, and R. H. Stalvey. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin*, 40(1):261–265, 2008.

D. Stevenson. Information and Communication Technology in UK schools: An independent inquiry. Technical report, The Independent ICT in Schools Commission, 1997.

L. Sturman and J. Sizmur. International Comparison of Computing in Schools. Technical report, National Foundation for Educational Research, 2011.

J. Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257285, 1988.

B. Tangney, E. Oldham, C. Conneely, S. Barrett, and J. Lawlor. Pedagogy and Processes for a Computer Programming Outreach Workshop – The Bridge to College Model. *IEEE Transactions on Education*, 53(1):53–60, 2010.

M. Tedre and P. J. Denning. The Long Quest for Computational Thinking. In *Proceedings of the 16th Koli Calling Conference on Computing Education Research*, pages 120–129, 2016.

A. Theodoropoulos, A. Antoniou, and G. Lepouras. How Do Different Cognitive Styles Affect Learning Programming? Insights from a Game-Based Approach in Greek Schools. *ACM Transactions on Computer Science Education*, 17(1)(3), 2017.

UK Digital Skills Taskforce. Digital Skills for Tomorrow's World. http://www.ukdigitalskills.com/wp-content/uploads/2014/07/Binder-9-reduced.pdf, July 2014.

UKForCE. The new computing curriculum: Terminology and definitions. Technical report, UK Forum for Computing Education, June 2016. http://ukforce.org.uk/wp-content/uploads/2016/09/UKForCE-definitions.pdf.

J. Vahrenhold. On the importance of being earnest: challenges in computer science education. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE'12)*, 2012.

M. J. Van Gorp and S. Grissom. An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming. *Computer Science Education*, 11(3):247–260, 2001.

I. Vekiri. Information science instruction and changes in girls' and boy's expectancy and value beliefs: In search of gender-equitable pedagogical practices. *Computers & Education*, 64:104–115, 2013.

A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*, pages 93–98, 2011.

L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978.

M. Webb, N. Davis, T. Bell, Y. J. Katz, N. Reynolds, D. P. Chambers, and M. M. Sysło. Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies*, 22(2):445–468, 2017.

E. Wenger. *Communities of Practice: Learning, Meaning, And Identity*. Cambridge University Press, 2000.

L. A. Williams and R. R. Kessler. All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*, 43(5):108–114, 2000.

S. Willman, R. Lindéna, E. Kaila, T. Rajala, M.-J. Laakso, and T. Salakoski. On study habits on an introductory course on programming. *Computer Science Education*, 25 (3), 2015.

F. Wilson. Can compute, won't compute: women's participation in the culture of computing. *New Technology, Work and Employment*, 18(2):127–142, 2003.

G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson. Best Practices for Scientific Computing. *PLoS Biology*, 12(1):e1001745, 2014.

J. M. Wing. Computational Thinking. *Communications of the ACM*, 49(3):33–36, 2006.

J. M. Wing. Computational Thinking and Thinking About Computing. *Philosophical Transactions of the Royal Society A*, 366(1881):3717–3725, 2008.

J. M. Wing. Computational Thinking: What and Why? https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf, November 2010.

L. E. Winslow. Programming pedagogy – a psychological overview. *ACM SIGCSE Bulletin*, 28(3):17–22, 1996.

A. Wolf. Review of vocational education: The Wolf report. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/180504/DFE-00031-2011.pdf, March 2011.

A. Yadav, C. Mayfield, N. Zhou, S. Hambrusch, and J. T. Korb. Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, 14(1)(5), 2014.

D. Zingaro, C. Bailey Lee, and L. Porter. Peer instruction in computing: the role of reading quizzes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*, pages 47–52, 2013.