Concepts in K–9 Computer Science Education

Erik Barendsen
Radboud University &
Open University, Netherlands
e.barendsen@cs.ru.nl

Linda Mannila
Åbo Akademi University
Turku, Finland
linda.mannila@abo.fi

Barbara Demo
University of Turin
Turin, Italy
barbara@di.unito.it

Nataša Grgurina
University of Groningen
Groningen, Netherlands
n.grgurina@rug.nl

Cruz Izu
University of Adelaide
Adelaide, Australia
cruz@cs.adelaide.edu.au

Claudio Mirolo
University of Udine
Udine, Italy
claudio.mirolo@uniud.it

Sue Sentance
King's College London
London, UK
sue.sentance@kcl.ac.uk

Amber Settle
DePaul University
Chicago, USA
asettle@cdm.depaul.edu

Gabrielė Stupurienė
Vilnius University
Vilnius, Lithuania
gabriele.stupuriene@mii.vu.lt

ABSTRACT

This exploratory study focuses on concepts and their assessment in K–9 computer science (CS) education. We analyzed concepts in local curriculum documents and guidelines, as well as interviewed K–9 teachers in two countries about their teaching and assessment practices. Moreover, we investigated the 'task based assessment' approach of the international Bebras contest by classifying the conceptual content and question structure of Bebras tasks spanning five years. Our results show a variety in breadth and focus in curriculum documents, with the notion of algorithm as a significant common concept. Teachers' practice appears to vary, depending on their respective backgrounds. Informal assessment practices are predominant, especially in the case of younger students. In the Bebras tasks, algorithms and data representation were found to be the main concept categories. The question structure follows specific patterns, but the relative frequencies of the patterns employed in the tasks vary over the years. Our analysis methods appear to be interesting in themselves, and the results of our study give rise to suggestions for follow-up research.

CCS Concepts

•**Social and professional topics → Computer science education; Student assessment; K-12 education;** *Computational thinking;*

Keywords

CS concepts; K–9 education; curricula; teachers; assessment; Bebras

ITICSE-WGR'15, July 04-08, 2015, Vilnius, Lithuania
© 2015 ACM. ISBN 978-1-4503-4146-2/15/07. . . $15.00
DOI: http://dx.doi.org/10.1145/2858796.2858800

1. INTRODUCTION

Computer science (CS) is no longer a subject area only relevant for a narrow group of professionals, but rather is a vital part of general education that should be available to all children and young people. CS "develops students' computational and critical thinking skills and shows them how to create, not simply use, new technologies. CS provides a fundamental set of concepts and skills needed to prepare students for the 21st century, regardless of their ultimate field of study or occupation."[1] Nevertheless, whereas digital literacy seems to have become a natural component of K–9 education, the extent to which CS concepts are included varies greatly, ranging from CS being a compulsory or elective subject to not being covered at all.

Various alternative terms are in use to indicate the foundational discipline or school subject called CS above, for example *Informatics* and *Computing.* In this report we will use the term CS uniformly, if necessary after explaining the local names of the subject. In contrast, *information technology* (IT) refers to usage of technical infrastructure and applications.

This report presents the results of an exploratory study. We are interested in the CS *content* in K-9, i.e., topics and ideas belonging to the subject matter, regardless of the specific skills or attitudes in which they appear. We will refer to these topics and ideas as *concepts.* In addition, we will focus on assessment of these concepts.

The aim of the report is hence to contribute to the discussion on what CS at K–9 level can entail. The report can hence be used to guide teachers, teacher educators, and curriculum developers in making informed decisions with regard to teaching and assessing CS concepts and skills for this particular school level, characterized by critical developments of pupils' cognitive abilities.

More specifically, we address a selection of concepts in K–9 CS education from a threefold standpoint: the *intended curriculum*, emerging from an assortment of national recommendations; the *implemented curriculum*, which results from teachers' actual practice; the *attained curriculum*, as

[1]"Why K-12 computer science?" on http://code.org and http://computinginthecore.org

attested by their assessment strategies. In addition, we analyze both the concepts and the style of assessment implied by the tasks assigned in the Bebras international competition. While focusing on the learning of CS concepts, indeed, it is likewise important to consider what exactly and how is assessed, in order to be able to make sense of the evidence supporting the achievement of a specific concept as well as to figure out the underlying characterization of the very idea of "CS concept."

We approach the two themes of CS concepts and assessment from three perspectives: curriculum standards, teachers' practice and international competitions. The analytic part of our research focuses on concepts and assessment in curriculum documents, teachers' practice, and Bebras tasks. Section 2 provides the background for the study by discussing the aspects included in our analytic part. The section also includes a brief review of the national situation of CS at K–9 level in seven countries. Our research questions are formulated in Section 3, followed by a more extensive description of the situations in the countries involved in our study in Section 4. Section 5 addresses our research methods. The results are organized according to the data sources in sections 6–8. The results are discussed in section 9, which ends with some final remarks.

2. BACKGROUND

2.1 CS at K–9 Level

As far as education in CS-related topics in the school is concerned, relevant documents and programs are most commonly addressed to the whole K–12 cycle than specific to K–9. We will mainly consider curricula with this broader scope.

A recent trend in educational discussions is an increased focus on the role and nature of CS in early education. Initiatives such as *Hour of Code*[2] and *Europe Codeweek*[3] are promoting CS and programming among educators, parents and students. Individual persons and large industries engage in active discussions acknowledging the importance of guaranteeing basic knowledge in CS for everyone. But if we are to teach CS in K–9, the question arises what exactly should be taught. After all, CS is a multifaceted field involving several dimensions in terms of concepts, capabilities and skills. A few proposals have been made to characterize suitable concept matter, e.g., by the organizations Computing at School (CAS) in England [2] and the Computer Science Teachers Association (CSTA) in the USA [81].

The extent to which programming may be beneficial to develop general problem-solving skills is still subject to debate, see, e.g., [75, 59]. Feurzeig et al. argue, for instance, that the need for rigorous thinking can be a reason to introduce programming in schools [35], whereas other authors stress the importance of creativity (e.g., [77]). An additional issue is how to achieve "language independence", often considered a desirable feature to assess programming competences [87].

If on the one hand several educators agree that programming is crucial for appreciating a computational perspective, a major objection is that it may be unsuitable for lower levels of education, as pointed out, e.g., in [63]. Duncan et al. remark, however, that it would be beneficial to shape the children's "attitudes to programming before it is too late" [33]. Programming is indeed implied by all sorts of artifacts, such as board games, visual programming tools, robots and various toy logic devices. Furthermore, even the scope of programming is now broader than it used to be: its practice can be seen as a means of self-expression and social participation [54, 80], a component of a new form of literacy [18, 93], a tool for developing creativity [12], a way to widen experience and experiment with personal ideas [11], and an instrument to foster children's metacognition [76].

Before high school, and especially at the primary level, CS ideas can also be introduced in the classroom through *unplugged*, or partly unplugged, approaches. This is the perspective of the *CS Unplugged* project [7], that has then inspired several educators and that we can also find in other projects such as *Informatik erLeben* [69] and *Abenteuer Informatik* [40].

Regardless of the approach or tools used, Hubwieser et al. point out that "there is a convergence towards *computational thinking* as a core idea of the K–12 curricula" and that "*programming* in one form or another, seems to be absolutely necessary for a future oriented CSE" [50] (added emphasis).

Computational thinking (CT) was probably first introduced in connection with children's education by Seymour Papert in his book *Mindstorms*, while referring to the initial attempts to "integrate computational thinking into everyday life" [72] (p. 182), and again in [73], a paper about the learning of mathematical ideas. More recently, Jeannette Wing re-introduced the category of CT in her column [97], that then turned out to be broadly influential. She later defined it as those "thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent" [98], a way of thinking characterized by different layers of abstraction. Although there is not full agreement on how to define CT, a few organizations have tried to come up with a more accurate characterization, notably [37, 38].

The CSTA [23] and the ISTE [53] have jointly developed a definition of CT suitable for use in K–12 education, identifying nine essential concepts: data collection, data analysis, data representation, problem decomposition, abstraction, algorithms, automation, parallelization and simulation. Skills related to these concepts are not limited to CS, nor even to STEM, but can be practiced and developed within all disciplines, which is crucial for broadening participation. The ISTE has also proposed an operational definition for CT as a problem-solving process with CS features.

Among other contributions coming from educators, Lee et al. [60] identify abstraction, automation and analysis as the key features in order for young pupils to deal with novel problems. They then suggest three possible approaches to introduce CT for the K–8 levels. As observed by Hu, however, we should be careful since if "the mainstream of computational thinking is thinking about process abstraction, then Jean Piaget's Stages of Cognitive Development may suggest that this thinking skill cannot be effectively taught until adolescence age" [49].

Further approaches to infuse CT in early education in the light of learning theories are also considered, in for instance [22, 49] and a broad survey of the CT perspective in the context of K–9 education can be found in [65]. Due to its cross-curricular nature and allround skill set, computational thinking (CT) constitutes an interesting perspective for K-9

[2]http://hourofcode.com
[3]http://codeweek.eu/

CS education.

Following the approach suggested in [65], the definition of CT used in this report is the one developed by the CSTA and the ISTE, which introduces the nine foundational concepts mentioned above.

2.2 Curricula for K–9 CS Education

While the interest surrounding CS at K-9 level is relatively new, CS at K-12 level has been a topic for discussion for a longer period of time. The first ACM model curriculum for K–12 CS was presented in 1993 and updated in 2003 [91]. The latter states that K–12 curricula should include "programming, hardware design, networks, graphics, databases and information retrieval, computer security, software design, programming languages, logic, programming paradigms, translation between levels of abstraction, artificial intelligence, the limits of computation [...], applications in information technology and information systems, and social issues."

More recently, the CSTA has developed the CSTA K–12 CS Standards [81]. These standards contain five strands: Computational thinking, Collaboration, Computing Practice and Programming, Computers and Communication Devices, and Community, Global and Ethical Impacts. The Exploring CS model curriculum [43] covers six areas: Human-Computer Interaction, Problem Solving, Web Design, Introduction to Programming, Computing and Data Analysis, and Robotics. As K-9 is part of K-12, these guidelines can also be considered part of the discussion on what to teach at K-9 level.

CS has also been introduced in official national K-9 curricula in several countries [34]. The situation in the countries represented by the authors of this report are summarised below:

Australia: The "F–10 Australian Curriculum: Technologies", where F-10 stands for "from Foundation to 10 grade", was fully developed and published on the Australian Curriculum website[4] by the end of 2013. This curriculum covers areas ranging from logical sequencing and simple problem solving to programming, projects and societal aspects.

Finland: Finland will get a new national curriculum for general education (grades 1–9) in fall 2016[5], where special attention has been paid to recognizing future competence needs. The document emphasizes the need for students to acquire basic knowledge about information technology and also includes programming, as a means to build understanding for central concepts and principles of how the technology lying behind the increased digitalization works and how to use this technology to create artefacts of their own.

Italy: The Italian curriculum includes topics referring to two rather broad areas: (1) A cross-disciplinary key citizenship *digital competence* area, including, for instance, proficiency and critical attitude in the use of ICT for work, life and communication. This area spans the whole period of compulsory education. (2) A general *technology* subject area, that includes, for instance, use of ICT tools, programming and societal aspects.

Lithuania: As part of an education reform in 1997, the Lithuanian core curriculum in Informatics (indicated as CS from now on) went through a major revision, which resulted in less focus on CS topics. Lithuanian pupils do however get familiar with basic CS concepts and skills in grade 5 or 6, when they take a *Logo* or *Scratch* course. For grades 9 or 10 there are three optional modules: Programming, Web design and Desktop publishing.

Netherlands: The learning objectives for primary education in the Netherlands (ages 4–12) are summarized in 58 general core objectives describing goals for the Dutch, Frisian, and English languages, arithmetic/math, world and personal orientation, arts and physical education [55]. Only 10 of these objectives contain aspects of CT. Recently, initiatives have been employed to explore the possibilities of introducing elements of Computer Science into K-9 education [89].

UK/England: England introduced a new subject called *Computing* in September 2013 [31], after a period of consultation following the disapplication of the previous ICT curriculum in January 2012. This was implemented as part of a revised National Curriculum for all subjects in September 2014. Computing has three elements: CS, Information Technology and Digital Literacy.

United States: There is no national curriculum for K-9 CS in the U.S., in part because the curricular standards have to be approved at the state level. Many states in the U.S. follow the Common Core Standard[6]. Several groups have worked to put together curricula for K–12 that could be used by U.S. teachers (e.g. the CSTA and Code.org).

Nevertheless, although there are several commonalities in the informal and national curricula, there is still no consensus with regard to what teaching CS in K–12 means. A general picture about the state of CS education in several countries is drawn in the special issue of TOCE on *Computing Education in K–12 Schools* [66].

2.3 Approaches to Assessment

Valid assessment is a crucial part of successful teaching and learning activities. A major issue, however, is to design assessment instruments that can be validated to actually assess the intended learning outcomes (see, e.g., constructive alignment [9]). In Tew and Guzdial's words, while "many STEM disciplines have standard validated assessment tools [...], computer science does not have" similar "tools, and practitioners must devise their own instruments each time they want to investigate student learning" [87].

As a matter of fact, assessment practices seem to vary a lot in the CS context. To this point, most assessment research in CS has focused on programming concepts [29, 67, 87, 96] and, for understandable reasons, mainly at tertiary level. Efforts have ranged from concept inventories [86] to

[4]http://www.australiancurriculum.edu.au/technologies/rationale

[5]http://www.oph.fi/ops2016/perusteet

[6]http://www.corestandards.org/

the use of taxonomies [82, 84, 90] and specialized exams. A cognitive assessment method for measuring problem solving and program development skills has been proposed by Deek et al. [30].

Another promising approach can be found in the international Bebras contest (www.bebras.org) in which CS concepts are addressed using compact, well designed tasks. An interesting question is how this "tasklet-based assessment" works as well as how it can be applied to a wider range of concepts. A more detailed introduction to Bebras is given below (Section 2.5).

Many K–12 teachers reportedly use projects or practical assignments [45]. This preference fits closely to the epistemic view of CS as an engineering discipline [8]. Such teaching methods clearly cover learning outcomes connected to, e.g., 'designing' or 'programming'.

Students are expected to use relevant concepts in design tasks. There is also strong evidence, however, that the relation between conceptual knowledge and designing is reciprocal. In science subjects, pupils who are involved in designing artefacts and are using relevant concepts and ways of scientific reasoning (e.g., from structure to function) would achieve deeper conceptual and technological understanding [28, 39, 58, 94]. In particular, students learn about CS concepts while working on programming assignments, e.g. [68]. It remains unclear, however, how to effectively assess and monitor development of conceptual understanding in practical contexts without disturbing the authentic design setting (by, e.g., letting students do a pencil and paper test).

Professional software developers constantly analyse and test their intermediate products [17]. Testing such preliminary results and explaining the reasoning steps appear to be crucial for the learning process [6, 56] and are potential starting points for assessment. Some promising exploratory experiments in CS have been carried out, both based on explanations and justification of intermediate products [46] and on the results themselves [95].

Several assessment instruments are based on the SOLO [10] and (revised) Bloom's taxonomies [1]. The former considers five levels of understanding: pre-structural, uni-structural, multi-structural, relational and extended abstract. The latter addresses the abilities to remember, understand, apply, analyze, evaluate and create, as well as the knowledge dimensions of factual knowledge, conceptual knowledge, procedural knowledge and metaknowledge. Meerbaum et al. devised an assessment instrument addressed to middle-school students learning to program in Scratch based on the SOLO and Bloom's taxonomies [68].

The relevance of devising assessment instruments is also pointed out by Werner et al., in that efforts "to engage K–12 students in" CT "are hampered by a lack of definition and assessment tools" [95]. The same is likely to hold also for K-9 level. Among the few attempts made in this direction, we can mention the framework in [12]. Moreover, according to Grover and Pea [47], without "attention to assessment," CT "can have little hope of making its way successfully into any K–12 curriculum."

2.4 Teacher Practice and Teacher Knowledge

Teachers' classroom practice is generally believed to be influenced by their knowledge and beliefs, e.g. [16, 74], as well as by their overall professional identity [71]. The relationship between knowledge and practice appears to be reciprocal: teacher knowledge is developed through an integrative process of action and reflection (cf. [79]).

The notion of *pedagogical content knowledge* (PCK) was introduced by Shulman [83] in order to try to describe the features of the teaching practice in a particular subject matter. In this author's view PCK is "the knowledge of teachers to help others learn", including "the ways of representing and formulating the subject that makes it comprehensible to others."

In the model by Magnusson et al.[64], four aspects of PCK with respect to a certain topic are distinguished: (a) knowledge about learning goals and objectives connected to the topic, (b) knowledge about students' understanding of the topic, (c) knowledge about instructional strategies for teaching the topic, and (d) knowledge about ways to assess students' understanding of the topic.

Eliciting PCK from teachers is not easy, however, since it tends to be tacit and usually the reasons motivating a particular instructional strategy are not explicitly articulated or shared with colleagues [62]. Several methods have been proposed, including interviews (e.g., [48]), Pedagogical experience Repertoires, PaP-eRs, [61], classroom observations (e.g., [5, 41]) and reflective journals (e.g., [99]).

As pointed out in [87], PCK has mainly been investigated for science education, whereas there are few studies in CS related contexts. However, a PCK perspective appears to be fruitful to explore the professional knowledge of CS teacher, as demonstrated by a large German project on the teaching of CS in schools [51], as well as by more focused investigations in the areas of programming [78, 3, 32] and UML design [57].

A promising instrument for the purpose of our study is the *Content Representation* (CoRe) format [61], aimed at capturing key ideas within a topic as well as the teachers' knowledge about each idea. According to the authors, indeed, two main elements characterize the PCK: the Content Representation (CoRe), i.e. an overview of the particular content taught, and something related to a teacher's Pedagogical and Professional-experience Repertoire (PaP-eR). This kind of instrument is based on eight questions that cover the PCK aspects addressed in [64]. In particular, Loughran et al. [62] originally introduced the CoRe format as an interview tool.

In the present study, questions from the CoRe tool appeared useful to elicit information about teachers' practice (and potentially their underlying knowledge and beliefs) with respect to *concepts* taught (cf. PCK aspect (a)) and *assessment* (cf. PCK aspect (d)).

2.5 The Bebras Contest

Bebras is an international contest on problem solving and CT, developed and initiated in 2004 in Lithuania[7]. The goal of the contest is to promote and raise interest in CS in general and CT in particular among teachers and students of all ages.

In practice this is done through local annual contests (mostly arranged online), where pupils of different ages are presented with attractive and compact tasks (sometimes called *tasklets*) that highlight one or more CS concepts. The intention is, however, that the tasks can be answered without prior knowledge of CS. In practice this indeed holds for the

[7]http://www.bebras.org

majority of tasks.

Some tasks are interactive (solved for instance by dragging and dropping), but most are multiple-choice questions. For the latter, the alternatives are carefully designed in order to reveal any potential (and expected) misconceptions. This *task based* assessment style employed in the Bebras contest is interesting in itself [19].

To solve Bebras tasks, participants use algorithmic concepts and are required to think about information, discrete structures, computation, and data processing. Each Bebras task can both demonstrate an aspect of CS and test the CS experience and ability of the participant. For this reason, the Bebras contest has also been studied as a basis for PISA-like assessment of CS competencies [52].

The tasks are categorized according to age groups and school levels as follows:

(0) Primary, 8-9 years (grade 3-4)

(I) Benjamin, 10-12 years (grade 5-6)

(II) Cadets, 13-14 years (grade 7-8)

(III) Junior, 15-16 years (grade 9-10)

(IV) Senior, 17-19 years (grade 11-13)

The Bebras contest follows a problem-solving approach to CS, observing two leading principles: (1) Problem solving is the individual capacity of using cognitive processes to compare and solve real, cross-disciplinary situations where the solution path is not immediately obvious [20], and (2) Interest and engagement are very important in problem solving [24, 25].

These principles have been translated into guidelines for task construction [19]. For example, tasks should present problems from various spheres of science and society, as close as possible to everyday life. They should stimulate thinking about efficient and effective use of applications of IT/ICT in everyday experience. Moreover, the influence of IT/ICT on culture and language should be emphasized, thus supporting the idea that cognitive, social, cultural and cross-cultural aspects are crucial in the use of technology.

The basic criteria for constructing Bebras tasks are: (1) the task can be solved within 3 minutes; (2) the problem statement is easy to understand; (3) the task can be presented in a single screen page; and (4) the task is independent from specific systems.

Tasks can be of different types, starting from the most common questions on IT/ICT and their applications in everyday life or including specific integrated problems related to history, languages, arts, and, of course, mathematics. It is also considered important to choose the problems so that the participants in the competition are not influenced by the digital tools (such as operating systems or the computer programs) they are experienced with.

The topics of the Bebras contest [26] are as follows:

- **Information (INF)** – conception of information, its representation (symbolic, numerical, graphical), encoding, encrypting;
- **Algorithms (ALG)** – action formalization, action description according to certain rules;
- **Computer systems and their application (USE)** – interaction of computer components, development, common principles of program functionality, search engines, etc.;
- **Structures and patterns (STRUC)** – components of discrete mathematics, elements of combinatorics and actions with them;
- **Social effect of technologies (SOC)** – cognitive, legal, ethical, cultural, integral aspects of information and communication technologies;
- **CS and information technology puzzles (PUZ)** – logical games, mind maps, used to develop technology-based skills.

This classification considers the tasks from the students' (or the "normal task-solver") point of view. Although other themes are not excluded, the contribution of CT concepts to Bebras topics is considerable.

3. AIM OF THE STUDY

In the analytic part of this study we investigate K–9 CS education from three curriculum perspectives (cf. [44, 92]). We will review the *intended* CS curriculum expressed in K–9 standards and curricula recommendations. Moreover, we will investigate the curriculum as it is actually *implemented*, by exploring the concepts covered in practice. Furthermore we address what is (provably) *attained*, by analyzing assessment practices in schools. Finally we investigate Bebras tasks, in particular to identify which concepts they address and what type of assessment is involved.

Our research questions are as follows:

1. Which concepts are present in K–9 CS curriculum documents?
2. Which concepts are taught in practice? Which assessment practices are used?
3. Which concepts are assessed in Bebras tasks? How can the assessment format of these tasks be characterized?

In order to set the stage for the study, we next present the context of the research and the methods used.

4. CONTEXT OF THE STUDY

We have addressed the research questions above in an exploratory case study involving (1) curriculum documents from England, the United States and Italy, (2) interviews with teachers from England and Italy, and (3) Bebras tasks from the contests between 2010 and 2014.

In Subsection 2.2 above, we briefly covered the main characteristics of CS education at K-9 level in the countries represented by the authors. Below, we give a more thorough review of the curricula used in England, the United Stated and Italy. This information is intended as background information to understand and interpret the curricula analysis and the context of the teacher interviews.

4.1 England

In England a Programme of Study for a new subject in the curriculum, *Computing*, was unveiled in September 2013, after a period of consultation following the disapplication of the previous ICT curriculum in January 2012. This was implemented as part of a revised National Curriculum for

UK Key Stage	1	2	3	4	5
Grades	1	2–5	6–8	9–10	11–12

Table 1: Key Stages

all subjects in September 2014. The changes in England have been well documented recently, for example [14, 15].

The CS curriculum for England has the following aims. Students:

- can understand and apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation
- can analyse problems in computational terms, and have repeated practical experience of writing computer programs in order to solve such problems
- can evaluate and apply information technology, including new or unfamiliar technologies, analytically to solve problems
- are responsible, competent, confident and creative users of information and communication technology [31]

The new National Curriculum is interesting itself as it is so short and there is no 'fleshing out'. Assessment levels have been removed. The Department of Education in England is seeking to increase teacher autonomy by being less prescriptive about how the curriculum is interpreted (in all subjects).

Computing has three elements: computer science (first two bullet points), IT (third bullet point) and Digital Literacy (fourth bullet point). One of the issues of the implementation of this curriculum is how to incorporate all three elements seamlessly. The three elements of Computing emanated from the Royal Society Report, *Shut Down or Restart* [88]. There is a strong emphasis towards computational thinking in the programme of study.[8]

The English curriculum is implemented in Key Stages. The relationship with other countries' grades system can be seen in Table 1.

There is a need for teacher development around the new curriculum because the introduction of a new subject has coincided with a political move towards a much less detailed curriculum.

The Computing curriculum in England is based on and heavily influenced by the Computer Science curriculum document produced by Computing At School in 2012, which was the result of two years' work by Computing At School (CAS) members around what should be taught in school if Computing was ever introduced. The CAS curriculum has influenced the new mandatory national programme of study in England, and the pace of change has been very rapid.

Scotland, Northern Ireland and Wales have their own Education departments and their own curricula. Scotland and Northern Ireland have their own awarding bodies. Scotland has a Computing Science curriculum which it has had for many years although this was reviewed and a new curriculum launched in 2010. Scotland's curriculum is very different to the English curriculum, whereas Wales and Northern Ireland teach ICT.

For readability and uniformity, we will use the term CS to refer to the Computing subject from now on.

4.2 Italy

Since 2007, the Italian school system has undergone a broad reform process, some aspects of which remain to be finalized. The reform is meant to change both the educational approach and the curricular organization. The duration of compulsory education in Italy is now up to 16 years of age, that usually corresponds to the 2nd high school year (grade 10). Since a major discontinuity arises at the transition from middle to high school, for the sake of this report we will focus on the first 8 grades. For grades K–8, CS and digital technologies are not in the scope of a specific subject, though some exposition to the ICTs should be part of a *Technologies* subject in the middle school (grades 6–8). The national curricular recommendations state that these contents should pertain to two rather broad areas:

- A cross-disciplinary key citizenship *digital competence* area: proficiency and critical attitude in the use of ICTs for work, life, communication; use of computer to retrieve, assess, retain, produce, present, share information as well as to cooperate through the Internet. This area spans over the whole period of compulsory education. (The Italian Ministry for Education has indeed adopted the "Recommendation of the European Parliament and of the Council" of 12/18/2006 on key competences for lifelong learning – 2006/962/EC.)
- A general *technology* subject area, that includes the use of the most common ICT tools and, "if possible," some computer and/or robot programming: proficiency and critical attitude toward the psychological, social and cultural impact of ICTs; if possible, introduction to programming with simple languages, to create and develop projects.

The reference to programming and robots is an attempt to acknowledge several (seemingly) successful experiences promoted by enthusiastic, self-motivated (as well as self-taught) teachers.

An independent CS subject, taught by qualified teachers, is only included in the curriculum of the first year(s) of scientific and technical high schools, which as already stated are not in the scope of this report. As to the basic competences at the end of compulsory instruction for the scientific-technological area, the national recommendations merely state that "[...] beyond the mastery of ICT tools, often acquired out of the school, it is necessary to develop a critical attitude [...] w.r.t. their social and cultural impact, some awareness of the relational and psychological implications of the way they are used, as well as of their effects for the environment and health; this crucial educational task is to be shared among the different disciplines" and that "whenever possible, students can be introduced to simple and flexible programming languages in order to develop a taste for creation and for the accomplishment of projects (interactive web sites, exercises, games, utility applications)

[8]The programme of study can be found in full (it only runs to 3/4 pages), at
https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

and in order to understand the relationships between source code and resulting behavior."

According to the general framework of the education of pre-service teachers, drawn in 2010, prospective primary school teachers, as well as middle school teachers of mathematics and sciences and of technology, will learn only some very basic digital literacy and are not prepared to properly deal with CS fundamental concepts.

A novelty for K–8 education in Italy is the three-year ministerial project "Program your future" launched in September 2014 with two main objectives:

1. to "provide schools with a set of simple, playful and easy-to-access tools in order for the students to learn basic computer science concepts" and

2. to "experiment the structural introduction of basic computer science concepts in the schools through programming" in playful contexts, this being the simplest and most enjoyable way to develop computational thinking. The activities are to a large extent drawn from *Code.org*, with some additional online assistance offered by CS teachers who volunteer to support their colleagues engaged in K–9 education.

The project's "ambition is that education in computational thinking will be introduced" as a curricular school subject. The objectives at the end of the third year include the involvement of 25% of the primary schools in the *Hour of Code* and of about 10% in some more advanced learning paths. This should be accomplished by sharing tutorials on selected CT topics (mostly algorithms and programming) and by encouraging the creation of communities where to find help and support. However, it rests almost entirely on the volunteering work of self-motivated teachers, and it is unclear if this initiative will be able to trigger long-lasting changes in K–8 CS education.

4.3 United States

There is no national curriculum for K-9 CS in the U.S., in part because the curricular standards have to be approved at the state level. Many states in the U.S. follow the Common Core Standard[9]. The Common Core includes standards for English and math[10] and in those standards there are mentions of technology literacy and mathematical reasoning skills that could be considered computational thinking. Details of this were provided in the working group report from 2014 [65].

Several groups have worked to put together curricula for K–12 that could be used by U.S. teachers. Among them, we can mention the CSTA and Code.org. Code.org has put together materials for K–5[11] as well as teacher training opportunities. They also have training opportunities and modules for CS in science and CS in math at the middle school level available[12]. The materials are intended to be aligned with the Common Core to make them more accessible for teachers. The CSTA materials are less detailed and include articles and activities developed for K–8[13].

[9]http://www.corestandards.org/
[10]http://www.corestandards.org/read-the-standards/
[11]http://code.org/educate/k5
[12]http://code.org/educate/curriculum
[13]http://csta.acm.org/Curriculum/sub/CSK8.html

There is a national standard of sorts at the high school level (grades 9–12) in the form of Advanced Placement computer science. As of 2016 there will be two AP computer science classes (one in Java programming and the other called CS principles that includes a lot of CT). Both offer students who complete the classes and earn a high enough grade on the associated test college credit.

5. METHOD

In this exploratory study, we have analyzed the conceptual content of curriculum documents and teachers' school practice, as well as the ways this content is assessed in school practice and the Bebras competition. To this end, we performed a document analysis, conducted interviews with K–9 teachers, and completed a conceptual analysis of a collection of Bebras tasks.

We will describe our method in more detail below, organized by the data sources used.

5.1 Curriculum Documents

In order to analyze the concepts and ideas, we constructed a classification of CS subjects into knowledge categories. Our classification is based on the 'knowledge areas' in the ACM/IEEE Computer Science Curricula report [91]. Although these guidelines are meant for higher education, the description of the content areas is useful for our classification, since they contain a recent overview of the field, certainly covering secondary school topics. Moreover the ACM/IEEE document contains detailed specifications of the knowledge areas, which was valuable for the analysis process.

We have clustered the knowledge areas into a conveniently small number of categories suitable to classify CS content for K–9 education, providing enough detail to distinguish variations in content. This approach has proved to be useful in analyzing teachers' survey responses with curriculum suggestions [4] and analyzing curriculum guidelines [85]. Our classification differs only slightly from the one used by Barendsen et al. [4].

Table 2 gives an overview of the knowledge categories, referring to the ACM/IEEE document for more detailed descriptions. Note that the knowledge area Software Development Fundamentals (SDF) is spread over four categories.

The documents analyzed in this preliminary report are

- CSTA curriculum [81], K–9 part
- CAS curriculum [2], K–9 part
- English (EN) national curriculum [31], K–9 part
- Italian (IT) guidelines [70], K–8 part

In the first phase, each document was subjected to open coding [21], extracting literal concepts and ideas from the curriculum texts. In the second (more axial, cf. [21]) coding phase similar codes were merged into one, slightly more abstract, code. Then the resulting codes were grouped into the general knowledge categories mentioned earlier. For the coding of the CAS and CSTA documents we have made use of Steenvoorden's [85] work.

To get a global idea about the focus of the documents, we looked at the number of occurences of codes in each category. We view the distribution of occurences over the categories as an indication of the relative importance of the categories.

knowledge category	*included ACM/IEEE knowledge areas*
Algorithms	Algorithms and Complexity (AL) Parallel and Distributed Computing (PD) Algorithms and Design (SDF/AL) **Remark:** concepts about data structures are covered by *Data*
Architecture	Architecture and Organization (AR) Operating Systems (OS) System Fundamentals (SF)
Modeling	Computational Science (CN) Graphics and Visualisation (GV)
Data	Information Management (IM) Fundamental Data Structures (SDF/IM)
Engineering	Software Engineering (SE) Development Methods (SDF/SE) **Remarks:** also contains ideas on collaboration; concepts without an engineering component are covered by Programming
Intelligence	Intelligent Systems (IS)
Mathematics	Discrete Structures (DS)
Networking	Networking and Communication (NC)
Programming	Programming Languages (PL) Platform Based Development (PBD) Fundamental Programming Concepts (SDF/PL)
Security	Information Assurance and Security (IAS) **Remark:** concepts about privacy are covered by Society
Society	Social Issues and Professional Practice (SP)
Usability	Human-Computer Interaction (HCI)

Table 2: Knowledge categories for curriculum analysis

We then explored the document contents through a more detailed analysis with respect to selected categories, using the frequencies and codes as pointers to relevant text segments.

5.2 Teacher Interviews

The aim of this part of the study was to establish some pointers to the nature of the *implemented curriculum* and the *achieved curriculum*, by conducting interviews about teachers' practice in terms of some of the concepts and ideas and also their assessment strategies with reference to these concepts.

To draw on some concrete and detailed examples, we decided to conduct a number of interviews with practising teachers, using the CoRe methodology described in section 2 [62], originally intended to identify aspects of pedagogical content knowledge [83]. For this exercise, it was important to focus on teachers where there was an actual curriculum in place at K–9; we surmised that working with a small number of teachers who were actively engaged with teaching CS with this age group was important to compare with what has been identified earlier in the *intended curriculum.*

Based on the analysis of curriculum documents, we selected three knowledge areas to focus on for the purpose of our interviews: *Algorithms*, *Programming* and *Security.* The two former ones are fundamental to CS and it was expected that many teachers would have experience from teaching these, hence providing us with comparable and rich data. Security was chosen as the third area since it is a rather new topic and we wanted to see if and in that case to what extent it is present in teachers' practice.

England was chosen for one group of teachers because the Programme of Study for CS [31] has been taught in schools across England since at least September 2014 and teachers can speak directly about their practice in delivering the intended curriculum. Italy was chosen as an alternative curriculum because there are teachers with experience of teaching at this level, despite the CS curriculum not being mandatory.

We interviewed the following teachers in England:

- 1 KS1 teacher (Grades K-1);
- 3 KS2 teachers (Grades 2-5);
- 1 KS3 teacher (Grades 6-8).

And correspondingly in Italy:

- 3 elementary school teachers (K-5);
- 3 middle school teachers of Mathematics and Science (K6-8), one of whom is also teaching CS topics in primary school;
- 2 high-school teachers, one with experience of teaching in elementary and middle school, the second being involved for several years in teacher training as well as in (K-13) educational projects.

Following the CoRe methodology [62], for each of the chosen categories, we presented the teacher with the concepts found in the preliminary analysis. The teachers were asked to indicate which concepts appear in their lessons and to select three "Big Ideas".

The CoRe methodology uses 8 questions for a concept to determine the nature and extent of a teachers' pedagogical content knowledge (PCK). Because the focus of the research was on concepts and assessments only, we considered only learning goals (questions 1–3) and assessment (question 8). We therefore omitted questions 4–7 which directly asked about teaching. Thus, our questions were the following:

CoRe Question 1:	*What do you intend the students to learn about these concepts?*
CoRe Question 2:	*Why is it important for students to know this?*
CoRe Question 3:	*What else do you know about this concept (that you do not intend students to know yet)?*
CoRe Question 8:	*What are the specific ways of ascertaining students' understanding or confusion around these concepts?*

Given the limited number of teachers that could realistically be interviewed, it was not possible to achieve a representative sample. Indeed, the K–9 range has many age

Year	Total tasks	Level 0-II
2010	139	65
2011	126	99
2012	124	84
2013	150	120
2014	129	101
All	**668**	**469**

Table 3: Overall number of Bebras tasks and corresponding number covering K-9 level

groups, and it is likely to be the case that teachers have different levels of content knowledge about CS. Instead, in this study we focused on two different national frameworks and interviewed a group of teachers with teaching experience covering a reasonable range of age groups. This in-depth approach provided a means to explore practices, issues and problems, and allowed us to test our PCK-based elicitation method.

In England the interview processes followed the ethics guidelines of the British Educational Research Association [13]. Teachers took part in interviews voluntarily and were provided with full information with respect to the study and use of their data. They gave permission for audio recordings to be made and transcribed. The process was less formal in Italy, but also the Italian teachers voluntarily accepted to participate in the interviews and had essentially the same preliminary information about the study and our approach.

The interview transcripts were used for a qualitative analysis. We used the concepts identified in the curriculum comparison analysis as initial codes in an analytic coding process [42] and added new codes where necessary until a complete code system was obtained. The teachers' responses related to assessment were analyzed by an inductive approach starting with an open coding phase. The open codes were then grouped into more general types of assessment, thus obtaining the final analytic codes [21, 42].

5.3 Bebras Task Analysis

For this report we considered Bebras tasks between 2010 and 2014. We obtained all of the recommended and elective tasks for each of those years, and Table 3 shows the total number of tasks by year as well as the breakdown by level of task relevant to this report. All tasks from the five years have been analysed.

In this work we are interested in considering two things regarding the Bebras tasks:

- What type of concepts are assessed in the tasks?
- What type of assessment is used in the tasks?

Because of the recent interest in CT we focus on using CT terms to classify the Bebras tasks. Regarding assessment it is important to keep in mind that a large part of Bebras tasks are multiple-choice questions. Using multiple-choice questions to assess CT concepts can be challenging so we consider the issue of how the Bebras contest organizers have structured the questions.

5.3.1 *Classification of CT Concepts*

When classifying the type of CT found in each Bebras task, we used a deductive process. The nine concepts as defined by the ISTE and CSTA (see section 2) were used as the starting point of an analytic coding procedure (cf. [21, 42]). Team members independently coded each Bebras task assigning one or more concepts. This coding was then reviewed by another team member, who marked any disagreements regarding the classification. The disagreements were discussed until the conflicts could be resolved. The result of the discussion was the production of an operational definition of each CT term. Table 4 gives for each CT concept a short operative description used to identify where the concept applies in particular Bebras tasks.

In most cases, the operational definitions we use are identical or nearly identical with the characterizations provided by the CSTA and the ISTE. In some cases we provided more elaboration on the concepts, which may have extended the tasks to which the terms can be applied. There were no tasks that introduced CT terminology not represented in the CSTA and ISTE document. However, it should be noted that Bebras tasks sometimes address pure ICT literacy, and those questions were marked as such.

5.3.2 *Classification of Task Structure*

When classifying the structure of the tasks, we decided to focus on one CT concept, namely algorithms. There were several reasons for this. First, the term is broad and allows the inclusion of a variety of tasks. However, since so many Bebras tasks involved algorithms in one form or another, it was necessary to limit the scope of the classification in order to be feasibly completed. We therefore limited our classification to Bebras tasks that involved only the category of algorithms. Tasks that were classified with multiple CT terms were not considered.

We used an inductive process (cf. analytic coding, [21, 42]) when considering the structure of Bebras tasks involving purely algorithms concepts. A member of the team read each task in the relevant years and produced a classification for the questions, including a description for the classification. The classification scheme was discussed with other team members and slightly refined before all of the tasks were classified (Table 5). The classification of each relevant task was completed by one team member and then reviewed by at least one other team member. Conflicts were resolved during a discussion period before the final classification for the task was determined.

6. RESULTS: CURRICULUM DOCUMENTS

The distribution of code occurrences found in the documents is displayed in Table 6. These absolute numbers reflect the respective sizes of the documents. For example, the English national and Italian documents are written in a more compact style than the CAS curriculum. To facilitate comparisons, table 7 gives the relative weights of the respective categories. The distribution of concept occurrences for the K–9 documents is visualized in Figure 1.

The global concept distribution suggests that all four K–9 documents give substantial attention to algorithmic aspects, especially CAS, EN and IT. Programming is seen in the documents in comparable fractions. The engineering aspect is absent in the Italian guidelines, and does not play an important role in EN either. CSTA seems to have more emphasis on societal aspects than the other two documents. For instance in CAS, societal aspects are not very prominent, in favour of the more technical aspects (Engineering,

CT category	Definition	Operational definition for Bebras
Data collection	The process of gathering appropriate information.	Find a data source for a problem area.
Data analysis	Making sense of data, finding patterns, and drawing conclusions.	Take data and transform it to solve a problem. Often there is some statistical analysis involved in the transformation, although the statistics do not have to be sophisticated.
Data representation	Depicting and organizing data in appropriate graphs, charts, words, or images.	Take data and put it into a specified format. Includes descriptions of data that involve particular structures. It may involve understanding the implications of graphs or other representations on the solution of a problem.
Problem decomposition	Breaking down tasks into smaller, manageable parts.	Breaking a problem or task into smaller pieces to enable an easier or better solution.
Abstraction	Reducing complexity to define main idea.	Problems that ask for the creation of a formula. The distillation of broader ideas out of narrower concepts. Finding rules that apply to a given problem. Finding a pattern to model some behavior. Identifying essential facts about a structure or problem to verify correct answers.
Algorithms & procedures	Series of ordered steps taken to solve a problem or achieve some end.	Solving maximization, minimization, or other optimization problems. Following a step-by-step procedure. Verifying potential solutions as valid or invalid. Encoding or encryption/decryption problems, including the application of an encryption scheme to a sample set of data. Debugging solutions and finding errors in a solution. Applying a set of rules to determine specific values. Choosing or verifying pseudocode or code.
Automation	Having computers or machines do repetitive or tedious tasks.	No instances found.
Parallelization	Organize resources to simultaneously carry out tasks to reach a common goal.	Scheduling problems.
Simulation	Representation or model of a process. Simulation also involves running experiments using models.	Tasks that are interactive and involve building and exploring a solution.

Table 4: Operational description of CT concepts in Bebras tasks.

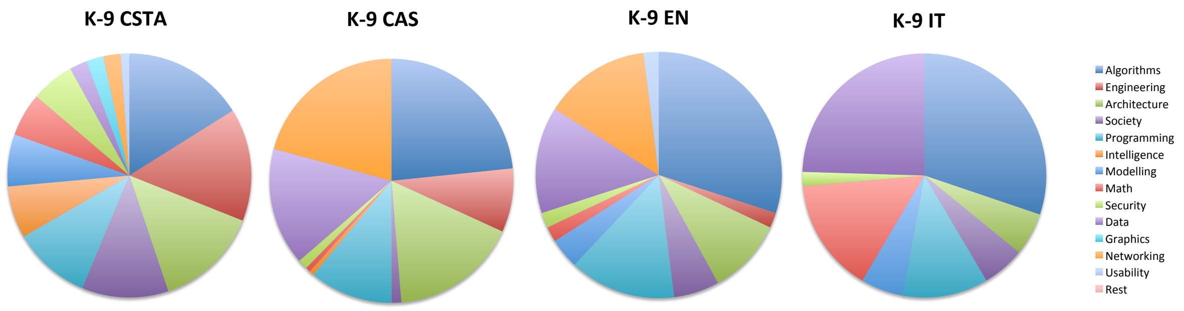

Figure 1: Visualization of concept distributions.

Question structure	**Description**
Constraint	A description of some rules (possibly with a diagram about the rules) and the addition of a constraint on those rules along with a listing of possible scenarios that achieve that constraint
Formula identification	A description of a problem that involves a formula and a question that asks for a specific answer involving the underlying formula
Optimization	A set of rules (and possibly a diagram relevant to those rules) along with a optimization question (minimize or maximize) and then a listing of possible values
Ordering	A list of objects and properties of objects with a definition of the relationship between those properties and then a listing of possible orderings of the objects
Procedures	A set of procedures and a situation involving the procedures along with a goal to achieve or a set of commands given and then a listing of the possible ways the goal can be achieved or the results that the commands produced. It may involve debugging the procedures
Sequencing	A description of a situation along with a sequence of actions that occur in that situation and then a list of possible results of the sequencing
Verification	A description of a problem and then a listing of possible solutions to the problem with a request to verify which is correct/incorrect

Table 5: Classification categories for Bebras tasks.

	CSTA	CAS	EN	IT
Algorithms	14	36	15	16
Engineering	13	13	1	0
Architecture	12	26	5	3
Society	10	2	3	3
Programming	9	17	7	6
Intelligence	6	1	0	0
Modelling	6	0	2	3
Mathematics	5	1	1	8
Security	5	2	1	1
Data	2	24	7	13
Graphics	2	0	0	0
Networking	2	32	7	0
Usability	1	0	1	0
Rest	0	0	0	0
Total	*87*	*154*	*50*	*53*

Table 6: Occurrences of codes within the knowledge categories

	CSTA	CAS	EN	IT
Algorithms	16%	23%	30%	30%
Engineering	15%	8%	2%	0%
Architecture	14%	17%	10%	6%
Society	11%	1%	6%	6%
Programming	10%	11%	14%	11%
Intelligence	7%	1%	0%	0%
Modelling	7%	0%	4%	6%
Mathematics	6%	1%	2%	15%
Security	6%	1%	2%	2%
Data	2%	16%	14%	25%
Graphics	2%	0%	0%	0%
Networking	2%	21%	14%	0%
Usability	1%	0%	2%	0%
Rest	0%	0%	0%	0%

Table 7: Distribution of codes over the knowledge categories (relative frequencies)

Networks). These categories appear to be the main differences between CAS and EN. Hardware (architecture) and Networks receive relatively much attention in the CAS curriculum, whereas the Mathematics contribution in IT appears to be large compared to others (e.g.: *truth values and propositions, language of logic and probability, links with set theory, geometry, mathematical models...*).

Below, we will explore the **Algorithms** category in some detail and discuss a selection of the other categories in a more global way.

The codes assigned in this category during the second phase are:

CSTA: algorithm, search algorithm, algorithm sharing, instruction set, abstraction, multiplicity, information sharing, complexity, decomposition, instruction sequence, resource, sort algorithm, parallelization.

CAS: input, instruction, task, sequence, steps, multiplicity, repetition, problem solving, algorithm representation, concurrency, ambiguity, decision, selection, component, abstraction, data processing, algorithm, output, precision, decomposition, instruction set.

EN: abstraction, algorithm, data processing, decomposition, input, instruction, output, problem solving, repetition, searching, selection, sequence, sorting.

IT: sorting, sequencing, plan description, order, problem solving, procedures, decision trees, algorithmic procedures, top down, problem solving trees, finding paths in graph, algorithm, combinatorial algorithms, problem formalization, problem decomposition.

The K–9 documents mention algorithmic building blocks (such as steps, sequence, choice, selection):

"Algorithms are sets of instructions for achieving goals, made up of pre-defined steps" (CAS);

"Algorithms can include selection (if) and repetition (loops)" (CAS);

"Describe and analyze a sequence of instructions being followed" (CSTA).

Algorithms are connected with problem solving aspects such as decomposition and abstraction:

> *"Problem formalization and problem decomposition into subproblems"* (IT);
>
> *"Use abstraction to decompose a problem into sub problems"* (CSTA).

The CSTA and Italian document moreover indicate specific types of algorithms, such as searching and sorting:

> *"Act out searching and sorting algorithms"* (CSTA);
>
> "[. . .] *design of simple combinatorial algorithms"* (IT);
>
> *"Simple algorithmic procedures (sorting, calculating, logical relationships in real situations)"* (IT).

For the English national curriculum, searching and sorting are optional subjects.

Finally, the CSTA and CAS curricula refer to parallelization aspects:

> *"Describe the process of parallelization as it relates to problem solving"* (CSTA);
>
> *"Computers can 'pretend' to do more than one thing at a time, by switching between different things very quickly"* (CAS).

The Italian guidelines do not contain **Engineering** aspects such as specifications, debugging and testing. The CAS document appears to focus on 'technical' aspects such as requirements, verification, and testing:

> *"Programs are developed according to a plan and then tested. Programs are corrected if they fail these tests"* (CAS).

The CSTA curriculum focuses on general aspects (designing, evaluating) and emphasizes development of artifacts in a team:

> *"Problem statement and exploration, examination of sample instances, design, implementing a solution, testing, evaluation"* (CSTA);
>
> *"Collaboratively design, develop, publish, and present products"* (CSTA);
>
> "[. . .] *using collaborative practices such as pair programming* [. . .]*"* (CSTA).

With respect to **Programming**, CAS, EN and CSTA mention program elements like variables and control structures, CSTA being the most elaborate of the three:

> *"The difference between constants and variables in programs"* (CAS);
>
> "[. . .] *work with variables"* (EN);
>
> *"They can use a variety of control structures"* (CAS);
>
> *"Implement problem solutions using a programming language, including: looping behavior, conditional statements, logic, expressions, variables, and functions"* (CSTA).

The CAS curriculum points at different kinds of programming errors:

> *"Understanding the difference between errors in program syntax and errors in meaning"* (CAS).

The Italian guidelines do not explicitly mention program elements, but only contain suggestions on a higher language level, such as "*pseudocode*". None of the K–9 documents refers to the concept of *recursion*.

In the **Data** category, the CAS curriculum stresses the distinction between *data* and *information*:

> *"There are many different ways of representing a single thing in a computer"* (CAS);
>
> *"Many different things may share the same representation"* (CAS).

The CSTA curriculum refers to representation details only in the grade 9–12 part. However, K–9 students should be able to:

> "[. . .] *represent data in a variety of ways"* (CSTA).

The Italian guidelines approach this category from the point of view of application areas and users:

> *"Representation of knowledge: building and reading double-entry tables"* (IT);
>
> *"Simple notions about the digital representation of non- textual information (sound, images, etc.)"* (IT);
>
> *"Linguistic applications: inflection and concordance tables (nouns and adjectives, articles and prepositions, articles and prepositions)"* (IT);
>
> *"Family trees"* (IT).

Several aspects related to CS and **Society** are covered in the K–9 curricula. The CSTA curriculum is most explicit, including technology impact, career, ethical and legal issues, and privacy:

> *"Identify interdisciplinary careers that are enhanced by computer science"* (CSTA);
>
> *"Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing)"* (CSTA).

The CAS and English national curriculum refer to societal impact in a more global way, including general keywords such as ethics and privacy:

> *"Social and ethical issues raised by the role of computers in our lives"* (CAS).
>
> "[. . .] *including protecting their online identity and privacy"* (EN).

The Italian guidelines and English national curriculum concern responsible use of the internet:

> *"Rules and guidelines for a responsible and correct use of the information available on the web; netiquette for web navigation and e-mail"* (IT).
>
> "[. . .] *recognise inappropriate content, contact and conduct, and know how to report concerns"* (EN).

7. RESULTS: TEACHER INTERVIEWS

7.1 England

Teachers were asked for some basic information about their teaching experience and then presented with a list of topics taken directly from the Algorithms and Programming sections of the National Curriculum [31] at KS1–KS3. No questions were asked about security as this is not included in the English Programme of Study in the form categorised as security within the ACM curriculum. There are some elements of privacy and internet safety within the English Programme of Study that do not fall into the security category as defined in the above section on curriculum. Table 8 gives an overview of the teachers initially interviewed in England. Between them they teach the whole K–9 curriculum but none of them teaches all of it. They have a variety of backgrounds in CS. Anna, Beatrice and Fiona are class teachers in primary school and teach CS as one of many other subjects; David is a CS coordinator in a primary school who teaches all the children from 4–11 in the school, mostly team teaching with the class teacher. Eliza is a secondary school teacher who has been teaching CS to children from 16–18 for 11 years and has gradually introduced CS lower down the school over the last few years as the curriculum has changed.

Teachers were asked to look at a list of topics provided from the CS programme of study and asked to identify what they do and do not teach and this information is shown in Table 8. They were also asked to select three topics that they were happy to talk about in more depth, and these are indicated by shading in the table. It would be expected that Eliza teaches at least the bottom half of the list and that Anna only teaches a few items from the beginning of the list, as the list moves from KS1 to KS3. On the whole this is true, although David claims to teach the material that is in the curriculum for 11–14 year olds although he teaches in a primary school. The curriculum includes the fact that students should learn about computational abstractions at Key Stage 2 (age 7–11) and some of the teachers were not sure what this meant.

Teachers were asked to choose topics that they felt they wanted, or felt confident, to talk about. Optionally they were able to elect their own topics to talk about — following the idea of Big Ideas from the CoRe methodology (Loughran et al, 2014). From the table it can be seen that 3 of the 5 teachers wanted to talk about creating simple programs, and three out of 5 wanted to talk about either debugging programs or detecting errors in algorithms. This indicates firstly that these areas are those they feel they are competent to talk about, but also has the side-effect of skewing our data to be in the area of simple programming and debugging. It is therefore not surprising that all teachers then mentioned debugging when it came to asking them the CoRe questions.

One of the teachers with a strong background in CS chose her own areas — Decomposition and Algorithms Design and Planning — as the ones she wanted to discuss, according to the CoRe methodology.

When coding the data we used the codes that were already established by the examination of the curricula from different countries. The teachers then responded to the questions in the areas that they had chosen and Table 9 shows the occurrence of different themes in their data, once coded. We added more categories as we came across them, but in essence our new categories were around teachers' beliefs and practices relating to pedagogy (these are discussed later in this section).

Teachers' Characteristics

The teachers interviewed had different backgrounds and attitudes towards the new curriculum.

Anna is an experienced teacher who not only teaches her own class, but has also participated in national initiatives to support primary teachers in England. Based on her extensive experience as IT developer, Anna holds firm beliefs on importance of teaching rigorous work flow, (represented by, e.g. the concept of decomposition) at the outset in Grades K–5. For each problem, she wants her students to *"slow down, break it up, and look at each bit"* rather than *"just bash the buttons"* or *"just do things"*. She teaches her students concepts she considers important for all of CS while considering what the students will learn in subsequent key stages.

Beatrice is relatively new to teaching but enthusiastic about learning to teach CS:

> *"As a new teacher I am still learning a lot about these topics. I am the only teacher in my school who understands anything about CS so I don't think there is a lot of confidence around about teachers in schools."*

Beatrice does not know anything more about the subjects she discussed than her KS2 students. When asked about the importance of learning about these concepts, she reiterates the importance of learning the CS concepts well and emphasizes the importance for students to understand them in order to be better equipped in their everyday life. She does not refer to the CS curriculum as a whole and the groundwork that is being laid in KS2 for the coming CS education in subsequent key stages.

David is a specialist primary CS teacher who has responsibility for CS teaching across the whole school. The model adopted at his school is for him to "team teach" with class teachers so that they can learn to teach CS from his model. He has expertise beyond what he is teaching the chldren and believes that an important aspect of delivering the curriculum content is resilience children can potentially build up through learning to debug. He also uses a range of assessment techniques to try to capture the learning of the students.

Eliza is an experienced CS teacher who has an industry background and has taught CS up to grade 12 for 11 years. Eliza has a clear picture of the whole of the CS curriculum for all the key stages and beyond. She intends to equip her students well for the learning and understanding of CS in coming key stages and plans and designs her teaching accordingly. In her words,

> *"I think it is important that you don't miss any foundations in their training because that's what creates the gaps"* (Eliza).

Fiona is an experienced teacher who is relatively new to CS but is undergoing training to become a primary CS specialist. While Fiona considers it important to teach her students good problem solving skills, she is not sure as to what exactly to teach them:

Name	**Anna**	**David**	**Beatrice**	**Fiona**	**Eliza**
Age group taught	4-7	4-11	7-11	7-11	11-14 (18)
CS knowledge (Strong/Middle/Weak)	S	M	W	W	S
Hours teaching CS/week	1	10+	2–3	1	10+
Years teaching	10	6	2	10	11
Years teaching CS	10	2	2	2	11
Algorithms					
Implementing algorithms as programs	Yes	Yes	Yes	Yes	Yes
Following precise and unambiguous instructions	Yes	Yes	Yes	Yes	Yes
Using logical reasoning to explain how simple algorithms work	Yes	Yes	Yes	Yes	Yes
Detect and correct errors in algorithms	Yes	Yes	Yes	Yes	Yes
Design and use computational abstractions	Yes	Yes	Not yet	No	Not sure
Understand key algorithms that reflect computational thinking (eg sorting and searching)	Yes	Yes	Yes	No	Yes
Programming					
Create simple programs	Yes	Yes	Yes	Yes	Yes
Debug simple programs	Yes	Yes	Yes	Yes	Yes
Use sequence in programs	Yes	Yes	Yes	Yes	Yes
Use selection in programs	No	Yes	No	Yes	Yes
Use repetition in programs	Yes	Yes	Yes	Yes	Yes
Write programs that control physical systems	Yes	Yes	No	No	Yes
Write programs that simulate physical systems	Yes	No	No	No	No
Use two or more programming languages (one of them textual)	No	Yes	No	No	Yes
Make appropriate use of data structures such as lists or arrays	No	Yes	No	No	No
Use procedures and functions in programs	No	Yes	No	No	Yes
Specific (Algorithms)					
Decomposition (chosen by Anna)	Yes				
Algorithm Design and Planning (chosen by Anna)	Yes				

Table 8: What teachers do and do not teach in England.

Name	**Anna**	**David**	**Beatrice**	**Fiona**	**Eliza**	Total
ALG algorithm	Y	Y	Y	Y	Y	5
ENG debugging	Y	Y	Y	Y	Y	5
ALG instruction	Y		Y	Y	Y	4
PRO program	Y		Y	Y	Y	4
ALG decomposition		Y	Y		Y	3
ALG output			Y	Y	Y	3
ALG sequence			Y	Y	Y	3
ENG correctness	Y	Y		Y		3
ENG design	Y	Y			Y	3
ALG logic			Y		Y	2
ALG precision				Y	Y	2
ALG problem solving		Y		Y		2
ALG repetition	Y		Y			2
ALG steps			Y		Y	2
ENG testing		Y		Y		2
PRO logic error			Y		Y	2
PRO syntax error			Y		Y	2

Table 9: The occurrence of different themes in teachers' data (England).

> *"I do wonder whether debugging may change into creating the bugs themselves,* [. . .] *we have to be careful with what we are equipping them with* [. . .] *we have had children in the past by the time they reach the end of Key Stage 2 have been able to bypass certain security systems that we have in school"* (Fiona).

She believes children need to learn CS to be able to understand technology rather than only use it. She does not mention content knowledge beyond what she teaches the students.

Themes

It can be seen in Table 9 that all five teachers discuss the terms algorithm and instruction and talk about debugging to some degree. Next, we look at some particular themes emerging from the interviews in terms of concepts teachers discuss under these headings:

- Algorithms
- Programming
- Debugging and the notion of correctness

Learning about Algorithms

All teachers discussed algorithms at some level. For example, with the youngest children, Anna described that *"the word algorithm has something to do with steps and instructions and getting something right."* For Anna, just the familiarity with the word is what is needed at this early stage. . . *"so we talk about an algorithm for making a Lego structure or making a drink. It's a word that I'm trying to get them to be aware of in their general vocabulary."* Anna refers to the Bloom's taxonomy and the importance of knowledge and terminology as a lower-level skill for young children.

With children who are slightly older, the teachers describe that they use both the words algorithm and instructions and ask children to show their understanding by being able to identify the outcome of the algorithm:

> *"so they need to be able to understand what the algorithm is for, so what the instructions are for, what the outcome is"* (Beatrice).

David is the only teacher who linked the teaching of algorithms directly to computational thinking:

> *". . . how it helps with their computational thinking and how it will help them see things more logically and be able to develop their own algorithmic thinking better and all of those sorts of things"* (David).

At the secondary level, there is more expectation that children should be familiar with algorithms for particular tasks such as sorting and searching. Eliza describes a range of algorithms that she introduces her students to, and is aware that there are many different algorithms and that students should try to understand them.

> *"So personally I would go quite slowly and do a lot of different algorithms so I would do the Intelligent Piece of Paper with the noughts and crosses and then sorts of things like that. . . "* (Eliza).

Eliza then discusses established algorithms that she would or would not introduce students to before the age of 14 (Grade 8):

> *"There are lots of nice searches and sorts in programming that are quite structured and you can build from one to the next one. . . I might do a binary search with a key stage 3 group if it was a quick group, but I might stick with the linear search. Sorting – will do a bit sorting with them but nothing complicated like quicksort algorithms or anything like that. Probably I wouldn't even go as far as a bubble sort just do a simple insertion sort or something like that – with a pack of cards"* (Eliza).

Debugging and the Concept of Correctness

Debugging was mentioned frequently throughout the interviews. It is clearly a skill that teachers feel is important for children to master.

> *"They need to know what debugging means, so debugging is obviously correcting the algorithm and making the program correct so that it works correctly"* (Beatrice).

Teachers are also able to see the cross-curricular benefits of being able to debug for other subject areas, and as a wider skill, as shown in this comment by David:

> *"And for me on a child development level I suppose that's a much bigger win, that's why debugging is a really useful thing for them to have"* (David).

The teacher teaching the youngest children, Anna, was very enthusiastic about getting children to design and plan before working and not tinker, and linked this to the difficulties that teachers have with debugging:

> *"When I'm talking to teachers and they say it's a nightmare when it comes to debugging I ask them do you have an algorithm for them to go back to? And they go* 'a what'*"* (Anna).

Another of the primary teachers wants her students to be able to understand an algorithm sufficiently to be able to predict if it would or would not run — some notion of correctness — and thus be able to debug before running:

> *"... before they run that code, be able to identify whether there may be errors in it straightaway"* (Fiona).

The idea of developing skills in predicting errors is commented on by other teachers:

> *"I want children to be able to look at work, whether it's their own or somebody else's, be able to see what the output of it is and when there are errors be able to work back from the output to be able to find the errors and hopefully correct them. So there may be some logical errors in there, there may be some syntactical errors and be able to tweak those so that they can then get a program function"* (David).

This is actually quite a difficult skill for the primary school students to master, depending of course on the complextiy of the algorithm; overall the emphasis on debugging and correctness can support children in developing some resilience and ability to keep trying when something is not successful:

> *"And I see the debugging element of CS as a really good way of developing their resilience and their determination skills"* (David).

Programming

All teachers talked about programming at some level. For the youngest age group, Jane talked about programming robots such as the BeeBot. Other primary teachers mentioned children using Scratch from age 7–11, with secondary school using a combination of Scratch and a text-based language. Teachers are clear that programming is important, as Beatrice comments:

> *"I just think it's important in this day and age when everything is programmed that they understand what it means and what goes on behind it in order for them to understand the world around them"* (Beatrice).

In terms of programming concepts, only one of the teachers mentioned variables or assignment. One teacher chose to talk about teaching sequence as a concept. Three teachers said that they taught selection but only one mentioned it in their interview. In contrast all teachers of all age groups taught repetition, to some degree.

Teachers had differing opinions on the extent to which children should tinker and explore when learning programming or whether they should always plan and design. For example, Anna feels strongly that even young children should design an algorithm before any hands-on work with BeeBot or whatever tool they are using:

> *"That's what I want them to learn: don't just go for the hacking. Sit back and have a little think about what you can work out that you want it to do and how it might work... if they're in literacy they write a plan. In DT they do their plan. In science they do a plan. Perhaps in art they don't, but even then we make them think about it. Why do we let them do it in CS?"* (Anna).

In contrast, Beatrice feels that exploring is a better way for children to learn concepts such as repetition:

> *"We use repeat, we use forever — I give them the opportunity to experiment with the different ones. I think that they should ... be given an outcome and then they can figure out what all the different coding and all the different colours mean ... only by just experimenting — otherwise — if they don't learn it themselves... that 'forever means that that can always keep happening' — if they don't experiment with that — I don't think I could just teach them like 'Look you use Forever, this is what it means' — they need to experience it"* (Beatrice).

However, Beatrice has been identified as having less content knowledge. Some teachers have developing content knowledge as new primary teachers of CS; the result is that it is difficult for them to clearly express what children learn with respect to programming with appropriate technical language.

> *"... and they need to understand the coding, and how the coding works, in all the different colours, to be able to create the program, and understand how it works"* (Beatrice).

Teachers working at primary school discuss visual environments as a suitable tool for teaching programming, with David reporting that he does not feel that text-based programming is appropriate at primary school. At secondary school, it is felt that text-based programming is more appropriate, as Eliza reports:

> *"So really at KS3 I want them to be exposed to the syntax of a language, to be exposed to the structures of a language"* (Eliza).

However she explains that visual environments have their uses at the beginning of secondary school as long as they are taught with constructs that map easily on to a text-based language.

Assessment

Teachers were asked, for each topic that they chose: *"What are the specific ways of ascertaining students' understanding or confusion around this idea."* A range of strategies were

Name	**Anna**	**David**	**Beatrice**	**Fiona**	**Eliza**	Total
Observation	Y	Y	Y	Y	Y	5
Open-ended task	Y	Y	Y	Y	Y	5
Questioning	Y	Y	Y	Y	Y	5
Giving buggy code	Y		Y	Y	Y	4
Screenshots of code	Y			Y	Y	3

Table 10: Teachers' assessment strategies (England).

suggested — we have summarised them as shown in Table 10.

As can be seen in the table all teachers talk about observing the children to find out how much they have learned about algorithms and programming:

> *"The ones that make more progress are the ones who sit back and map it out. A lot of it is just watching and giving them activities"* (Anna, teaching age 4–7).
>
> At all ages, observation is a key formative assessment tool: *"Just by going and looking at what they are creating"* (Beatrice, teaching age 7–11).

Another key mechanism being used to assess childrens' progress is through talking to them, questioning them and asking them to talk about their own progress:

> *"Talking to them... and saying what can you see that's working, which bits do you think are working and which bits do you think aren't working, why that might be and why do you think and using some of those probing or open ended questions can be quite a useful way of judging their understanding"* (David).
>
> *"Asking them how they would code a program. How would they put the algorithm and coding together to make something work. From asking them those questions I would definitely be able to identify whether they knew what they were talking about, they would know the colours of instructions that they would need to use. I think I could easily assess them by talking to them about what they have created and how they did it"* (Beatrice).

Another strategy used is self and peer assessment:

> *"I want children to be able to look at work, whether it's their own or somebody else's, be able to see what the output of it is and when there are errors be able to work back from the output to be able to find the errors and hopefully correct them. So there may be some logical errors in there, there may be some syntactical errors and be able to tweak those so that they can then get a program function"* (Eliza).

Giving them opened-ended tasks and then asking them questions are key aspects of formative assessment:

> *"Why are you doing that? Do you have one of these?"* (Anna).

Teachers used open-ended tasks or programming tasks to help children practice the skills they needed and implement algorithms as programs:

> *"I have a lot of task-based sheets – we teach a topic and then we give them a task to do on that topic, so there's an awful lot of... here's a little algorithm go away and write it... here's a little algorithm, have a go at that one"* (Eliza).

In terms of summative assessment, teachers talking about photographic evidence of what the children have achieved at primary school, and having some multiple-choice tests or homework at secondary school, but across all interviews there was a greater emphasis on formative assessment or assessment in order to support students' progress.

> *"We did take photos of things that they'd fixed or things they had broken and bits and pieces like that and again sort of throw that up on the blogs"* (David).
>
> *"I do the multiple choice exercises and I do tests and I set the homeworks where they write in Python and when they do bug-checks – so I'm trying to cover all different angles to get a grip on which bits they didn't understand of each task. But they are all different – each individual will misunderstand something in their own individual way"* (Eliza).

Fixing errors is an other activity that can lead to assessment as it enables teachers to see how good children are at trouble-shooting: four out of the five teachers mentioned this:

> *"I think giving them bad stuff has got a really valuable place to play. In terms of their assessment you are then looking at whether they are able to rectify the problems and I suppose that blurs into the debugging skills..."* (David).

In summary, teachers focus on trying to ascertain childrens' understanding to a large extent without formal testing:

> *"It's when a child gets that lightbulb moment — that's when the real power comes — being able to capture those, in assessment terms"* (David).

Teachers are obviously able to employ a range of assessment strategies to this end. Processes for summative assessment at primary level are less evident in the teachers interviewed.

7.2 Italy

The Italian teachers who accepted to take part in the interview received an outline with the general areas and the intended questions a few days before the appointment. Together with the interview scheme, they were also given a few notes about the following curricular material:

- the (thin) national recommendations on *digital competence* and *technology* areas, see Section 2;

- a tentative document with more comprehensive potential guidelines for (Italian) compulsory education;
- the ACM/CSTA models of CS curricula [81];
- the recent national CS curricula of UK [36].

(Only one teacher hadn't had a chance to look at them until the interview).

Two primary school teachers were interviewed together; all the other individually, either directly or via Skype. At the beginning of the conversations, the teachers described their experience. Then each interview proceeded according to the proposed scheme. The teachers could however interpret the questions freely, to some extent, in the light of their actual views. The interviews lasted from 1 hour and 15 minutes to about 2 hours and a half.

Teachers' Characteristics

The teachers came from a very wide area of Northern Italy. In conformity with the national education systems, teachers are usually able to teach at only one of three levels: primary (K–5), lower secondary (6–8), or high school (9–13). Some of them, however, have been able to teach at different levels within special institutional or inter-institutional cooperation projects.

Most of the teachers do not have a strong background in CS, but have learnt what they know about this subject for instructional purposes. The information in the header of Table 11 roughly characterizes the sample of teachers interviewed in Italy.

Alessandro and **Roberto** are experienced primary school teachers. They introduced Logo in the mid-80s and have since then been interested in introducing programming in elementary education. **Sonia** is a young primary school teacher. She completed her degree five years ago with a teacher internship program on CS topics given to 5th graders, that included programming in Scratch and Logo. Then she continued to cooperate with her colleagues on this subject.

Francesco, **Lorenzo** and **Martina** are middle school teachers of Mathematics and Science. Francesco regularly teaches a variety of CS-related topics, including programming in Scratch, BeeBot and Lego robots both to his students (grades 6-8) and, in team teaching, to elementary school children (K-5). Lorenzo teaches some CS: an introduction to miscellaneous CS topics and an extracurricular (elective) Scratch lab. Martina is an enthusiastic teacher, who has been responsible for the information/web services used in her school for about 15 years. Besides being involved in basic digital literacy programs (word processing, spreadsheet, presentation programs, use of browsers, construction of web pages), at present she works in cooperation with a team of scholars in computer science education.

Maurizio is an experienced high-school teacher in CS, with a very strong background in the subject. He has also been teaching CS topics in primary and lower secondary schools for the last 6 years (special projects), namely using Scratch (mostly grade 5, but also lower grades), Lego robots (grade 8) and GIS (grade 7). Finally, **Giuseppe** worked for several years as a teacher of Mathematics and Physics in the high school, where he introduced CS subjects since the 80s, within the National CS Project (PNI) initiative that gave him the opportunity to develop a broad knowledge of the foundations of CS. He then turned to teacher training and to the study of pedagogical issues arising in mathematics and CS at all levels of instruction, which led him to cooperate in educational projects with elementary and middle school teachers.

In the Italian context where the national recommendations are quite vague, the main concern of teachers appears to be the potential of CS topics and abilities to attain general, trans-disciplinary educational objectives.

As the teachers remark:

> *"Primary school has usually taken a pre-disciplinary approach, in contrast to the lower secondary level* [...]*. And above all, in my view, primary school should limit as far as possible any 'formalization' of the disciplines"* (Alessandro).
>
> *"My aim is not to train prospective computer scientists or experts about robots. I don't care at all about this. I care that [pupils] see what there is in the world and are able to choose. And that they learn some method. Teamwork is fundamental for me, this is essential. And the fact that they have a logic in the work they do, that they can explain what they do"* (Francesco).
>
> *"The important* [aspects] *are fondness, enthusiasm* [...]*, discovery, curiosity* [...]*. And* [...]*, as usual but important, cooperation, teamwork. And I might add respect for the work of others"* (Maurizio).

The sample of teachers interviewed in Italy have quite diverse backgrounds, namely: Science of Education, as usual, to teach in the elementary school; Agricultural Science, Biology, Computer Science, Mathematics and Physics in the case of middle and high school. The paths that led them to choosing to teach a bit of CS and programming are varied and often interesting. Alessandro, for instance, looks back to his first experiences:

> *"The educational, pedagogical approach* [of Papert's 'constructionism'] *has really changed my perspective. In the mid-80s I had the opportunity to work in one of the first schools* [...] *that created a Logo lab, with the Commodore 64.* [...] *Thus, I had one of the very first experiences in my area and, I think, in Italy too. I came back in my school* [...] *and I suggested to develop a lab also there.* [...] *I learned Logo by myself, and then I started to explore with the children* [the potential of Logo]*"* (Alessandro).

Alessandro explains the role of CS in his pedagogical view:

> *"In primary school the approach surely cannot be rigid.* [...] *It should be an 'immersive' approach, in some respect, i.e.: I build a challenging environment, I bring you within this environment, I encourage you to formulate projects and I help you — I'm a mentor, I give you advice.* [...] *Scratch is the real descendant of Logo, in terms of educational philosophy.* [...] *What it has actually added is the '2.0' social environment. That was the big step forward"* (Alessandro).

Sonia, on the other hand, follows a more standard, structured approach, but cares about the children's attitude toward the devices:

"I've noticed that the children at the computer tend to be passive. They see the computer as something 'intelligent', something far out of their grasp, and they don't know that it is some people who wrote the programs, who made it work" (Sonia).

To mention also a couple of meaningful excerpts concerning the middle school, Francesco seems to take an 'engineering' perspective:

"I want them to realize what it means to design, the difference between production and design. [...] *And then the technical report in which they analyze the starting point, the problem, the solution, the project"* (Francesco).

And Martina cares about the implications of the evolution of ITs for her subject:

"Looking at what's going on, for example, around 3D printing [...] *I came across a problem: well, I must change the approach to teaching solid geometry, since by doing it in the standard way, the boys won't be able to deal with something so nice, so creative, potentially,* [...] *something that will be in their houses"* (Martina).

Themes

Tables 11 and 12 report the "big ideas" — or sometimes "relevant abilities" — explicitly suggested by the teachers (shading) or simply emerging from the analysis of the transcripts (white background). It should be noticed that the listed "ideas" are those deemed to be most important by the interviewed teachers, but are by no means exhaustive of their learning objectives (consequently, the option "No" does not appear in the table cells). Of course, the teachers were unaware of the choices of their colleagues.

As an overall picture, we can see that not all the items indicated by the Italian teachers can be categorized precisely as "ideas". On one hand, they do not appear to care much about specific disciplinary concepts. Rather, they are interested in the development of mental structures, general competences and abilities with trans-disciplinary potential that cannot be easily formalized. On the other hand, they tend to propose operational tasks, that may result in concrete experiences for their pupils and tangible products.

Moreover, as far as the "security" area is concerned, the teachers are mostly interested in social and individual safety issues, rather than in its technical implications that fall in the corresponding category of the ACM/IEEE curricular models [81] — see Table 12.

The Italian data were also coded using the concepts identified in the curriculum comparison analysis as well as a few additional categories taking into account the teachers' beliefs and pedagogical practices. The occurrence of different themes is summarized in table 13.

Learning about Algorithms

All teachers discussed algorithms in some respect and, in particular, considered this broad category from the viewpoints of procedural thinking, problem solving and design. A selection of the teachers' comments follows.

In the earliest stages of instruction the focus is on children's active, bodily experience. In Giuseppe's words:

"In the elementary school children, before conceiving any algorithm, should be educated to follow procedures, to concretely 'do' such things. [...] *For the children of first and even second grade there are several prerequisites. Otherwise they won't be able to think of a real algorithm. They'll struggle"* (Guiseppe).

Indeed, according to Roberto, usually kids' approach to (their) procedures is not mindful:

"In terms of children's experience, I see that they don't use an algorithm because they proceed by trial and error. [...] *They go there, they begin to tinker, they do... but they aren't aware of what they have done.* [...] *So they don't have an algorithm"* (Roberto).

Teachers appear to be especially concerned with the connections of algorithms with problems and problem solving practice.

"A first important idea, a basic one, which is in my opinion at the root of algorithms, is to identify the problem. [...] *To distinguish the problem from its variables items.* [...] *That is, to recognize the problem in itself, the core of the problem, I don't know, what is common to problems that may look different"* (Martina).

"I think it's fundamental to figure out what's the problem to be addressed" (Francesco).

"I would say that the primary school should endeavor to find the situations, the tasks, the environments in which, for example, children are led to distinguish between those problems that can be solved by an algorithm and those that cannot. [...] *Because often the kids get confused in this respect"* (Alessandro).

The relationships between algorithms and problems include the idea that one can explore different algorithms to solve the same problem and then it is important to ask, as suggested for instance by Giuseppe,

"if I can get the same result by a different procedure, in a different way, by following a different path" (Guiseppe).

Another aspect on which the teachers insist is the need of precision, accuracy:

"In an algorithm there is a sequence of operations to be carried out in a strict, inescapable order, in the sense that if you change the sequence, you get a different result. That is, you cannot do things in a haphazard way, you have to choose the right sequence of operations to get a correct output" (Martina).

Interestingly, algorithms are also viewed as a means to devise models,

"intended as a simplified representation of a complex system. [...] *The most important thing is that [children] understand that in order to deal with a complex situation they have to build a model"* (Maurizio).

Name	**Alessandro**	**Roberto**	**Sonia**	**Francesco**	**Lorenzo**	**Martina**	**Maurizio**	**Giuseppe**
Age group taught	6–11	6–11	6–11	6–14	11–14	11–14	8–19	(6–19)
CS knowledge (Strong/Middle/Weak)	M	M	M	M	M	M	S	S
Hours teaching CS/week (average)	1	< 1	(3)	4–5	< 1	1	18	n.a.
Years teaching (approximately)	30	30	5	10	10	15	20	30
Years teaching CS (approximately)	30	30	3	5	5	10	20	30
Algorithms: procedures								
Practicing procedural tasks (K-3)		Yes						Yes
Introspection and verbalization of procedures (K-8)		Yes			Yes			
Sequencing the operations in the right order (K-8)		Yes		Yes		Yes	Yes	Yes
Understanding the logic of algorithms (K-8)				Yes			Yes	
Understanding conditional and iteration (K6-8)						Yes		
Thinking in terms of whole strategy (K-5)		Yes						
Algorithms: problem solving								
Decomposing problems into smaller parts (K-8)		Yes	Yes	Yes		Yes		
Problems that can/cannot be solved by algorithms (K-5)	Yes							
Generalizing to several problem instances (K4-8)		Yes	Yes		Yes	Yes		
Different procedures can lead to the same result (K4-5)		Yes	Yes					Yes
Logical relationships between processed data (K-8)								Yes
Algorithms: design								
Providing a clear 'workflow' to follow (K-5)							Yes	
Modeling simple behavior (K6-8)				Yes			Yes	
Describing algorithms in different languages (K6-8)								Yes
Compactness and effectiveness of a solution (K-5)	Yes	Yes	Yes					Yes
Programming: language								
Knowing the typical basic operations (K-8)				Yes				
Building blocks and syntax of a formal language (K-5)						Yes	Yes	
Universality of the main constructs (K-5)	Yes							
Using different languages (K-8)			Yes	Yes		Yes		
Programming: coding								
Formalizing accurately and precisely (K-8)	Yes	Yes	Yes			Yes	Yes	
Modifying code for self-expression (K-5)	Yes						Yes	
Programming to implement models of behavior (K6-8)							Yes	
Coping with debugging tasks (K-8)	Yes			Yes		Yes		
Programming: design patterns								
Understanding the role of control structures (K-8)	Yes			Yes	Yes	Yes	Yes	
Procedure parameters (K-5)	Yes	Yes	Yes					
Using a counter (K-5)	Yes							
Absolute assignment vs. operator assignment (K6-8)					Yes			
Data: representation								
Knowing that there are different types of data (K6-8)					Yes		Yes	
Representation of spatial information (K6-8)					Yes		Yes	
Data coding in files (K6-8)					Yes			
Data: interpretation								
Data vs. information (K6-8)								Yes
Knowing reliable sites (K-5)								Yes
Evaluating data sources (K-8)		Yes		Yes		Yes	Yes	Yes
Data: collection and organization								
Basic use of a search engine (K6-8)				Yes	Yes	Yes		
Logical organization of data (K6-8)				Yes				Yes
Organization of files and folders (K6-8)					Yes			
Architecture of digital documents (K6-8)						Yes		

Table 11: What Italian teachers focus on; Giuseppe is a teacher trainer. (To be continued in tab. 12.)

Name	**Alessandro**	**Roberto**	**Sonia**	**Francesco**	**Lorenzo**	**Martina**	**Maurizio**	**Giuseppe**
Age group taught	6–11	6–11	6–11	6–14	11–14	11–14	8–19	(6–19)
Security: technical issues								
Saving a backup copy of work done (K-8)								Yes
Issues related to password and authentication (K6-8)					Yes	Yes		
Treatment of network data (K6-8)					Yes			
Security: personal safety ad social issues								
How to behave in digital environments (K-8)		Yes	Yes			Yes		
Protecting personal identity and data (K6-8)				Yes		Yes		
Positive vs. negative behavior in social networks (K6-8)						Yes		
Being aware of cyber-bullying (K-8)	Yes			Yes				
Copyright and licensing implications (K6-8)				Yes		Yes		
Other topics								
Basic operations of an operating system (K6-8)					Yes			
Cooperation in virtual communities (K-8)	Yes					Yes		
Caring about enthusiasm-cooperation-respect (K-8)							Yes	

Table 12: What Italian teachers focus on. (Continued from tab. 11)

Programming

A variety of reasons are mentioned to account for the educational value of programming. According to Alessandro, programming is important, first of all, because *"it is a form of self-expression."* For Roberto, *"the child is usually moody, emotional, [...] impulsive,"* so programming *"should help him be rational."*

> *"Learning a programming language, from a certain point of view, allows you to better understand the world. You have an additional tool for understanding the world"* (Maurizio).

Only one teacher seems to be a little skeptical about the role of standard formal programming in early education:

> *"When it comes to connections between language structures and some reality — because problem-solving is essentially this — with children and pupils [...] we have to start from the language structures that are more familiar to them. [...] We made a mistake, when we started using Logo in the elementary school, because [...] the language syntax was too strict, with spacing, with brackets, and so on. [...] It wasn't a simple language for them"* (Giuseppe).

Often, in primary school, programming is introduced as 'storytelling':

> *"There is much programming work before. So: What story do I invent? What I want to do? What is the main character? Where?... The plot, in short. And that's programming for me"* (Roberto).

Next, we present a few excerpts addressing a selection of the various aspects discussed by teachers.

To begin with, the features of precision and accuracy of a programming language have been taken into account by most of the teachers. For instance:

> *"Precision of language. And correctness of the language, too: If you swap those two things you don't get the same result"* (Roberto).

Quite unexpectedly, we can find frequent remarks about the benefits of being exposed to more than one programming language.

> *"About programming my concern is that they understand that there are languages with which you can give instructions to a machine. And you can do it using Logo, using Scratch, or using any language"* (Sonia).
>
> *"About programming my concern is that they learn to use different languages, that they understand what are the very key elements of programming"* (Francesco).
>
> *"Important idea: to understand terms and syntax of a formal language, a visual one and a textual one — to see both examples. [...] To have an idea of a textual one is important as well, in my opinion, in order to develop abstraction, and then to get to the point. [...] And also knowing different numbering systems has the same logic"* (Martina).

Variables are considered to be a hard topic, both for primary and middle school. As stated by Lorenzo,

> *"In Scratch, for example, you have two instructions: to assign a value and to increment (decrement) it."*

However, pupils find it difficult *"to understand in what circumstances to assign a value"* and in what circumstances to change it. So, according to the teachers, the main use of variables is in order to represent procedure parameters.

Also the educational potential of debugging is worth considering, particularly in the middle school:

Name	**Alessandro**	**Roberto**	**Sonia**	**Francesco**	**Lorenzo**	**Martina**	**Maurizio**	**Giuseppe**	
Age group taught	6–11	6–11	6–11	6–14	11–14	11–14	8–19	(6–19)	Total
ALG: algorithm	Y	Y	Y	Y	Y	Y	Y	Y	8
PRO: program	Y	Y	Y	Y	Y	Y	Y		7
ALG: sequence/order		Y		Y		Y	Y	Y	5
ALG: abstraction/generalization		Y	Y		Y	Y		Y	5
ALG: precision/formalization	Y	Y	Y			Y	Y		5
ENG: design	Y	Y		Y			Y	Y	5
PRO: control structure	Y			Y	Y	Y	Y		5
DAT: data				Y	Y	Y	Y	Y	5
OTH: evaluation of data source		Y		Y		Y	Y	Y	5
ALG: problem decomposition		Y	Y	Y		Y			4
ENG: evaluation/efficacy	Y	Y	Y					Y	4
PRO: variable	Y	Y	Y		Y				4
PRO: language	Y		Y	Y		Y			4
ENG: pair programming/team work		Y	Y	Y			Y		4
DAT: data organization				Y	Y	Y		Y	4
ALG: procedural task		Y			Y			Y	3
ALG: algorithm logic				Y			Y	Y	3
ENG: testing/debugging	Y			Y		Y			3
NET: search engine/data collection				Y	Y	Y			3
MOD: modeling/simulation				Y			Y		2
ALG: input/output				Y		Y			2
ENG: exploration/heuristics				Y			Y		2
PRO: implementation/coding	Y						Y		2
PRO: syntax/program structure						Y	Y		2
DAT: data representation					Y		Y		2
DAT: type					Y		Y		2
DAT: table				Y				Y	2
ALG: selection/repetition						Y			1
ALG: problem solving/feasibility	Y								1
ALG: algorithm representation								Y	1
ENG: work plan							Y		1
ALG: instruction set				Y					1
PRO: constant					Y				1
DAT: data vs. information								Y	1

Table 13: The occurrence of different themes as teachers' focus (Italy).

"An important aspect is 'debugging'. Once you have written the program, you have run it... Well, you don't have to take the fact that it doesn't work as a defeat. It is a starting point anyway, to be able to solve the problem, that is, to see what was wrong. Debugging is important from the educational, pedagogical viewpoint... There is a procedure, you have tried to formalize it, may be the procedure is wrong or maybe the formalization is wrong, isn't it? [...] *Just as an approach to problems, to difficulties. The real problems are like that"* (Martina).

Italian teachers, like their English colleagues, do not fully agree on the implications of explorative tinkering vs. more rigorous planning when learning to program. In Roberto's perspective, for instance, a major educational objective is precisely to overcome the trial-and-error approach, whereas Maurizio reports:

"I also allow the boy [...] *to proceed by trial-and-error, to tinker, possibly without much reasoning. Somehow heuristically"* (Maurizio).

Finally, although all teachers agree on the need of fostering cooperation among pupils, they show contrasting opinions as to the actual implementation of pedagogical strategies such as pair-programming or the like.

"I read the results of some research according to which the children perform better if they work in group in front of a single iPad. I totally disagree about this. [...] *Collaborative learning must be a choice as well as an achievement. That is, if I have to work with others, and I must be in a group with others,* [...] *I may do everything myself or* [...] *leave it all to the others. If I have already made my own experiences, I have something to say, then I can choose to join others and share my knowledge, my skills with them"* (Alessandro).

"Usually the children do these activities in pairs, or in larger groups of 4–5 people. A child is never working alone to develop a program, to figure out an algorithm. Usually we apply cooperative learning" (Sonia).

Assessment

The teachers mentioned a range of possible assessment strategies, which are summarized in table 14. Their effort to view the implications of CS education (at the considered level of instruction) from a more abstract educational perspective seems interesting to consider.

Relative to primary school, the observations span several elements, including some aspects of the pupil's behavior:

"Assessment is always global, not just of the final product. You take into account all aspects, at least in elementary school" (Sonia).

Or:

"The assessment is not about what they produce, but about what is their way of behaving in different situations" (Roberto).

"If I have to assess a child's learning about programming, I have to know whether and how often he makes debugging. [...] *For example, if a program doesn't work, does he drop it and restart? Or does he apply some techniques, some strategies in order to try to find the errors?* [...] *Does he usually borrow pieces of code from other projects in order to develop his own projects? This is a usual* [helpful] *practice today"* (Alessandro).

In middle school, on the other hand, the assessment tends to be more formal and to address technical features, even for an open-ended task:

"I assess the pupils on the final project, for example. That is by using a structured evaluation grid. For the final project I use an evaluation grid, that I also give them for self-assessment" (Francesco).

There are also attempts to evaluate students' ability to decompose a problem into smaller parts:

"You assign a problem that has been discussed, that has already been broken down, and you can see if the student is able to do the same thing again. And then you assign a problem that may look completely different to him, but such that the underlying algorithm is actually similar... But I would assess precisely his ability to break it down" (Martina).

Maybe by proceeding in the opposite direction, and asking students:

"Try to envisage a problem whose solution can be represented by this structure" (Martina).

Finally, an example of summative assessment of debugging skills:

"Debugging can be assessed. Just give him a simple little program, maybe of 4, 6, 20 lines of code, and see how they behave. Maybe one of the tasks is straightforward to do on paper; then something that requires a couple of additional steps; and in the third case, instead, the task is more complex because, maybe, you have to stop the program at different points, to split it up... " (Martina).

8. RESULTS: BEBRAS TASKS ANALYSIS

CT Concepts Classification

Table 15 shows the distribution of CT concepts in the 2010, 2011, 2012, 2013, and 2014 Bebras tasks. Note that any individual task could be coded using more than one CT term so that the percentages total to more than 100.

For the purposes of this report we are interested in the tasks assigned to lower grades, so in Table 16 we show the CT classification for tasks in levels 0 through II (aimed at 8 - 14 year old pupils) .

The relative importance of the various CT concepts in each year can be seen in the charts in Figure 2.

Since there were many terms that occurred together, for example algorithms and data representation, it was useful to

Name	**Alessandro**	**Roberto**	**Sonia**	**Francesco**	**Lorenzo**	**Martina**	**Maurizio**	**Giuseppe**	
Age group taught	6–11	6–11	6–11	6–14	11–14	11–14	8–19	(6–19)	Total
Observation (process, result, product)	Y	Y	Y		Y	Y	Y		6
Questioning (teacher's/peers' questions)	Y	Y			Y		Y		4
Questionnaire/quiz/test	Y					Y	Y		3
Structured task			Y	Y		Y			3
Open-ended task (individual/group work)				Y			Y		2
Contest				Y			Y		2
Report (project/lab)				Y					1
Problem-solving test (transdisciplinary)								Y	1
Code		Y							1
Giving buggy code						Y			1

Table 14: Teachers' assessment strategies (Italy).

CT term	**2014**	**%**	**2013**	**%**	**2012**	**%**	**2011**	**%**	**2010**	**%**	**2010–14**
abstraction	6	5%	27	18%	37	30%	26	21%	11	8%	16%
algorithms	99	77%	118	79%	87	70%	75	60%	59	42%	66%
data analysis	6	5%	4	3%	3	2%	7	6%	8	6%	4%
data collection	1	< 1%	0	0%	0	0%	0	0%	2	1%	0%
data representation	70	54%	46	31%	33	27%	33	26%	71	51%	38%
parallelization	2	2%	2	1%	2	2%	2	2%	0	0%	1%
problem decomposition	7	5%	3	2%	0	0%	3	2%	8	6%	3%
simulation	7	5%	18	12%	3	2%	15	12%	6	4%	7%
literacy	1	<1%	3	2%	6	5%	17	14%	15	11%	6%
Total tasks	129		150		124		126		139		

Table 15: Distribution of CT concepts in the 2010–2014 Bebras tasks.

CT term	**2014**	**%**	**2013**	**%**	**2012**	**%**	**2011**	**%**	**2010**	**%**	**2010–14**
abstraction	3	3%	20	17%	20	24%	18	18%	6	9%	14%
algorithms	77	76%	96	80%	59	70%	57	58%	29	44%	68%
data analysis	5	5%	4	3%	3	4%	5	5%	2	3%	4%
data collection	1	<1%	0	0%	0	0%	0	0%	1	2%	0%
data representation	56	55%	38	32%	19	23%	26	26%	35	54%	37%
parallelization	2	2%	2	2%	0	0%	1	1%	0	0%	1%
problem decomposition	4	4%	1	<1%	0	0%	2	2%	4	6%	2%
simulation	5	5%	17	14%	2	2%	14	14%	2	3%	9%
literacy	0	0%	2	2%	6	7%	14	14%	6	9%	6%
Total tasks	101		120		84		99		65		

Table 16: Distribution of CT concepts in the 2010–2014 Bebras tasks, Levels 0–II only (8-14 year olds).

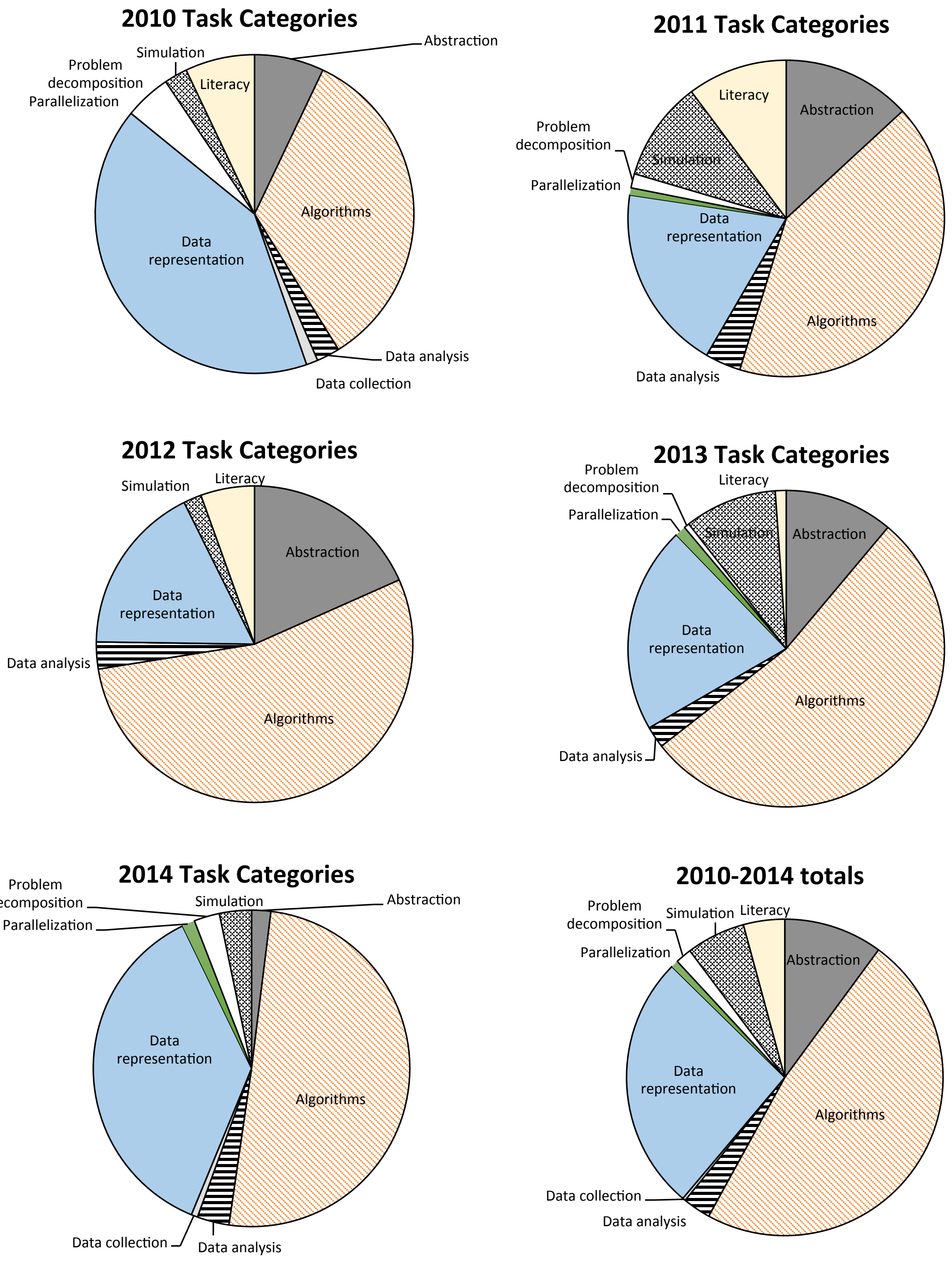

Figure 2: Distributions of types of CT for the earlier (0–II) school/age levels by year.

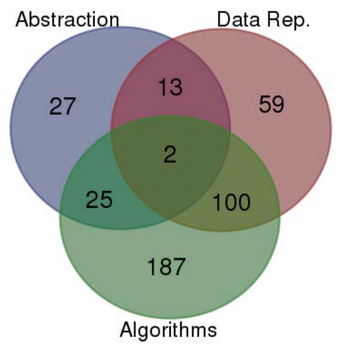

Figure 3: Overlap of CT term for the earlier (0–II) school/age levels for all four years.

consider groups of CT categories. To capture that information all possible tuples of categorizations were considered. The following lists show all of the possible tuples organized by complexity.

Pairs:

- algorithms, abstraction
- algorithms, data analysis
- algorithms, data representation
- algorithms, parallelization
- algorithms, problem decomposition
- algorithms, simulation
- abstraction, data representation
- data analysis, data representation
- data analysis, problem decomposition
- data collection, data representation
- data representation, problem decomposition
- data representation, simulation

Triples:

- algorithms, data analysis, problem decomposition
- algorithms, data representation, abstraction
- algorithms, data representation, simulation
- algorithms, problem decomposition, simulation

Table 17 shows the differences in distributions of groups of CT tasks by year for the earlier levels (0–II). There were many tuples that were infrequent, so any tuple that constituted less than 4% of the data in all years considered was discarded. Put another way, the only rows that are displayed are those that included at least one year that had the indicated tuple in at least 4% of the data. Figure 3 shows the intersection between the three top categories for the K–9 level (0–II) tasks of all four years. We should note that many tasks describe navigational problems, i.e. find a path in a maze, or the shortest way to reach a given point, which most students will find familiar in real life. Hence, this is reflected in the high frequency (100 tasks over 4 years) that have both Algorithms and Data representation.

Question Structure Classification

All of the Bebras tasks in years 2010, 2011, 2012, 2013, and 2014 that had been labeled with the CT concept of algorithms were classified for problem structure. Table 18 shows the task classification for all years, including all levels (0–IV).

The bar graph in figure 4 shows the distribution of the types of algorithms questions found in the 2010, 2011, 2012, 2013, and 2014 Bebras contests. For this graph, only tasks categorized as algorithms tasks have been considered. Tasks belonging to more than one question type have not been included. Similarly, only the younger groups (0–II) were considered since that is the focus of this report.

9. CONCLUSIONS AND DISCUSSION

The exploratory study presented in this report focused on key concepts and their assessment in K–9 CS education. The analytic part of the study focused on the following research questions:

1. Which concepts are present in K–9 curriculum documents?
2. Which concepts are taught in practice? Which assessment practices are used?
3. Which concepts are assessed in Bebras tasks? How can the assessment format of these tasks be characterized?

Question 1

The following patterns emerge from the analysis of the national recommendations and guidelines. From a global perspective, there is a significant difference in broadness, with the CSTA model at one extreme, covering a variety of topics, and the Italian informal guidelines at the other extreme, mainly focusing on few, somewhat "traditional" areas, namely algorithms, programming, and their relationships with maths and data.

The CAS curriculum, on the other hand, is characterized by an emphasis on technical concerns, specifically in the areas of programming, engineering (proceduralization of tasks), architectures and networking.

Algorithms represent a significant concern in all considered documents. However, whereas all documents mention the general ideas about algorithms, there is considerable difference in the number and variety of suggested examples.

Programming, overall, is given a similar relative weight in terms of reported items. The CAS recommendations appear to stress the technical aspects of programming more than others.

Societal issues are most prominent in the CSTA model, whereas security is hardly considered in all documents.

The concept classification procedure turned out to be quite useful to analyze and compare curriculum documents written in a variety of styles. It will be interesting to apply our method to other curriculum documents. For example, we intend to apply the classification method in the future to compare the conceptual content of K–9 curricula to those intended for grades 10–12 (e.g., in countries that only have a CS curriculum for higher grades like the Netherlands and France).

The coding was done by three researchers. Some parts were coded by two researchers independently, who later compared their classifications. These comparisons showed a high

CT tuple	2014	%	2013	%	2012	%	2011	%	2010	%
(abstraction, algorithms)	1	1%	11	9%	7	8%	8	8%	0	0%
(abstraction, data representation)	2	2%	4	3%	0	0%	5	5%	2	3%
(algorithms, data representation)	37	37%	23	19%	12	14%	10	10%	13	20%
(algorithms, simulation)	0	0%	10	8%	1	1%	7	7%	1	2%

Table 17: Most common groupings of CT concepts in the 2010–2014 Bebras tasks, Levels 0–II only.

Question structure	2014		2013		2012		2011		2010	
	0-IV	0-II	0-IV	0-II	0-IV	0-II	0-IV	0-II	0-IV	0-II
constraint	9	6	10	9	5	5	11	7	4	1
formula identification	2	1	3	2	0	0	0	0	4	2
optimization	10	8	17	12	10	9	6	6	7	3
procedures	5	4	3	2	10	5	0	0	5	2
verification	2	2	21	19	19	15	16	13	11	5
sequencing	9	7	4	3	3	3	4	4	0	0
ordering	2	1	1	0	2	1	0	0	0	0

Table 18: Question structure classifications for algorithms tasks in 2010–2014 for all levels and for (0-II) levels only.

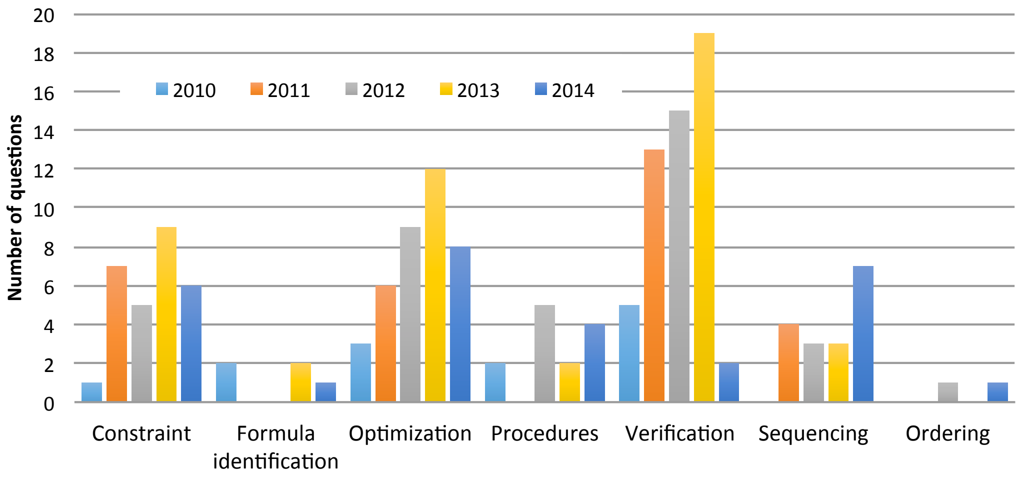

Figure 4: Distribution of the structure of algorithms questions found in the 2010–2014 Bebras contests.

inter-coder agreement. In a follow-up analysis we intend to investigate reliability in a more formal way.

Question 2

Our findings suggest that, at least in England, the intended curriculum is becoming the implemented curriculum to a large extent. Teachers are able to select areas of algorithms and programming that they are teaching at an appropriate level.

Depending on subject knowledge, education and any experience working in the IT industry, teachers' answers vary. We observed that it was necessary for a teacher to have a good content knowledge to be able to see the big picture of the whole of the CS curriculum and to see the direction the teacher was heading with their CS education. Some teachers recognize the importance of the subject in the future lives of their students rather than the importance of learning a particular concept in relation to the rest of the curriculum.

By the same token, some teachers cannot see beyond the level at which they are teaching because of their burgeoning content knowledge; others are able to see the boundary between what can be taught above and below the age of pupils they are teaching.

A variety of assessment strategies are being used (e.g. observations, questioning and test), which are age appropriate. The teachers feel comfortable with these forms of assessment. For many of them it is more difficult to assess the understanding of algorithms than programming abilities. The older the students, the more formal assessment methods and tests get employed.

The fact that the Italian national recommendations are still quite vague might be an explanation for the variety of concepts taught by Italian teachers, besides the differences in the teachers' backgrounds. The main concern of teachers appears to be the potential of CS topics and abilities to attain general, trans-disciplinary educational objectives.

The small sample of teachers can be seen as a limitation of the study. The data elicited by the four CoRe questions is surprisingly rich, however, which provided us an in-depth view about the teachers' beliefs and reasoning underlying their classroom practice (cf. [3]). We intend to pursue this direction of analysis further. Moreover, we think it will be worthwhile extending this part of the research to investigate teachers' practice in other countries.

Question 3

The most common CT concepts in the Bebras tasks are algorithms. Questions classified using the term algorithms represented, e.g., 77% of all tasks and 76% of lower-level tasks in 2014. The next most common CT concept was data representation with 54% of all tasks and 55% of lower-level tasks in 2014. Abstraction and simulation were also popular concepts, although they were not seen as consistently across all years as the other two concepts. There were no consistent or noticeable differences between the CT classifications for all tasks versus lower-level tasks.

There was an interesting decline in the popularity of literacy-related tasks through the years analysed. Literacy tasks were common in the earlier years (e.g., 10.8% in 2010) and became much less frequent in later years (e.g., <1% in 2014). We hypothesize that this is related to the stronger interest in CT skills in schools during the time period considered.

When combinations of CT terms are considered, the most common tuples involved abstraction, algorithms, and data representation. The most commonly paired terms were algorithms and data representation, in part because the representation of the data for a problem can have an impact on the algorithmic approach used in a problem. Abstraction and algorithms, abstraction and data representation, and algorithms and simulation were other popular pairings, although the importance of each tuple varied a bit through the years. Part of the reason why simulation was somewhat inconsistently represented was related to the use of interactive exercises in the Bebras question sets. In some years these interactive questions were more common than in others, and interactive questions were more likely to be classified as simulations by the team members. Again, no large or consistent differences are visible between the classifications for all tasks versus the classifications for lower-level (0 – II) tasks.

We have constructed a classification for question types. Among the algorithms-related tasks verification questions were the most common in all years except for 2014. In 2014 the most common type of algorithms question was optimization, which was quite popular in other years ranking second for 2010, 2012, and 2013. Constraint questions were also popular, ranking second for 2011, tied for third in 2012, and third in both 2013 and 2014. It was interesting to note that tasks classified as procedures were relatively uncommon except in 2012.

It makes some sense that verification questions are popular for a contest structured around multiple-choice questions. Providing participants with a description of a problem and then asking them to choose among possibilities for the correct answer or scenario is natural in that format. It's also easy to understand why optimization questions would be popular since many computing problems ask for the minimization or maximization of particular values. What is more surprising is the relatively uncommon use of procedures questions, given how important programming is in the later curricula in schools and universities. That the Bebras tasks do not emphasize this more suggests that the contest organizers may be focusing more broadly on CS concepts and trying to deemphasize programming-specific tasks.

Our findings regarding assessment practice showed that teachers find it difficult to assess students' understanding of the concept of algorithm. The predominant presence of algorithmic aspects in Bebras tasks might make this 'tasklet based assessment' interesting for K–9 teachers.

We expect that the Bebras community might benefit from our analysis of tasks from previous contests. For instance, the classification system might be useful to future task developers. Moreover, it would be interesting to investigate whether our task classification can be used to refine existing transnational comparative studies such as [27].

Final Remarks

The results presented in this report provide a multifaceted view of CS concepts in K-9 education and their assessment. This small scale study has shown that the selected analysis methods are promising and follow-up studies could benefit teachers and teacher trainers, curriculum developers and the Bebras community.

10. ACKNOWLEDGEMENTS

Many thanks to Tim Steenvoorden for his help in ana-

lyzing the curriculum documents and to the teachers taking part in the interviews, providing us with valuable information.

11. REFERENCES

[1] L.W. Anderson and D.R. Krathwohl. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives.* Longman, New York, 2001.

[2] Computing at School Working Group. Computer science: A curriculum for schools, March 2012. http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf.

[3] E. Barendsen, V. Dagiene, M. Saeli, and C. Schulte. Eliciting computing science teachers' PCK using the content representation format: Experiences and future directions. In *Proceedings of ISSEP*, pages 71–82, September, 22–24 2014.

[4] E. Barendsen, P. Fisser, J. Krüger, and J. Tolboom. Herziening van het Nederlandse informaticacurriculum havo-vwo, 2014. Paper presented at ORD 2014, Groningen.

[5] E. Barendsen and I. Henze. Teacher knowledge versus teacher practice: reflecting on classroom instruction and interaction through PCK-related observation. In *Proceedings of NARST*, 2012.

[6] E. Baumgartner. Designing inquiry: Contexualizing teaching strategies in inquiry-based classrooms. In *Proceedings of the Annual Conference of the American Educational Research Association*, April, 22 1999.

[7] Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. Computer Science Unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1):20–29, 2009.

[8] Anders Berglund and Raymond Lister. Introductory programming and the didactic triangle. In *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103*, ACE '10, pages 35–44, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[9] J. Biggs. Enhancing teaching through constructive alignment. *Higher Education*, 32:347–364, 1996.

[10] J.B. Biggs and K.F. Collis. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome).* Academic Press, 1982.

[11] Russell Boyatt, Meurig Beynon, and Megan Beynon. Ghosts of programming past, present and yet to come. In Benedict du Boulay and Judith Good, editors, *Proceedings of the 25th Annual Workshop of the Psychology of Programming Interest Group – PPIG 2014*, pages 171–182, 2014.

[12] K. Brennan and M. Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*, 2012.

[13] British Educational Research Association. Ethical guidelines for Educational Research. Technical report, BERA, 2013.

[14] Neil C. C. Brown, Sue Sentance, Tom Crick, and Simon Humphreys. Restart: The resurgence of computer science in UK schools. *Trans. Comput. Educ.*, 14(2):9:1–9:22, June 2014.

[15] Neil Christopher Charles Brown, Michael Kölling, Tom Crick, Simon Peyton Jones, Simon Humphreys, and Sue Sentance. Bringing computer science back into schools: Lessons from the UK. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 269–274, New York, NY, USA, 2013. ACM.

[16] S. Brown and D. McIntyre. *Making sense of teaching.* Open University Press, Buckingham, 1993.

[17] L. Bucciarelli. *Designing engineers.* MIT Press, Cambridge, MA, 1994.

[18] Quinn Burke. The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education*, 4(2):121–135, 2012.

[19] Antonio Cartelli, Valentina Dagiene, and Gerald Futschek. Bebras contest and digital competence assessment: analysis of frameworks. *International Journal of Digital Literacy and Digital Competence*, 1(1):24–39, 2010.

[20] Patrick J. Casey. Computer programming: A medium for teaching problem solving. *Computers in the Schools*, 13(1–2):41–51, July 1997.

[21] Louis Cohen, Lawrence Manion, and Keith Morrison. *Research methods in education.* London, New York: Routledge, 2013.

[22] Stephen Cooper, Lance C. Pérez, and Daphne Rainey. K–12 computational learning. *Commun. ACM*, 53:27–29, November 2010.

[23] CSTA – Computer Science Teachers Association. http://csta.acm.org/.

[24] V. Dagiene. Information technology contests – introduction to computer science in a attractive way. *Informatics in Education*, 5(1):37–46, 2006.

[25] V. Dagiene and J. Skupiene. Learning by competitions: Olympiads in informatics as a tool for training high grade skills in programming. In T. Boyle, P. Oriogun, and A. Pakstas, editors, *Proceedings of the 2nd International Conference on Information Technology: Research and Education*, pages 79–83, Washington, DC, 2004. IEEE Computer Society.

[26] Valentina Dagienė and Gerald Futschek. Bebras international contest on informatics and computer literacy: Criteria for good tasks. In *Proceedings of the 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives: Informatics Education - Supporting Computational Thinking*, ISSEP '08 – LNCS 5090, pages 19–30, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] Valentina Dagiene, Linda Mannila, Timo Poranen, Lennart Rolandsson, and Pär Söderhjelm. Students' performance on programming-related tasks in an informatics contest in Finland, Sweden and Lithuania. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 153–158. ACM, 2014.

[28] M. J. De Vries. Concept learning in technology education. *Journal of Technical Education (JOTED)*, 1(1):147–151, 2013.

[29] A. M. Decker. *How Students Measure Up: An Assessment Instrument for Introductory Computer Science*. PhD thesis, University at Buffalo (SUNY), Buffalo, NY, 2007.

[30] Fadi P. Deek, Starr Roxanne Hiltz, Howard Kimmel, and Naomi Rotter. Cognitive assessment of students' problem solving and program development skills. *Journal of Engineering Education*, 88(3):317–326, 1999.

[31] Department for Education. National Curriculum for England: Computing programme of study. Technical report, Department for Education, 2013.

[32] S. Doukakis, A. Psaltidou, A. Stavraki, N. Adamopoulos, P. Tsiotakis, and S. Stergou. Measuring the technological pedagogical content knowledge (tpack) of in-service teachers of computer science who teach algorithms and programming in upper secondary education. In K. Fernstrom, editor, *Readings in Technology and Education: Proceedings of ICICTE 2010*, pages 442–452, 2010.

[33] Caitlin Duncan, Tim Bell, and Steve Tanimoto. Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, WiPSCE'14, pages 60–69. ACM, 2014.

[34] European Schoolnet. Computing our future: Computer programming and coding – Priorities, school curricula and initiatives across Europe (Update 2015), October 2015.

[35] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4(2):13–17, April 1970.

[36] British Department for Education. Computing programmes of study: key stages 1 and 2. national curriculum in england, 2013. http://www.computingatschool.org.uk.

[37] Committee for the Workshops on Computational Thinking; National Research Council. *Report of a Workshop on The Scope and Nature of Computational Thinking*. The National Academies Press, 2010.

[38] Committee for the Workshops on Computational Thinking; National Research Council. *Report of a Workshop on the Pedagogical Aspects of Computational Thinking*. The National Academies Press, 2011.

[39] D. Fortus, R.C. Dershimer, J. Krajcik, R.W. Marx, and R. Mamlok-Naaman. Design-based science and student learning. *Journal of Research in Science Teaching*, 41(10):1081–1110, 2004.

[40] Jens Gallenbacher. *Abenteuer Informatik*. Springer Spektrum, Heidelberg, 2012.

[41] Gess-Newsome, J. et al. Impact of educative materials and transformative professional development on teachers' PCK, practice, and student achievement. In *Proceedings of the Annual Meeting of the National Association for Research in Science Teaching*, pages 79–83, April 6 2011.

[42] Graham R. Gibbs. *Analysing qualitative data*. Sage, 2007.

[43] J. Goode and G. Chapman. Exploring Computer Science. http://exploringcs.org.

[44] J.I. Goodlad. The scope of the curriculum field. In *Curriculum inquiry*, pages 17–41. New York: McGraw-Hill, 1979.

[45] Nataša Grgurina, Erik Barendsen, Bert Zwaneveld, Klaas van Veen, and Idzard Stoker. Computational thinking skills in dutch secondary education: Exploring pedagogical content knowledge. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, Koli Calling '14, pages 173–174, New York, NY, USA, 2014. ACM.

[46] Shuchi Grover. Robotics and engineering for middle and high school students to develop computational thinking. Paper presented at the Annual Meeting of the American Educational Research Association, New Orleans, LA, 2011.

[47] Shuchi Grover and Roy Pea. Computational thinking in k–12: A review of the state of the field. *Educational Researcher*, 42(1):38–43, 2013.

[48] I. Henze, J.H. Van Driel, and N. Verloop. Development of experienced science teachers pedagogical content knowledge of models of the solar system and the universe. *International Journal of Science Education*, 30(10):1321–1342, 2008.

[49] Chenglie 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, New York, NY, USA, 2011. ACM.

[50] Peter Hubwieser, Michal Armoni, Michail N. Giannakos, and Roland T. Mittermeir. Perspectives and visions of computer science education in primary and secondary (k-12) schools. *Transactions on Computing Education*, 14(2):7:1–7:9, 2014.

[51] Peter Hubwieser, Marc Berges, Johannes Magenheim, Niclas Schaper, Kathrin Bröker, Melanie Margaritis, Sigrid Schubert, and Laura Ohrndorf. Pedagogical content knowledge for computer science in German teacher education curricula. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, WiPSE '13, pages 95–103, New York, NY, USA, 2013. ACM.

[52] Peter Hubwieser and Andreas Mühling. Playing PISA with Bebras. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, WiPSCE '14, pages 128–129, New York, NY, USA, 2014. ACM.

[53] ISTE – International Society for Technology in Education. http://www.iste.org/.

[54] Yasmin B. Kafai and Quinn Burke. The social turn in K–12 programming: Moving from computational thinking to computational participation. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 603–608. ACM, 2013.

[55] KNAW. Digitale geletterdheid in het voortgezet onderwijs. Technical report, Koninklijke Nederlandse Akademie van Wetenschappen, 2012.

[56] J.L. Kolodner. Case-based reasoning. In K.L. Sawyer, editor, *The Cambridge handbook of the learning sciences*, pages 225–242. Cambridge University Press, 2006.

[57] H. Koppelman. Pedagogical content knowledge and educational cases in computer science: An exploration. In *Proceedings of the Informing Science & IT Education Joint Conference (InSITE)*, pages 125–133, Santa Rosa, CA, June 22-25 2008. Informing Science Institute.

[58] J.S. Krajcik and P. Blumenfeld. Project-based learning. In K.L. Sawyer, editor, *The Cambridge handbook of the learning sciences*, pages 317–333. Cambridge University Press, 2006.

[59] D. Midian Kurland, Roy D. Pea, Catherine Clement, and Ronald Mawby. A study of the development of programming ability and thinking skills in high school students. *J. of Educational Computing Research*, 2(4):429–458, 1986.

[60] Irene Lee, Fred Martin, and Katie Apone. Integrating computational thinking across the k–8 curriculum. *ACM Inroads*, 5(4):64–71, December 2014.

[61] J. Loughran, A. Berry, and P. Mulhall. *Understanding and Developing Science Teachers' Pedagogical Content Knowledge*. Sense Publishers, Rotterdam, The Netherlands, 2006.

[62] J. Loughran, P. Mulhall, and A. Berry. In search of pedagogical content knowledge in science: Developing ways of articulating and documenting professional practice. *Journal of Research in Science Teaching*, 41(4):370–391, 2004.

[63] James J. Lu and George H.L. Fletcher. Thinking about computational thinking. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, pages 260–264. ACM, 2009.

[64] S. Magnusson, J. Krajcik, and H. Borko. Nature, sources, and development of pedagogical content knowledge for science teaching. In J. Gess-Newsome and N. G. Lederman, editors, *Examining pedagogical content knowledge: The construct and its implications for science education*, pages 95–132. Kluwer, 1999.

[65] Linda Mannila, Valentina Dagiene, Barbara Demo, Natasa Grgurina, Claudio Mirolo, Lennart Rolandsson, and Amber Settle. Computational thinking in K–9 education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, ITiCSE-WGR '14, pages 1–29, New York, NY, USA, 2014. ACM.

[66] R. McCartney and J. Tenenberg, Eds. *Trans. on Computing Education – Special Issue on Computing Education in K–12 Schools*, 14(2), June 2014.

[67] Jerry Mead, Simon Gray, John Hamer, Richard James, Juha Sorva, Caroline St. Clair, and Lynda Thomas. A cognitive approach to identifying measurable milestones for programming skill acquisition. In *Proc. of the 11th Conference on Innovation and Technology in Computer Science Education (ITiCSE '06)*, 2006.

[68] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. Learning computer science concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, pages 69–76, New York, NY, USA, 2010. ACM.

[69] Roland Mittermeir, Ernestine Bischof, and Karin Hodnigg. Showing core-concepts of informatics to kids and their teachers. In *ISSEP 2010*, volume 5941 of *LNCS*, pages 143–154. Springer, 2010.

[70] MIUR. Syllabus di elementi di informatica la scuola dell'obbligo – anno 2010, December 2010. Working document.

[71] Lijun Ni and Mark Guzdial. Who am I?: understanding high school computer science teachers' professional identity. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 499–504, New York, NY, USA, 2012. ACM.

[72] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[73] Simon Papert. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1):95–123, 1996.

[74] Soonhye Park and Jee Kyung Suh. From portraying toward assessing pck: Drivers, dilemmas, and directions for future research. In A. Berry, P. Friedrichsen, and J. Loughran, editors, *Re-examining pedagogical content knowledge in science education*, pages 104–119. Routledge Press, London, 2015.

[75] Roy D. Pea and D. Midian Kurland. On the cognitive effects of learning computer programming. *New Ideas in Psychology*, 2(2):137–168, 1984.

[76] Mitchel Resnick et al. Scratch: programming for all. *Communications of the ACM*, 52:60–67, 2009.

[77] Ralf Romeike. What's my challenge? the forgotten part of problem solving in computer science education. In *Proceedings of the 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives: Informatics Education - Supporting Computational Thinking*, ISSEP '08, pages 122–133, Berlin, Heidelberg, 2008. Springer-Verlag.

[78] M. Saeli. *Teaching programming for secondary school: a pedagogical content knowledge based approach*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2012.

[79] D.A. Schön. *The reflective practitioner: How professionals think in action*. Basic Books, New York, US, 1983.

[80] Carsten Schulte. Reflections on the role of programming in primary and secondary computing education. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, WiPSE '13, pages 17–24, New York, NY, USA, 2013. ACM.

[81] Deborah Seehorn, editor. *K-12 Computer Science Standards – Revised 2011: The CSTA Standards Task Force*. ACM, October 2011. Deborah Seehorn, Chair; CSTA - Computer Science Teachers Association.

[82] Judy Sheard, Angela Carbone, Raymond Lister, Beth Simon, Errol Thompson, and Jacqueline L. Whalley. Going SOLO to assess novice programmers. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 209–213, New York, NY, USA, 2008. ACM.

[83] Lee S. Shulman. Those who understand: Knowledge growth in teaching. *Educational Researcher*,

15(2):4–14, 1986.

[84] Christopher W. Starr, Bill Manaris, and RoxAnn H. Stalvey. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, SIGCSE '08, pages 261–265. ACM, 2008.

[85] T. Steenvoorden. Characterizing fundamental ideas in international computer science curricula. Master's thesis, Radboud University, The Netherlands, 2015.

[86] C. Taylor et al. Computer science concept inventories: past and future. *Computer Science Education*, 24(4):253–276, 2014.

[87] Allison Elliott Tew and Mark Guzdial. Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 97–101, New York, NY, USA, 2010. ACM.

[88] The Royal Society. *Shut down or restart? The way forward for computing in UK schools.* London: The Royal Society, 2012.

[89] A. Thijs, P. Fisser, and M. Van der Hoeven. *21ste eeuwse vaardigheden in het curriculum van het funderend onderwijs.* Enschede: SLO, 2014.

[90] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for CS assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[91] Allen Tucker, editor. *A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee.* ACM, New York, NY, USA, October 2003. Allen Tucker, Chair; CSTA - Computer Science Teachers Association.

[92] J.J.H. Van den Akker. Curriculum perspectives: An introduction. In J.J.H. Van den Akker, W. Kuiper, and U. Hameyer, editors, *Curriculum landscapes and trends*, pages 31–10. Springer, 2003.

[93] Annette Vee. Understanding computer programming as a literacy. *Literacy in Composition Studies*, 1(2):42–64, 2013.

[94] K.B. Wendell and H.-S. Lee. Elementary students learning of materials science practices through instruction based on engineering design tasks. *Journal of Science Education and Technology*, 19(6):580–601, 2010.

[95] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pages 215–220. ACM, 2012.

[96] Jacqueline Whalley and Raymond Lister. The BRACElet 2009.1 (wellington) specification. In Margaret Hamilton and Tony Clear, editors, *Eleventh Australasian Computing Education Conference (ACE 2009)*, volume 95 of *CRPIT*, pages 9–18. Australian Computer Society, Inc. (ACS), 2009.

[97] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[98] Jeannette M. Wing. Computational thinking: What and why? *The Link Magazine*, 2011.

[99] D. Wongsopawiro. *Examining science teachers' pedagogical content knowledge in the context of a professional development program.* PhD thesis, Leiden University, Leiden, The Netherlands, 2012.