Assessment in Computer Science courses: A Literature Review

Maria Kallia

King's College London

Table of contents

Introduction

Assessment is one of the most critical dimensions of the education process; it focuses not only on identifying how many of the predefined education aims and goals have been achieved but also works as a feedback mechanism that educators should use to enhance their teaching practices. Assessment is located among the main factors that contribute to a high quality teaching and learning environment.

The value of assessment can be seen in the links that it forms with other education processes. On this matter, Lamprianou and Athanasou (2009:22) point out that assessment is connected with the education goals of "diagnosis, prediction, placement, evaluation, selection, grading, guidance or administration". Consequently, assessment is a critical process that provides information about the effectiveness of teaching and the progress of students and also makes clearer what teachers expect from students (Biggs, 1999).

Evaluating students' knowledge and learning in computing courses is challenging. The fact that computer science is a multifaceted subject that includes the development of various and compound skills - such as computational thinking skills - makes the design of appropriate, valid and rigorous assessment approaches that satisfy the learning objectives non-trivial. An additional difficulty in this process is how assessment approaches can be adapted to match the specific needs of each class. In any case, a computing curriculum needs to ensure that assessment is conducted in a way that meets the learning objectives and is beneficial and helpful for both teachers and students.

The purpose of the literature review is to outline research studies in the assessment of computing courses and to highlight the studies that can be used in the assessment of school computing in the UK. The main objective is to summarise what is currently known about the effective assessment of computer science courses and to identify gaps in knowledge where future research should focus. Specifically, the current literature review attempts to address the following research questions:

- What research has been undertaken nationally and internationally into the assessment of computing in schools and higher education (relevant to schools)?
- What are the key findings from this research?
- What are the limits of this research?

- Are there research findings that could be applied to computing assessment in UK schools?
- Are there any clear priorities for future research in this area or areas that would not benefit from future research?

Research Methodology

The research studies were collected from a variety of sources: ACM Digital Library, IEEE, Taylor and Francis, Elsevier, Wiley Online Library, Eric and Google Scholar. The search terms used to identify these papers were:

- (assessment or evaluation or testing) and (computing or computing courses or computer science or computational thinking)
- (automated or design) and assessment tools and (computing or programming or computational thinking)

The search was not bounded by a specific year range. The list of references of the papers identified were used to find additional papers, a technique known as snowballing[1]. This search resulted in over 140 papers and included studies in higher, primary and secondary education. However, there is limited research about assessment in primary and secondary education computing; most research has been conducted for higher education computing subjects. Research in primary or secondary education focuses primarily on how to evaluate students' projects on Scratch and how to assess computational thinking in this environment.

A total of 66 papers in the assessment of computing courses were selected to be presented in this review and these are listed in Appendix 1. The criteria for selecting these papers were:

a) their relevance to the UK school computing curriculum
b) the potential adaptation of the assessment methods suggested in UK's schools and
c) the clarity of the study.

The years that these studies were published range from 2001 to 2016. It should be noted that in the case of assessing programming only recent studies were included. This is because the

[1] "Snowballing refers to using the reference list of a paper or the citations to the paper to identify additional papers" (Wohlin, 2014:38)

tools employed in these studies include functionalities that are absent in older tools but are essential for assessing students' programs efficiently.

The studies included a different number of participants: small-scale studies refer to 50 participants or less, middle-scale studies refer to studies that employed more than 50 and less than 300 participants while large-scale studies refer to more than 300 participants. To present a more comprehensive picture of these studies, some statistical information regarding their characteristics is described in the following paragraphs.

Figure 1 illustrates the number of papers published between 2001 and 2016. As can be seen from this chart, most of the studies were conducted between 2013 and 2016. This indicates an increase in the interest in assessment and in effective assessment tools in computing.

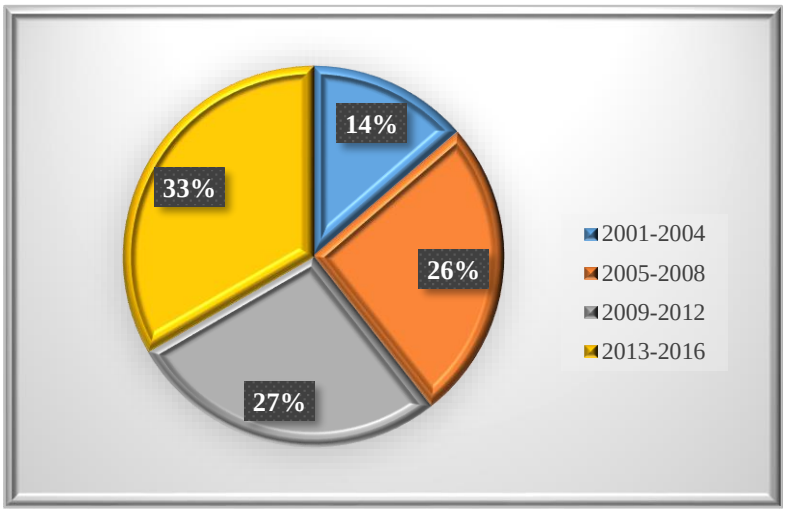

Figure 1 Percentage of studies per four years

In the analysis of these papers, four distinct themes appeared repeatedly:

1. assessment approaches,
2. developing assessment instruments,
3. assessment tools, and
4. assessment instruments.

The studies were categorised according to these four themes. Some articles can be categorised to more than one theme but the major one was selected as the most representative. The first theme includes research studies that explore the way that assessment is carried out and it is further divided in self-assessment and peer assessment (based mostly on a web-based marking

system) approaches. The second theme is focused on the design of assessment instruments and includes teaching and learning theories such as Bloom's and SOLO Taxonomy and different types of questions and metrics that add value to the assessment process. The third theme focuses on automated tools to assess programming and computational thinking skills[2]. The last one is concerned with other instruments that can be used to assess students' learning in computing, such as concept maps, code comprehension and debugging tasks, multiple-choice questions and quizzes.

Figure 2 depicts the number of papers per theme, showing that most of these studies refer to the way that instruments for assessing computing should be developed (theme 2: developing assessment instruments). Additionally, "assessment approaches" is the category with the lowest number of studies which may indicate a gap in the literature and a direction for future research.

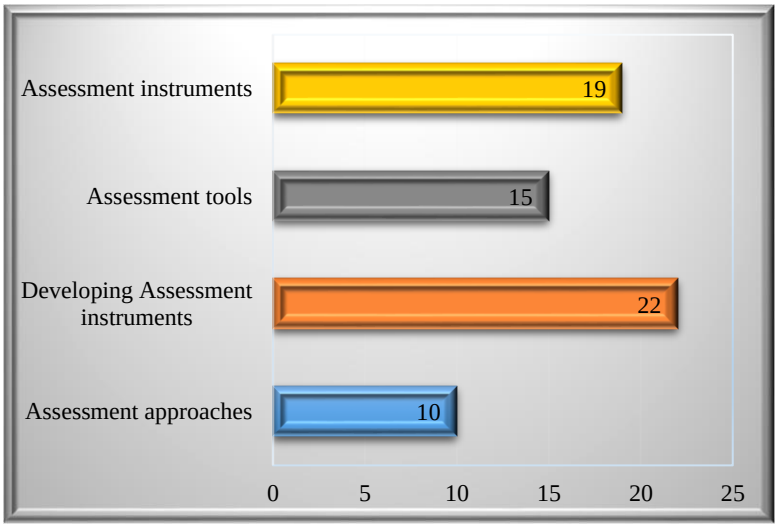

Figure 2 Number of studies per theme

The vast majority of the papers have been conducted in higher education but some could be applied with few modifications in primary or secondary schools. Of the 66 studies selected to be part of this review, 49 were conducted in higher education, with 17 relating to primary or secondary education. Figure 3 portrays the number of studies conducted by country while Figure 4 demonstrates the number of studies conducted per phase of education (primary, secondary, higher) in each of these countries. As it is illustrated by the charts, most of the studies were conducted in the USA. USA also concentrates most of the studies in all phases of

[2] The papers about computational thinking were classified in the "assessment tools" theme because most of them are based on tools like Scratch, Alice, REACT etc. to assess computational thinking.

education. Other countries that conducted research in primary or secondary education include Spain, UK, Israel, Taiwan and Greece.

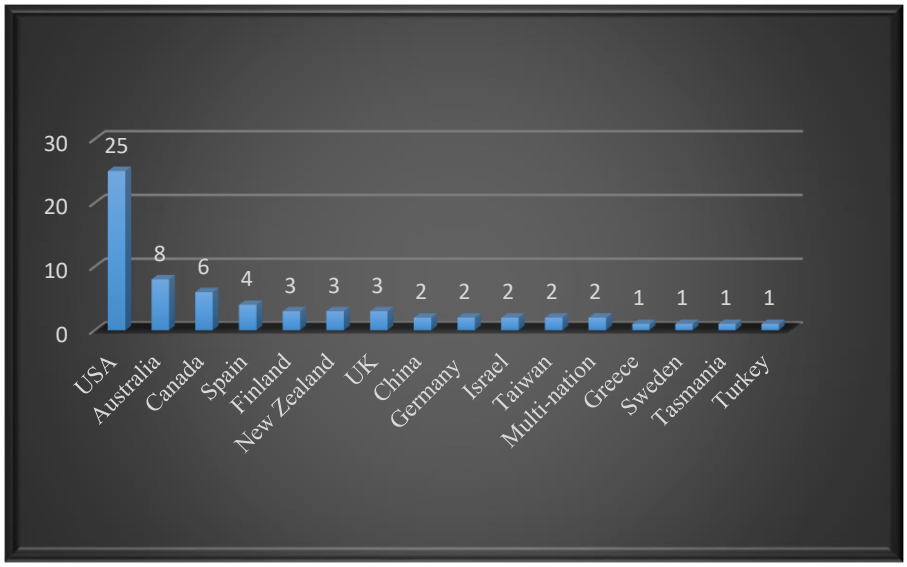

Figure 3 Number of studies per country

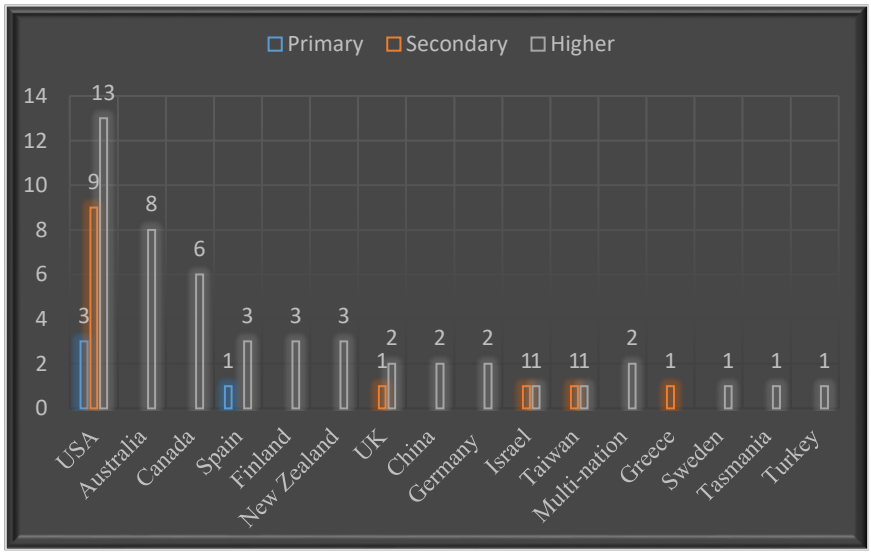

Figure 4 Number of studies per phase of education and country

Some of the studies that are presented in this review are proposed as suitable to replicate in schools. These suggestions are based on the following criteria:

- The skills assessed in the corresponding studies are skills that are also assessed in school settings.
- The approach employed in the corresponding studies appears to have benefits and advantages on students' learning and skills.

- The tool or approach employed in the corresponding studies can be easily used by school teachers. This indicates that teachers will not need any further or extensive training in adapting the tool and approach.
- The tool or approach employed in the corresponding studies can be used by students with teachers' guidance.
- The approach proposed by the corresponding studies has been employed by other disciplines in school settings.

The remainder of this study is organised as follows: Firstly, we present the studies of each of the themes described previously (see Appendix 1 for the list of papers per theme). Secondly, we outline the key findings of this study. We then consider assessment approaches and instruments that could be adopted in primary or secondary computing in the UK and finally recommend areas in which future research in assessment in computing should focus.

Assessment in Computing courses

Assessment approaches

Self-assessment

Self-assessment is regarded as one of the most efficient ways to engage students in learning (McMillan & Hearn, 2008). In fact, researchers' interest in this type of assessment was greatly influenced by the changes in teaching and learning and, specifically, the change from a teacher-centred model to a learning environment that actively engages students and urges them to take responsibility for their learning and skills (Spiller, 2012). Indeed, assessment practices that only involve the teacher are contradictory to the constructivist learning environments and limit the students' progress in all aspects (Spiller, 2012). In contrast, getting students involved with theirs or others' assessment engages them with a learning and an assessment process which brings many benefits (Sitthiworachart & Joy, 2003). The students become responsible for their learning and get involved in reflective practices to evaluate their performance and guide their learning actions respectively. This need of students to take control of their learning is what makes self-assessment important in the evaluation process (Spiller, 2012).

Boud (1991) defines self-assessment as "*the involvement of students in identifying standards and/or criteria to apply to their work, and making judgements about the extent to which they have met these criteria and standards*" (Boud, 1991 cited in Kay et al., 2007:89). By this definition, Boud (1995) highlights two main aspects of assessment: the first is setting the standards of the levels of achievement expected and the second is evaluating the value of achievement in the light of those standards. By getting involved with both these aspects of assessment, students set goals for their learning and start learning for themselves (Hanrahan & Isaacs, 2001). Thus, the benefits of self-assessment can be summarised as follows: a) self-assessment can provide motivation for further engagement in learning and can also enhance the learners' responsibility and independence as they take control of their learning (Spiller, 2012) b) self-assessment gives a chance to learners to shift from the view of satisfying their teachers and to focus on a better quality of learning for themselves (Boud, 1995).

In computer science courses, self-assessment plays an important role and it is a useful tool for tracking students' learning. In fact, García-Beltrán and Martinez (2006) created a web-based self-assessment environment to enhance students' motivation in programming. Setting aside

the assessment character of this approach, they were also interested in creating an environment that would encourage students to practice programming. Students can select a variety of tests and also re-take them if they want to re-assess their learning. This is an important characteristic of automated tools which will be addressed in a later section (pg. 23). García-Beltrán and Martinez (2006) reported that students' feedback was very positive; the students also stated that they worked harder and that they were more motivated. The authors also pointed out that self-assessment and immediate feedback seem to increase students' levels of confidence. Likewise, Gayo-Avello and Fernádez-Cuervo (2003) were keen to explore ways that could improve students' learning specifically on concepts. To this end, they employed an online multiple-choice, self-assessment tool and evaluated this technique on the effects on students' learning at the University of Oviedo (Spain). Their findings mirror the ones of García-Beltrán and Martinez (2006) and suggest that self-assessment is a useful method that improves students' learning.

Research supports that it is also important for students to get involved in approaches that assess their perceptions about the level of their learning. Such a study was conducted by Murphy and Tenenberg (2005) at an American University and involved 61 students. The researchers explored the students' perspectives on the level of their knowledge in data structures. They argue that self-knowledge, or in psychology terms calibration of knowledge, is critical because it has a vital role in metacognitive skills and especially in self-regulation. To measure students' knowledge, they employed quizzes with multiple-choice questions referring to data structures. Additionally, the students completed questionnaires that evaluated their calibration ability, an estimation of how well they performed on the quiz. Their findings indicate that students' estimates about themselves correlated with how they performed in the quiz, although the calibration accuracy of weaker students was less than that of the stronger ones. This needs further discussions about the role that this type of assessment could have in computing courses; true reflection and calibration can help students recognise their learning needs while, imprecise calibration, like over-estimation, may lead to failure (Sheldrake et al., 2014). Accordingly, Murphy and Tenenberg (2005) highlight the important role of calibration in computing courses.

Peer Assessment

Another type of assessment is peer assessment. Boud and Falchikov (2007) define peer assessment as a process of providing feedback on peers' work based on success criteria that the students may previously have established while Topping (2009) describes peer assessment

as the learners' agreements to examine and determine the quality of a product. Consequently, self-assessment involves the learner with a self-evaluation process while peer assessment engages students in the process of making judgements on their peers (Somervell, 1993).

The advantages of employing peer assessment include many aspects. To begin with, Topping et al. (2000) argue that this type of assessment helps students to really understand the aims and objectives of a course. Thus, students have a better understanding of the criteria against which they are assessed. Additionally, students collaborate with each other to identify assessment criteria and to make judgements of each other's work; this leads to the development of a community whose primary focus is to work together to help each other (Nulty, 2011). Additionally, the feedback that the students get from their peers seems to enhance their motivation for improvement more than the judgements of their professors (Searby and Ewers, 1997). Finally, Topping (2009) argues that peer assessment in the form of formative assessment enables learners to work together and support each other to organise their learning, recognise their strong points and their weaknesses, identify parts that need improvement, and finally, develop a range of skills. These skills include writing skills (Liu et al., 2002), metacognitive, personal and professional (Topping, 2009), critical thinking skills (Davies & Berrow, 1998; Wolfe, 2004) and self-efficacy (Anewalt, 2005).

Many research studies have explored the use of peer assessment in different fields. Specifically, in computer science, most of the studies have been conducted in higher education settings, and few of them evaluate this assessment technique in secondary education. Also, most of the research studies focus on using an online tool for peer assessment and few studies are based on a paper-based marking scheme. The benefits of employing a web-based peer assessment are outlined by Lin et al. (2001): firstly, the fact that the students assess their peers through an online system and not face to face ensures anonymity which is a major factor for involving students in this process; secondly, a web-based peer assessment offers flexibility to teachers since they can control students' progress during the assessment process at any time; finally, web-based peer assessment is a cost-free technique (in terms of money and time) since the students do not need to photocopy their assignments or work for their assessors.

Influenced by the promising benefits of peer assessment in learning and improving students' skills, researchers in the field of computer science evaluate this technique on a range of computing courses and on students' learning and skills. An illustrative example of these studies is that described by Sitthiworachart and Joy (2003) at a UK university. The researchers were

interested in exploring the effects of peer assessment on learning computer programming and in evaluating the validity of the students' assessments. Peer assessment was based on a web-based platform that ensured anonymity. Specifically, the researchers decided to evaluate students' assignment based on the average of three marking schemes: the teacher's evaluation of the assignment, an automatic tool's mark, and finally, the mark that the peers assigned. What is interesting, though, is that the researchers, to increase the validity of the peer assessments, gave the opportunity to students to assess the quality of the evaluation they received. In fact, if the evaluation was ineffective, then the peer's grade was disregarded and not included in students' final mark. To evaluate the effects of this process, the researchers employed a questionnaire that the students had to answer. Their results indicate that a high percentage of students recognised errors in their assignments by evaluating the peers' work. Moreover, most of the students were satisfied with their peers' marking, and they believed that the feedback they received was useful and helpful. Finally, most of the students suggested that comparing good and bad programs helped them with developing theirs while marking urged them to consider more their assignments. Sitthiworachart and Joy (2003) admitted that students tended to over-mark their peer friends, which is a valid issue, but concluded that, overall, their peer system provides evidence that peer assessment benefits students' learning.

On the same grounds, Clark (2004) and Smith et al. (2009) reported many benefits of peer testing[3] based on their personal experience adapting this technique in software engineering courses. Clark's (2004) study was conducted at a Tasmanian university and included 108 students. Specifically, Clark highlights that with peer testing students have the chance not only to realise the importance of software testing but also to increase the quality of their work and the levels of their understanding as they communicate and collaborate with each other. Actually, he argues that a strong sense of community is created in the class and collaboration between students is increased. In the same line, Smith et al. (2012) advocate peer review or peer testing and state that such a technique is not limited to identifying errors or bugs in students' programs, but also provides a deeper understanding of the students' sources of errors. They also mention that by employing this assessment approach, their students showed a positive reaction and they reported that they found this technique very informative and helpful.

Tseng and Tsai's (2007) study resulted in the same conclusions. However, their study differentiates from the previous ones in three ways. Firstly, their study was conducted in high

[3] Peer testing or peer review is an effective way to identify code defects; a group of peers is working together to detect these errors (Clark, 2012).

schools in Taiwan (in a computing course about the use of the Internet). Secondly, the researchers included an evaluation of different types of feedback on the quality of students' work. The researchers categorised the form of feedback using Chi's (1996) categories: corrective, reinforcing, didactic and suggestive. Thirdly, the researchers incorporated a qualitative evaluation of the students' assignments (formative feedback) based on three dimensions: creativity, relevance and feasibility. The process that they followed included a three-round web-based peer assessment and included 184 students. In each round, the students were asked to improve their designs according to their peers' guidelines. In this way, every student had three roles: "author, assessor, and adapter" (Tseng and Tsai, 2007: 1165). After the analysis of their data, the authors observed that students' score on each of the three dimensions (creativity, relevance, feasibility) increased on each round. This suggests that the quality of students' project benefit from on-line peer assessment. Additionally, the teachers' and students' scores were highly correlated which led the authors to suggest that on-line peer assessment in high school is a valid method. Finally, with regards to the type of feedback and its impact on students' performance, their findings conclude that reinforcing and suggestive feedback have an active role in the quality of students' work. However, didactic feedback produced an adverse effect on students' performance.

The literature suggests that peer assessment enhances students' higher level thinking skills and metacognitive skills. To investigate this, Liu et al. (2001) explored the effects of peer review in the course of operating systems at a Taiwanese university. In total, 143 students participated. Adopting almost the same assessment process with the studies above, Liu et al. (2001) suggest that web-based peer review is an effective strategy in computer science courses. Specifically, many students reported that peer review, and especially the process of comparing their projects with their peers, helped them realise their strengths and weaknesses. The authors also argued that during this process students demonstrate skills such as critical thinking and regulation and are more motivated to learn. Nevertheless, the authors admitted that their approach does not handle some validity problems like the students' tendency to under-grade their competitors or overrate their friends.

In comparison with these studies, the findings of Chin (2005) and Sajjadi et al. (2016) do not seem so promising. Both studies concentrate on evaluating peer assessment on a course of algorithms. However, they do not indicate any negative correlation between the course and the assessment technique. In particular, Chin's (2005) study intended to identify if the evaluation and critical judgement skills can be taught in computer science classrooms and to what level

this assessment technique identifies misconceptions or misunderstandings in this course. The methodology involved traditional methods (homework, quizzes, exams) as well as peer assessment activities. Findings showed no significant change in students' evaluation and critical judgement skills. Additionally, the type of errors that the students identified were conceptual rather than technical. Finally, Sajjadi et al. (2016) adopted a three-way grading mechanism which included: self-assessment and grading, peer grading, and teaching assistant grading to evaluate peer assessment on an Algorithmic and Data Structures course at Hamburg University (sample size: 219). The findings of their study refer more to the students' attitudes towards marking. For example, the results indicate a natural tendency of students to self-grade themselves higher than do their peers who also tend to set higher grades than the teaching assistants. This may reveal students' inefficacy of finding errors or may reveal students' bias as the above studies also indicate. Additionally, students who grade themselves higher are likely to be more dependable in grading peers, but at the same time, the researchers recognise the existence of negative bias (students with good performance may grade strictly and with high standards when grading their peers).

Summary

This section described two approaches on assessment in computing courses: self-assessment and peer assessment. The majority of the studies presented here have been conducted in higher education. Specifically, nine studies have been conducted in universities (USA, Spain, Taiwan, UK, Tasmania, Germany) and only one study evaluated peer assessment in high school settings (Taiwan). Most of the research studies were medium-scale and indicated positive effects on students' learning and skills by employing self and peer assessment. It seems that these approaches can be used in school settings but more studies are needed to evaluate the validity of peer assessment in computing classes in school.

Developing assessment instruments

Bloom's Taxonomy

The idea behind Bloom's Taxonomy is the arrangement of educational objectives in a hierarchy, beginning from less to more complicated levels. To reach a higher level, one must first conquer the lower ones. The original levels of this hierarchy are: Knowledge,

Comprehension, Application, Analysis, Synthesis and Evaluation (Bloom et al., 1956) and describe three domains of knowledge: the cognitive, affective and psychomotor. Anderson and Krathwohl (2001) suggested a revised taxonomy which includes the following levels: Remembering, Understanding, Applying, Analysing, Evaluating, Creating.

Bloom's Taxonomy has frequently been used to categorise curriculum objectives and in assessment to indicate the depth of the objectives (Krathwohl, 2002). In computer science courses, some studies attempt to adopt Bloom's Taxonomy to create or categorise assessment questions. This need was highlighted by Scott (2003) who argues that, in an assessment test, it is a common phenomenon that most of the students can answer questions from a low-level category and few students can answer the most advanced questions. He further argues that on designing a test, the author must pay attention to include questions from all six levels of Bloom's Taxonomy. The educator will thus have a more accurate reflection of students' learning. Likewise, Lister and Leaney (2003) were also concerned about the different levels and skills that students demonstrate in a computer science course. They argue that most assessment tasks aim at the average student, leaving behind the weaker students and not challenging the stronger ones. In their study, they suggest using different tasks based on Bloom's Taxonomy to address this problem.

Based on this premise, Starr et al. (2008) suggest the use of Bloom's Taxonomy in computer science courses to identify and improve learning objectives before assessment. A characteristic attempt to match assessment questions with Bloom's Taxonomy is illustrated in Thompson et al.'s (2008) study. The authors collected exam scripts from six institutions which included true-false questions, multiple-choice, and open-ended questions from programming courses. They attempted to classify each of these questions to Bloom's revised Taxonomy which, as they reported, was not a clear process (mostly because of the verbs used to describe computer programming tasks). The analysis of the exam papers and the successful categorisation of the questions led the researchers to conclude that Bloom's Taxonomy is an effective way to design exam questions and assessment processes in programming. However, another interesting issue that arose during their research was that each individual solves the task by employing different cognitive skills. This indicates that to assess the level of effort a question requires, an educator must investigate the question's context and consider the cognitive effort required by most of the students.

Another study connecting Bloom's Taxonomy with the design of assessment tools was conducted by Alaoutinen and Smolander (2010) in a higher institution in Finland and included 48 students. The researchers explored students' abilities in evaluating themselves and classifying their knowledge in relation with Bloom's Taxonomy. They argue that this will enable teachers to understand and monitor students' levels of learning by only giving students a self-assessment survey. The authors created a taxonomy-based scale, web-based questionnaire in programming with which students could evaluate their skills and knowledge (1 to 6 level) and answer in open-ended questions about their experience with the questionnaire. There was a statistically significant correlation between students' final grades on the course and students' self-evaluation, indicating that students seem to be able to self-assess the level of their learning, although, there were cases where students overestimated their skills or underestimated them. In general, the researchers argue that this is a valuable and easy to apply assessment tool for the teachers. They suggest that using such an assessment instrument could help teachers frequently to measure students' levels of understanding and organise their teaching more effectively.

Along the same lines, Chatzopoulou and Economides (2010) presented a web-based adaptive system for assessing students' learning in a high school programming course in Greece. The questions were classified according to Bloom's Taxonomy. The difference with other existing tools is that this is an adaptive one which means that if a student accurately responds to a question, the system would present another, more difficult question. In a different case, the student returns to an easier one. They evaluate the usability of their tool by employing 78 students. Their findings suggest that this tool can predict students' final grade in the Greek National Exams in the computer programming course and also that students can identify their levels of learning. From the teachers' perspectives, they could also use such a tool to assess their students or identify the curriculum areas that need more focus.

SOLO Taxonomy

Another Learning Taxonomy was also suggested by Biggs and Collis (1982) who proposed a way to assess and classify cognitive performance based on the organisation of the product of learning. They proposed SOLO Taxonomy (Structure of Observed Learning Outcomes), a hierarchical model that is appropriate for determining levels of understanding and measuring learning results (Biggs & Collis, 1982). The levels of the SOLO Taxonomy are: Prestructural,

Unistructural, Multistructural, Relational, and Extended Abstract. In its simplest form, the SOLO Taxonomy forms four levels "one idea, multiple ideas, relating the ideas and extending the ideas" (Hattie & Brown, 2004:5). The taxonomy has been mostly employed to evaluate students' learning; the higher the categorisation of students' knowledge the deeper their learning is (Burnett, 1999).

In the field of computing, the BRACElet[4] group has conducted research in the use of SOLO Taxonomy and students' understanding of programming problems. In particular, Lister et al. (2006) were interested in classifying students' responses on an exam question (explain verbally what the code does) according to SOLO's first four levels. Their results led the researchers to formulate the argument that students who cannot respond in relational (Relational level) terms cannot write code with the same cognitive demands. A later work by the same group generated results convincing enough to lead the researchers in designing an assessment instrument based on SOLO Taxonomy that could assess students' skills to higher levels of the Taxonomy (Sheard et al., 2008).

Seiter (2015), to assess students' computational thinking, evaluated students' responses based on the SOLO Taxonomy. The study was conducted in a primary school in the USA and included 72 students. Seiter (2015) argues that SOLO can be employed to identify students' levels of understanding of a problem's structure. Likewise, Izu et al. (2016) explored students' levels of progress after 12 weeks of instruction on iteration and vectors at an Australian university. To achieve this, they evaluated students' answers in two questions of the final exam by using the SOLO Taxonomy. The researchers concluded that SOLO Taxonomy is an effective way to assess students' skills in programming.

Finally, Meerbaum-Salant et al. (2010) created questionnaires and tests on Scratch to test students' learning on concepts in computer science at an Israeli middle-school (sample size: 46). The authors employed a combination of the revised Bloom's Taxonomy (Understanding, Applying, Creating) and SOLO Taxonomy (Unistructural, Multistructural and Relational) to build the questionnaire. Their taxonomy uses SOLO categories as super-categories each of which has Bloom's categories as sub-categories. In the same line, Shuhidan et al. (2009) explored students' understanding of writing code by using Bloom's Taxonomy to categorise the multiple-choice questions and SOLO to classify students' responses. Shuhidan et al. (2009)

[4] The BRACElet project studied the undergraduate students' performance on exam papers to understand students' difficulties in programming. The group consisted of the following researchers: Jacquelline L. Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P.K. Ajith Kumar, Cristine Prasad.

suggest that for classifying exam questions, the designer should consider Bloom's Taxonomy, the instructors' estimation of questions' difficulty and students' percentage of correct answers. For categorising students' responses on writing code and their understanding, the SOLO Taxonomy is more useful.

Isomorphic and Parameterised questions

Questioning is a useful way for teachers to understand their students' current state of learning. One type of questions that has been investigated for its effectiveness is isomorphic questions. These are questions that are used to examine the same concept but with some differences in their structure and the way they are formulated. Smith et al. (2009:123) point out that these questions have different "cover stories", but they are approached by using the same principles or concepts. They can also be used to evaluate the degree to which learning, based on Peer Instruction, is generalisable (Zingaro & Porter, 2015). In general, isomorphic questions help instructors to assess learning gains by a discussion with peers (Porter et al., 2011).

In computing, most of the work in this area has been conducted by Porter and colleagues in Canada. Specifically, research in this field indicates that the first question (of an isomorphic pair) and the learning that takes place as a result of this question indicates the students' answer in the second one (Porter et al., 2011; Zingaro and Porter, 2014; Zingaro and Porter, 2013). Another work of Porter et al. (2014) explored the relationship between clicker questions and student performance. The authors combined peer instruction with multiple-choice clicker questions in an undergraduate computer science class to identify questions that could estimate students' achievement on a final exam in programming. Among these questions, the isomorphic ones were found to be highly predictive (Porter et al., 2014). In a later work of the same group, Zingaro and Porter (2015) assessed students' learning with this type of question in a computer science course. The researchers were mostly interested in evaluating peer instruction and, specifically, how class learning is reflected in student performance on a final exam by employing isomorphic questions.

Another type of questions is the Parameterised questions which have been proposed as one of the most promising assessment instruments. Sosnovsky et al. (2003:7) describe these questions as "*a pattern of a question which at the presentation time, the pattern is instantiated with randomly generated parameters from a particular set*". Thus, because a parameterised question

is essentially a pattern of a question, it can generate a great number of different questions with the same assessment goal. The benefits of these questions are well summarised by Sosnovsky et al. (2003) who argue that a) a small number of parameterised questions can generate assessment for large classes and b) the same parameterised question can be used with a different context as a self-assessment instrument.

Pathak and Brusilovsky (2002) discuss an approach for creating web-based parameterised quizzes, especially for programming subjects. The authors first started by describing the problem of using static quizzes for assessment: cheating and lack of material. To address these problems, they suggest the use of parameterised questions. They developed a system, QuizPACK, that helps teachers to form parameterised evaluation questions and quizzes and they evaluated it with 27 students who wanted to practice their learning. The students' feedback was positive indicating that they were really helped in their learning. The opportunity of retaking a parameterised quiz was assessed as a very useful functionality. A further study conducted by Brusilovsky and Sosnovsky (2005) indicates that this is an excellent tool for self-assessment, improving students' learning as well as their higher-level skills.

Rubrics and Primary Trait Analysis

An important issue in assessment is how teachers can reliably and fairly evaluate their students. Rubrics provide a way of offering more consistent assessment and grading of students. They are used to define detailed criteria that describe what students should achieve (Becker, 2003) and they have two shared characteristics. The first is a list of criteria that are necessary for the particular task and the second is the degrees of quality (Andrade, 2000). These features make rubrics proper tools to reliably evaluate students' performance, but they can help students understand their progress as well (Black & Wiliam, 2009). Popham (1997) refers to rubrics as instructional illuminators.

Rubrics have been used to assess computer programming assignments from the early 1980s (Miller and Peterson, 1980; Hamm et al., 1983). More recently, though, Becker (2003) highlights the need for proper feedback for students and a reliable and valid marking scheme that can be employed in a sensible time. He suggests the use of rubrics to handle this problem; the rubrics would describe what students should accomplish to satisfy or surpass the assessment requirements.

Barney et al. (2012) evaluated rubrics and oral feedback effects on students' learning at a Swedish university. Overall, 67 students participated in the study. The authors argued that rubrics help students to understand better what their teachers expect from them. Additionally, students think that this is a fairer system and, thus, complaints about grades are reduced. However, the researchers were reluctant to make any generalisations about the technique's effectiveness. On the contrary, Mustapha et al. (2016) highlighted the importance of rubrics to address inconsistencies in students' assignments in programming. They developed assessment rubrics for each of the domains of the Bloom's Taxonomy, the cognitive, psychomotor and affective domain. The rubrics were then applied by different instructors at a Turkish university. The analysis of their data indicates a consistency among the instructors, suggesting further that their rubrics were well-defined. The authors emphasise that showing the rubrics to students before the assessment helped them understand the level of effort that is needed to achieve the desired learning objectives.

Finally, Smith and Cordova (2005) explored the use of a primary trait analysis tool to assess students' assignments in computer programming at an American university. Primary traits analysis describes both the criteria upon which the evaluation occurs and the measures for each of the criteria. It is a rubric scale that describes directly to students what is anticipated from them and, also, what they must do to reach a specific level of performance (Baughin et al., 2002). Smith and Cordova extended this process by adding weights to each of these criteria or traits. The researchers created four categories: program correctness, programming style, program documentation and design documentation. They provided specific traits and achievement levels for each of these categories and developed a spreadsheet application that gives the opportunity for teachers to put weights on each criterion. They reported that the tool can be used successfully in computer science courses and provides a more reliable way of assessment.

Summary

This section described methods for developing effective assessment instruments for computing courses. The methods presented include Bloom's Taxonomy, SOLO Taxonomy, isomorphic and parameterised questions, rubrics and primary trait analysis. As in the previous section, the majority of the studies were conducted in higher education. Specifically, from the 22 research studies discussed in this section, only 3 were conducted in primary or secondary education

(Greece, high school; USA, primary; Israel, middle school) and the rest in universities in various countries (USA, Australia, New Zealand, Finland, Canada, Sweden, Turkey).

Bloom's Taxonomy is suggested as an effective method to create assessment instruments with different levels of difficulty. It is a valuable approach to create assessment questions that include all the six levels of the Taxonomy. This is an important factor that educators should consider to address the problem stated by Scott (2003) and Lister and Leaney (2003): most of an exam's questions target the average students, not challenging the stronger ones and leaving behind the weaker students. Bloom's Taxonomy can also be employed to improve learning objectives. Additionally, a study like the one of Chatzopoulou and Economides (2010) who proposed an adaptive system based on Bloom's Taxonomy should be further investigated. By employing such a tool, teachers can identify students' difficulties and the area in which these are accumulated. Students can assess and improve their learning as well. SOLO Taxonomy is also a good approach to categorise students' responses on an exam. The study of Meerbaum-Salant et al. (2010), which combined Bloom's and SOLO Taxonomies, is an interesting starting point for further research in this area.

The studies presented about isomorphic questions suggest that these questions can predict students' performance in an exam. If this is the case, then these questions can be a strong tool for teachers to identify weak students and the areas where they need help. Parameterised questions are more suitable for large classes and, thus, for universities. They can also be considered for creating online self-assessment instruments with a variation in the questions asked. Finally, research suggests that rubrics are necessary for a reliable assessment method and for communicating to students the criteria by which they are assessed.

Assessment tools

Automated assessment tools for programming

Searching the literature, one will come across a number of automated tools. Pears et al. (2005) distinguish four broad categories of automated tools in programming: a) visualisation tools, b) automated assessment tools, c) programming support tools, and d) microworlds. This section will focus only to automated tools for assessment and will provide a summary of the general

characteristics that these systems should demonstrate from pedagogical perspectives and the benefits for teachers and students of employing automated assessment tools.

Automated tools for assessment were initially created to help the educators handle a significant amount of marking but later incorporated functionalities that benefit students as well. Dreher et al. (2011) explain that the quality of assessment depends on the educators; their extensive range of work responsibilities and the pressure they feel can lead to inadequate assessment methods that limit the quality of feedback the students get. Consequently, automating the assessment process offers benefits for all the stakeholders.

Automated assessment tools have been used broadly for the assessment of computer programming. An extensive review of these tools was conducted by Ala-Mutka (2005). His survey identified two types of tools based on dynamic and static analysis. Dynamic analysis assesses a program by execution and it is used for evaluating functionality, efficiency and testing skills (Ala-Mutka, 2005 cited in Ihantola et al., 2010). Additionally, static checks analyse the program without executing it and provide feedback about style, programming errors, software metrics and even design (Ala-Mutka, 2005 cited in Ihantola et al., 2010). However, as Tiantian et al. (2009) argue, to design such systems, one must be very careful about the criteria under which the students' programs are to be assessed and the pedagogical reasons to employ them. The authors further point out that an automated assessment system for programming should demonstrate four basic functionalities:

- "Sufficient testing
- Checking whether the program meets the specification or not
- Dealing with programs with syntactic and semantic errors
- Immediate and corrective feedback" (Tiantian et al., 2009:221)

Tiantial et al.'s tool, Autolep, meets the above criteria and is based both on static and dynamic analysis to analyse students' programs. Tiantial et al. (2009) point out that the tool has been used at the Harbin Institute of Technology since 2004 with success; Autolep improves the quality of teaching, enforces students' interests in programming, and improves their skills and their competence in self-learning.

Vihavainen et al. (2013) also reported benefits of using their automated assessment tool, TMC (Test My Code), at a Finnish university. Their tool is focused on providing scaffolding opportunities: new goals are available as soon as the students have completed the previous

ones. TMC also supports bidirectional feedback (code reviews, students' and instructors' feedback), and collects data for analysis from students' programs. Generally, Vihavainen et al. (2013) pointed out that there are a number of advantages to be gained by employing this automated tool for assessment: scaffolding opportunities for students, reflecting opportunities for teachers and students and a better understanding of programming and improved learning experience. In the same vein, English and English (2015) evaluated the use of their tool (Checkpoint) and the effects on students' learning in a variety of computing courses (e.g. programming in C# and Java) at an Israeli University (sample size = 141). Checkpoint is an automated assessment tool that evaluates students' responses to open-ended questions, provides feedback and allows the resubmission of the assignment as soon as the errors are corrected. From the instructors' perspectives, an automated tool like Checkpoint can be very advantageous: it gives the opportunity for teachers to identify questions and places where their students make the most mistakes and the type of these mistakes. Thus, teachers can identify the tough topics in each course. Students also get feedback on their wrong answers which helps them understand their mistakes and correct them appropriately. The students' opinions of using this tool were very positive, and they highlighted the importance of the immediate feedback they received and the benefits of resubmission. Many of them also reported that they felt more confident in learning and more motivated as well.

Sant (2009) proposed a graphical email-client which supports automated assessment of programming tasks and discussed the advantages and disadvantages of employing such a system. From pedagogical perspectives, the strong characteristics of this approach are that it does not require any specific infrastructure, provides detail feedback and reduces the time that is needed for students to get their marked assignments.

Some tools also have been developed for school use. Bryce et al. (2013) introduced Bug Catcher, a system that creates testing competitions for high school students with no particular experience in programming. Their tool presents a program with bugs and the requirements and input fields that the students need to test. Even though the authors did not create this tool for assessment but for instilling interest in computer programming, it can potentially be used as a self-assessment tool for students. Finally, Burke and Kafai (2012) and Boe et al. (2013) proposed Scrape and Hairball, respectively, as assessment tools for Scratch projects. Specifically, Boe et al. (2013) argue that it is time-consuming to execute manually Scratch programs and this led them to propose an automated system, named Hairball, that both students and teachers can use to identify potentials errors and assess the students' projects. Hairball is a

static analysis tool which incorporates two roles: formative assessment, which gives students the opportunity to check their programs for errors, and summative assessment that assists teachers with the manual assessment of students' programs. Hairball can identify the existence of specific constructs and to evaluate their correctness.

In summary, Pieterse (2013:46) describes the success factors of automatic assessment for computer programming: "*quality assignments, clear formulation of tasks, well-chosen test data, good feedback, unlimited submissions, student testing maturity* and *additional support"*.

Assessment tools for computational thinking

Computational thinking is at the centre of the new national curriculum in the UK and has attracted huge interest worldwide in the last decade. According to Werner et al. (2012) though, any attempt to involve students in computational thinking is hindered by the deficiency of a definition and assessment tools. Selby et al. (2014) argue that to assess computational thinking, it is necessary to understand that computational concepts can be presented in multiple ways.

The difficulty in assessing computational thinking skills was discussed by Werner et al. (2012) who suggested an approach to assess computational thinking in middle school by game programming. The study included 325 students from the central California coast. Specifically, students were given a game with fairies in the Alice platform and they were asked to engage with it and make modifications when indicated by the game. The students were involved in three tasks and their performance was recorded. The authors concluded that an assessment environment like this is motivating and interesting for students and provides information about computational thinking skills and students' ability to apply them. Similarly, Denner et al. (2012) engaged middle school girls with game programming using Stagecast Creator, but they argued that the complexity of student programs was moderate.

Brennan and Resnick (2012) considered assessment in Scratch. In their article, they discuss three assessment techniques: project portfolio analysis, artefact-based interviews, and design scenarios and describe the advantages and weaknesses of each of these techniques. Portelance and Bers (2015) used a peer video interviewing technique as an additional assessment technique to the ones proposed by Brennan and Resnick (2012). They evaluated this technique with 62 students in a primary school who worked in pairs to develop their ScratchJr projects. The interviews were incorporated into the assessment process as an opportunity for students to present their work to their peers.

Another interesting approach has been suggested by Koh et al. (2014) who developed an automated tool (REACT) to help US teachers monitor, in real time, the computational thinking concepts that the students (grade 6) have grasped and the ones in which the students need further assistance. Their tool presents, in real time, the patterns of computational thinking that students employ and their correctness, and the patterns that students have not yet implemented. One of the advantages of this tool is that it offers formative assessment in real time which directly leads to fast and complete monitoring of students. Specifically, while the students create games and simulations, the system provides information to teachers about students' mastery of computational thinking concepts. The combination of formative and real-time assessment gives teachers the opportunity to identify students that need assistance and the tasks that the class has not yet understood. The researchers reported that teachers welcome this tool and that they intend to use it in their classrooms.

Seiter and Foreman (2013) proposed a model, "Progression of Early Computational Thinking Model" (PECT), to assess computational thinking of primary school students in USA. The model is based on the assumption that students' expertise in computational thinking is evidenced by students' skills in designing and implementing programs for defined tasks. For this reason, PECT is comprised of three components that set a framework for assessing computational thinking: "Computational Thinking Concepts, Design Pattern Variables, and Evidence Variables" (Seiter & Foreman, 2013:60). In their article, the authors explain in detail each of these categories. They pilot tested their framework to assess its effectiveness in identifying differences in computational thinking among students of different ages in primary school settings. The researchers concluded that their model is effective in monitoring students' progress of computational thinking.

Another study, conducted by Grover et al. (2014), describes a variety of assessment techniques employed by different schools. The goal of their study, which was part of the module "Foundations for Advancing Computational Thinking", was to explore different mechanisms for evaluating computational concepts in middle schools in the USA. They used formative assessment and incorporated it into multiple-choice quizzes based on feedback. The aim of this type of assessment was to familiarise students (n=52) with code tracing and code comprehension. The form of some of the questions was similar to Parson's puzzles (pg. 30) while others were open-ended and were manually assessed based on rubrics. To assess students' computational skills, the researchers also used questions from Israel's national exams which are based on Scratch and focus on decomposition, sequences, conditionals and loops as

the basis of algorithmic thinking. The researchers concluded that assessment in computer science courses needs various forms. Finally, Moreno-Leon and Robles (2015) presented Dr Scratch, a tool that assesses students' computational skills on Scratch projects and calculates a grade based on the score of each of the computational concepts.

Summary

This section has presented automated tools for assessing computer programming tasks and computational thinking. For assessing programming, seven studies were presented; three of them were conducted in middle or high schools in the USA and the rest in higher institutions in China, Finland, Canada and Israel. The tools presented offer many benefits to students and it is worthwhile to consider how these tools or new tools that will share the same pedagogical principles can be created for computing in UK schools. The research studies on computational thinking were conducted in primary or secondary education. Apart from Moreno-Leon and Robles (2015) who carried out their study in Spain, the rest were conducted in the USA. These tools can be applied to the UK computing curriculum which also is focused on computational thinking. However, more and large-scale research studies are needed in this area with a focus on how to effectively evaluate students' computational thinking skills.

Assessment Instruments

Concept maps

Concept maps were invented by Joseph Novak and his team in 1972. They were influenced by Ausubel's (1968) learning theory (Soika et al., 2012). Concept maps visually depict the relationship between concepts and ideas and their structure is based on these concepts and ideas that are linked to formulate comprehensive thoughts (Stoddart et al., 2000). Kinchin and Hay (2000) argue that a concept map is a useful metacognitive tool which can improve understanding and in which new information connects with the students' previous knowledge. The aim of a concept map is to reflect the perceptions, experience, and understanding of students and to demonstrate two characteristics of understanding: the representation and the organisation of knowledge (Kinchin and Hay, 2000).

Novak (1990) summarised the way that concept maps can be used for the advancement of learning and teaching. He argues that concepts maps can be used as a learning strategy, as an

instructional strategy, and as a strategy for planning the curriculum to assess students' understanding. Concept maps have been broadly used as a tool to assess students' understanding. Borda et al. (2009) argue that this is a unique assessment tool because it provides both an evaluation of the students' quality of understanding and it visually represents the structures of this understanding. Markham and Mintzes (1994) also suggest that there is a strong link between the quality and the structure of knowledge representation in concept maps. This indicates that students with complicated depictions have a deeper and profound understanding.

To evaluate a concept map, an educator must take into consideration the range of concepts depicted and the relationships they form. Equally important is the complexity and correctness of the illustrated information. Novak and Gowin (1984) proposed a detailed scoring system that instructors can easily apply to measure students' conceptual change.

In computer science courses, there are studies that have evaluated this assessment method. For example, Keppens and Hay (2008) provided a review of concept map techniques and assessed their suitability in computer programming. Their survey identified seven concept map assessment methods of which three were considered suitable for computer programming. The first is the closeness index and linkage analysis which is appropriate to assess students' understanding of basic concepts in computer programming. The second is the chain-poke-net differentiation which can be used for assessing students' knowledge of software libraries and, finally, the qualitative simulation-based approach for evaluating students' model structure skills.

Moen (2009) used concept maps to evaluate students' learning in database concepts. The research took place at the University of Helsinki and it was a small-scale study. The method employed involved students with the construction of a concept map on a weekly basis, followed up by a group discussion. The students also had the opportunity to compare their maps with the ones of their peers. By this experience, the students reported that concept maps were a useful tool to enhance and evaluate their learning and understood things that previously were unclear to them. From the teachers' perspective, concept maps enabled them to identify students' misunderstandings and evaluate how deep students' knowledge was, albeit being a very challenging and time-consuming process. The method may be best in classes of a small number. Moen (2009) concluded that concept maps are suitable tools both for learning and evaluating students' knowledge.

In a recent paper, Mühling (2016) evaluated the computing knowledge structures of students that had just graduated from high school and were entering university. In total, 338 students participated in the study which was conducted at the Technical University of Munich. The method that Mühling employed involves the analysis of students' concept maps to find mutual structures or differences in the students' knowledge. His study mostly centres on evaluating the structures of knowledge of a group of students rather than of an individual. Mühling (2016) claims this to be important information that educators should seek to collect to organise their courses. Two software tools, CoMapEd and CoMaTo, were used to collect, aggregate and analyse the students' concept maps. Analysis indicated that concept maps and the aggregation and analysis method that was used can identify the effects of the curriculum on students' organisation of knowledge. Mühling (2016) thus recommended that educators should consider employing concept maps to monitor students' progress.

Wei and Yue (2016) also evaluated students' learning in Information Systems at the University of Houston-Clear Lake by employing concept maps. In total, 124 students took part in this study. The researchers investigated the way that concept maps can be effectively used to evaluate students' learning by employing a tool for constructing and evaluating the maps (CmaoTools). The students' maps were both quantitatively and qualitatively analysed, and the results indicate that it is an effective tool to assess students' knowledge and improve their learning. Wei and Yue (2014) also concluded that concept maps are a valuable tool to assess students' knowledge, especially in Information Systems courses. These findings are in line with an earlier study conducted by Freeman and Urbaczewski (2001) who also evaluated this assessment method in students' understanding in Information Systems. They reported that the students found this assessment to be more entertaining and informative.

Code Comprehension

Code comprehension is also used in computer science courses as an assessment approach (Schulte et al., 2010). Sudol et al. (2012) argue that code comprehension tasks are effective indicators of students' abilities to produce and write code. In their study, the researchers explored the effectiveness of code comprehension problems in students' learning at an American university (sample size: 435). Particularly, the students were presented with a problem and they had to provide both open-ended responses and select an appropriate answer to a multiple-choice question. If their answer was wrong the system provided feedback and the students had to answer the same question again. Analysis of the students' data led the

researchers to conclude that code comprehension accompanied with feedback is an effective learning event.

An earlier study by Lopez et al. (2008) examined the correlation between students' verbal explanation of a program's functionality and their writing code ability. Their results indicate that there is a correlation between students' tracing ability and students' verbal explanations with students' ability to write programs. Lopez et al.'s (2008) findings mirror the ones of Murphy's et al. (2012) whose research also demonstrates a strong correlation between students' ability to verbally explain a program's functionality and students' ability to write code. The researchers also suggested that teaching programming should not only focus on coding but on other things as well, such as reading and explaining code. Finally, Simon and Snowdon (2011) assessed students' ability to explain what a code fragment does; they found that students' difficulties in articulating the meaning of the code is not due to their inefficacy to verbally explain it, rather it lies in their difficulty to really understand what the code does.

Even though these studies investigate approaches to teach programming, these suggestions have implications in assessment as well. For instance, if teaching programming should also focus on reading code, code tracing and code comprehension, as suggested by Murphy et al (2012), then appropriate tools and approaches for assessing these tasks should be developed and established.

Debugging

Other approach in assessing students' learning in programming make use of debugging tools. In fact, Fernandez-Medina et al. (2013) suggest an approach for assessing programming which is based on the mistakes that the students make in their programs. The study took place at a higher institution in Spain and focuses on exploring compiler messages and collecting information about a number of errors and warnings and their semantic importance. By this approach their tool, Colmena, can produce a progress report for all students. Colmena demonstrates both quantitative and qualitative feedback, and it is based on determining techniques and learning analytics. The information that is depicted by the tool gives the opportunity to instructors to understand the current learning state of their students and to students to track their learning progress.

Ahmadzadeh et al. (2005) also investigated the compiler errors with patterns of debugging tasks. The study was conducted at the University of Nottingham and 155 students participated.

The research showed that students who had a good understanding of programming did not necessarily have debugging skills, whereas students with debugging skills were also good programmers. Ahmadzadeh et al. (2005) contend that students' insufficient debugging skills inhibit their ability to write programs. The understanding of the program's implementation appears to be one of the main aspects that inhibit students' abilities to debug. These observations led the researchers to conclude that to increase students' abilities in programming, more emphasis is needed on debugging skills as well as on reading and understanding code. These suggestions mirror the ones of Murphy et al. (2012) and further enhance the need for generating assessment tools or instruments that will test students' abilities in reading and comprehending code.

Multiple-Choice questions, quizzes and other tasks

Multiple-choice questions are one of the most applied types of assessment tools. To help educators in computing to create effective multiple-choice instruments, Lister (2005) presented a multiple-choice exam for assessing undergraduate students' learning in programming. In his paper, he discusses criteria for assessing such questions. He created a twenty-six multiple-choice question exam to address three learning objectives covering object-oriented programming concepts, programming constructs and arrays. The researcher intended to start a community discussion about criteria to assess such exams and to suggest a repository with assessment questions in which students' responses should be stored as well for future analysis.

Kuechler and Simkin (2003:396) provide a list of advantages of employing multiple-choice questions in an exam: "machine scoring, volume grading, covering a wide range of course topics, more objective tools". Kuechler and Simkin (2003) also explored the correlation between multiple-choice questions and constructed response questions in computer programming in higher education. Even though their results do not suggest using multiple-choice questions exclusively for assessment, the authors argue that other factors, such as limited budget and large number of students, could justify the exclusive use of multiple-choice questions for assessment tasks. In a similar vein, Roberts (2006) welcomes this assessment for programming and describes a way of connecting multiple-choice questions with feedback and the benefits of this approach to learning. He argues that multiple-choice questions provide opportunities for learning and are objective instruments; thus, disagreement over marking hardly arises. Woodford and Bancroft (2005) also proposed guidelines for educators on creating effective assessment multiple-choice questions for Information Technology courses.

Their suggestions are very useful and they can be applied to other courses as well. The authors argue that multiple-choice questions can assess higher levels of knowledge and they give suggestions on how to construct items that belong to the Comprehension, Application and Analysis level of Bloom's Taxonomy.

Shuhidan et al. (2010) explored instructors' viewpoints of summative assessment with multiple-choice questions. Their findings indicate that instructors regard this assessment method as a valid method but they believe that multiple-choice questions are easy to answer and test low-level understanding. This contradicts Woodford and Bancroft's (2005) point who suggested a way to create multiple-choice items that test higher levels of thinking. Shuhidan et al. (2010) also used exam questions and applied four measures to them: syntax knowledge, semantic knowledge, problem-solving skills and level of difficulty, reporting that most of the instructors organise questions according to the topic of programming without applying any measures. The recommendation is that these four suggested measures should be applied by the instructors to better connect the questions with course objectives.

Denny et al. (2008) address the issue of code tracing, code writing and creating assessment problems through more creative processes such as Parson's puzzle problems. These problems are formed by giving fragments of code that the students must place in a logical order to form a solution. In their study, they found that these problems evaluate similar skills with code writing tasks but are considered easier, and more objective as well. Additionally, they are more suitable for identifying students' problems and misconceptions than code writing questions.

Currently, in the UK, a new assessment project is under development. Project Quantum[5] is a crowd-sourced question bank for school computing. The questions can be used by teachers to generate quizzes within an online quiz generation tool. In addition to providing question classes for primary and secondary computing, it intends to produce analysable data that will lead to improvements in the quality of teaching.

A variety of other assessment instruments have been used to assess students' performance in computing courses. For instance, Yin (2006) suggested the use of project plans in group assessment techniques, weekly and final group presentations, portfolio assessment, verbal progress reports and oral examinations in higher education settings. A more comprehensive investigation in the field of computing assessment in secondary education was conducted by

[5] http://projectquantum.org/

Yadav et al. (2015) in the USA. Specifically, they conducted interviews with teachers and identified that teachers use a variety of approaches which usually are focused on formative and summative assessment. Within the summative spectrum, teachers employ a number of techniques to measure students' learning. The investigators divided these techniques into multiple-choice assessment, open-ended assessment and rubrics to score assignments. Within the formative assessment spectrum, the teachers usually ask their students to write programs to measure students' understanding of concepts. They also use small quizzes at the beginning of the class to evaluate students' deeper level of learning. Some teachers also indicated that they assess students by "looking over students' shoulders". This means that the teacher is walking around the class and supervising the students as they work on an assignment. Response systems is another assessment technique used which gives teachers the opportunity to observe students' answers directly. By doing that, they can decide if students have difficulty in understanding a concept or if they can continue to a more difficult concept. Some teachers also reported using learning management systems because they help them control whether students achieve the learning goals. Yadav et al. (2015) concluded that teachers use a variety of tools. They use tools for formative assessment when they are interested in accumulating information about whether their students are reaching the learning goals and summative assessment when they seek information on students' learning.

Summary

This section has presented assessment instruments and tasks for assessing computing. These include concepts maps, code comprehension tasks, debugging tasks, multiple-choice questions, Parson's puzzles and the Quantum Project. Most of the studies were conducted in higher education (USA, Germany, Finland, New-Zealand, Australia, Spain, UK) and two of them were conducted in primary and secondary education (Quantum project and Yadav et al.'s research).

The research studies presented in this section lead to the conclusion that concept maps are an appropriate tool to illustrate students' structure and depth of knowledge. This can help educators identify gaps in students' learning and misconceptions. Concept maps have been employed in primary and secondary education in other disciplines with success and, thus, their use in computing courses seems appropriate. Based on and extending upon Moen's (2009) work, it could be a good approach to combine concept maps with self and peer assessment. The

advantages stemming from both these methods could be of a great value for both the teachers and the students.

Code tracing, comprehension and debugging tasks are known to be good indicators of students' abilities to write code. As already mentioned in this section, the fact that teaching programming should include these tasks highlights the need to develop effective assessment instruments that will test students' abilities in tracing and comprehension before students progress to the process of writing code. By employing such assessment tasks, teachers can better determine the time that this transition (from code tracing and comprehension to writing code) should take place.

Key findings

From the literature review presented, there are many different assessment techniques employed in computing courses. In summary, these include: a) peer assessment and self-assessment b) Bloom's and SOLO Taxonomies c) isomorphic and parameterised questions d) rubrics e) automated tools for assessing programming f) tools for assessing computational thinking g) concept maps h) code comprehension tasks i) debugging tasks k) multiple-choice questions and quizzes. Table 1 depicts each of the approaches and tools described above and gives a summary of the advantages to the computing assessment.

Table 1 Advantages of the assessment methods presented

Approach	**Advantages**
Self-assessment	• Encourages independent learning (Gayo-Avello and Fernádez-Cuervo, 2003) • Motivates learners to take control of their learning (García-Beltrán and Martinez, 2006) • Increases self-confidence and improves learning (García-Beltrán and Martinez, 2006) • Calibration leads to self-regulation and metacognition (Murphy and Tenenberg, 2005)
Peer-assessment	• Communicates aims and criteria of assessment (Topping et al., 2000)

	• Students work together and create a community of learning based on collaboration (Clark, 2004) • Students recognise their errors by assessing their peers' assignments (Sitthiworachart and Joy, 2003; Clark, 2004; Smith et al., 2012; Tseng and Tsai, 2007) • Increases quality of assignments (Sitthiworachart and Joy, 2003; Clark, 2004; Smith et al., 2012; Tseng and Tsai, 2007) • Enhances metacognition and higher and critical thinking skills (Liu et al., 2001) • Students recognise their weaknesses and strengths (Liu et al., 2001)
Bloom's & SOLO taxonomy	• Identify and improve learning objectives before assessment (Starr et al., 2008) • Effective way to design exam questions and assessment processes in programming (Thompson et al., 2008) • Engage students with different types of questions, different level of difficulty, and include questions that assess different skills (Lister and Leaney, 2003; Scott, 2003) • Help teachers understand students' levels of knowledge (Alaoutinen and Smolander, 2010) • Identify students' levels of understanding of the problem's structure (Seiter, 2015; Lister et al., 2006; Shuhidan et al., 2009)
Isomorphic Questions	• Teachers can monitor students' learning and changes in their knowledge (Porter et al., 2011; Zingaro and Porter, 2014) • Can predict students' performance (Porter et al., 2011; Zingaro and Porter, 2014; Porter et al., 2014) • Can evaluate the effect of a teaching approach (Zingaro and Porter, 2015)
Parameterised Questions	• Help teachers produce questions for large classes (Sosnovsky et al., 2003)

	• Students can self-assess themselves (Brusilovsky and Sosnovsky, 2005) • Assist students with their learning (Pathak and Brusilovsky, 2002)
Rubrics	• Reliable and valid assessment (Becker, 2003; Mustapha et al., 2016; Smith and Cordova, 2005) • Students understand teachers' expectations (Becker, 2003; Barney et al., 2012) • Students understand the level of effort needed (Mustapha et al., 2016)
Automated tools for programming	• Provide immediate feedback (Tiantian et al., 2009; Vihavainen et al., 2013; English and English, 2015; Sant, 2009) • Provide environments that motivate students to practice programming (Tiantian et al, 2009; English and English, 2015) • Self-assessment opportunities (Bryce et al., 2013) • Improve self-confidence and competence (Tiantian et al., 2009; English and English, 2015) • Improve quality of teaching (Tiantian et al., 2009) • Teachers can identify places in the curriculum that the students face difficulties (English and English, 2015) • Provide scaffolding opportunities (Vihavainen et al., 2013) • Improve learning experience (Vihavainen et al., 2013)
Assessment tools for computational thinking	• Assessment based on gaming provides motivating and interesting learning environments (Werner et al., 2012) • Provide information to teachers about students' mastery of computational thinking concepts (Werner et al., 2012; Seiter and Foreman, 2013; Koh et al., 2014; Grover et al., 2014) • Can present in real time the patterns of computational thinking that students have employed (Koh et al., 2014) • Assessment in real time directs fast and complete monitoring of students (Koh et al., 2014)

	• Can track students' progress in computational thinking (Seiter and Foreman, 2013)
Concept maps	• Identify students' misconceptions (Moen, 2009) • Enhance and evaluate students' learning (Moen, 2009; Wei and Yue, 2016) • Depict students' organisation of knowledge (Mucling, 2016) • Depict students' levels of understanding (Moen, 2009) • Teachers can monitor students' learning (Mucling, 2016; Wei and Yue, 2016; Freeman and Urbaczewski, 2001) • Students feel that they are informative and more entertaining that other methods (Freeman and Urbaczewski, 2001)
Code comprehension	• Effective learning events (Sudol-DeLyser et al., 2012) • Strong correlation between students' abilities to explain what a program does with students' abilities to write programs (Lopez et al., 2008; Murphy et al., 2012)
Debugging tasks	• Depict students' errors and mistakes (Fernadez-Medina et al., 2013) • Teachers can understand and monitor the current learning state of their students (Fernadez-Medina et al., 2013; Ahmadzadeh et al., 2005) • Students can track their learning progress (Fernadez-Medina et al., 2013)
Multiple-choice questions and quizzes	• Assessment is more objective (Roberts, 2006; Kuechler and Simkin, 2003) • Create opportunities for learning (Roberts, 2006) • Parson's puzzles evaluate the same skill as writing code but they are easier tasks (Denny et al., 2008) • Parson's puzzles can identify students' misconceptions and problems in programming (Denny et al., 2008)

	• Can assess higher levels of knowledge (Woodford and Bancroft, 2005)

Research findings and computing assessment in UK schools

From this literature review, it is apparent that a large number of studies have been conducted in higher education institutions. However, the approaches and methods of assessment used may apply to UK schools with some modifications.

Peer and self-assessment approaches could readily be applied to computer science courses in school. In school, peer assessment is more likely to be paper-based and not anonymous, but is a popular strategy for teachers to use for assessment for learning. It may be that more automated approaches to peer assessment in school might be a useful area for investigation. There are some issues emerging from the literature that would need to be addressed for primary or secondary education settings. Concerns have been raised about the quality and bias of the feedback that students exchange. For instance, Vickerman (2009) points out that students have a tendency to give higher grades especially if the students know each other. Additionally, Patton (2012) argues that some students do not agree with being assessed by their peers because they believe that they lack the expertise needed in the evaluation process. This subsequently may lead students to disregard this type of assessment and generally regard it as an unfair method (Foley, 2013).

Some researchers have tried to address these issues in computing courses. For example, the study of Sitthiworachart and Joy (2003), described earlier, reported that the students' evaluation was highly correlated with the one of the teachers. Hamer et al. (2009) have evaluated peer assessment in computer programming but their interests focused on the quality of students' reviews. Their findings signify a good correlation between students' and tutors' marks and the quality of comments which is in agreement with Sitthiworachart and Joy's (2003) findings. But again, these research studies were conducted in undergraduate settings. More research is needed to determine if the validity issues raised in higher education settings hold in primary or secondary settings.

Teachers in schools tend to use multiple-choice questions, quizzes and other instruments. The literature suggests considering Bloom's and SOLO Taxonomy and the use of rubrics, isomorphic and parameterised questions in the design of these instruments. Bloom's Taxonomy can be used to create and categorise exam questions and the SOLO Taxonomy to measure students' understanding based on their responses. However, it should be noted that the application of Bloom's Taxonomy for computer programming is not an easy task which is demonstrated by the disagreement among experts in classifying questions according to the corresponding levels (Fuller et al., 2007). It would also be good for teachers to consider isomorphic and parameterised questions. Isomorphic questions can be used to monitor changes in students' learning and predict students' performance while parameterised questions can help teachers generate questions for large classes. Additionally, rubrics seem essential for a valid and fair assessment and to communicate to students the criteria by which they will be evaluated.

Regarding automated tools for assessment, there are many benefits for using them in computing courses. Computer programming is a complex task and students need immediate and quality feedback for their programming assignments. Assessing students' progress with automated tools could help both teachers to improve their effectiveness and students to improve their learning. However, automated assessment tools for educational purposes should be carefully designed using good pedagogical practices. Unfortunately, studies about automated tools for text-based languages in secondary education settings were not found during this review. Most of the existing tools are developed for undergraduate students and their benefits are considerable, as discussed earlier in this review. Pieterse (2013) proposed what an instructor should look for when employing an automated tool for programming. These are good suggestions for any level of education.

Tools for assessing students' computational thinking in primary and secondary education were presented in this review. These are directly applicable to UK schools since these studies were also conducted in school settings. However, more research, large-scale and robust, is needed to evaluate the effectiveness of these tools and methods on students' learning and whether they actually measure students' computational thinking skills.

Multiple-choice questions may be a strong tool for assessment if they are designed properly. It seems that there is a controversy between educators and researchers upon the use of this tool; Shuhidan et al. (2010) highlighted that educators think that multiple-choice questions only test lower levels of thinking while Woodford and Bancroft (2005) and Roberts (2006) give

suggestions on how to create multiple-choice items that can evaluate higher levels of thinking. The use of concept maps is suggested to monitor students' structures and depths of understanding. This tool can be used without difficulty by primary and secondary teachers to monitor students' progress. Code tracing, comprehension and the verbal explanation of the functionality of code are good practices for teaching programming as well as assessing students' learning. They can directly be applied in school settings and would be useful for teachers to understand students' abilities in writing computer programs.

Clear priorities for future research

Assessment has a critical role in the education process. For an assessment method to be effective, it is necessary first to identify the criteria or the standards that will be used for evaluation. Having considered that, the educator should proceed by choosing the assessment technique that is best for adressing these criteria (Bull & McKenna, 2003). There is a number of methods of assessment which can be employed with different mechanisms such as self and peer assessment and automated tools. Firstly, a teacher should choose the method of assessment and then the appropriate mechanism to conduct the assessment (Bull & McKenna, 2003).

Without rigorous assessment, computational thinking cannot be successfully embedded in education (Grover et al., 2014). Ragonis (2012) also argues that if the quality of assessment is insufficient, the teaching will be ineffective. However, assessing computer science courses is non-trivial. Brennan and Resnick (2012) emphasise the difficulties in evaluating computational thinking and underline that computer courses need a variety of assessment means. The diversity of assessment tasks is important to include all students but should be explicit as well to deliver appropriate feedback. As Yin (2006) highlights, assessment should include:

- "Multiple participants that include the students, their peers and their teachers
- Multiple units of assessment that include assessment of individuals, groups, the whole class
- Multiple forms that include written work, observations, presentations, projects, etc" Yin (2006:43)

There is broad agreement among researchers that a multi-faceted approach to assessment in schools is necessary. Ragonis (2012) gives three reasons for this: a) different type of questions highlight different learning aspects b) different type of questions require students to employ a different type of skills and c) different type of questions enable teachers to offer a variety of tools for assessment and, thus, making it more interesting to students. Grover et al. (2014) point out that assessment in computer science courses should consider both students' motivation and creativity and also include objective criteria for assessment that can evaluate both students' understanding of computational concepts and students' skills, such as debugging and code-tracing. They further envisage a multi-nation collaboration for building collective knowledge to generate "computationally literate learners" (Grover et al, 2014:62). Supporting this, Yadav et al. (2015) recommend the development of valid and reliable assessment tasks and also the development of an online source that would enable teachers to access and share these tasks.

It follows that the next research directions in this area should focus on:

- The development of measures and rubrics for assessment in computing courses and their evaluation on identifying students' levels of learning and students' misconceptions.
- Investigation of effective assessment approaches in computing courses. Self and peer assessment can be used in primary and secondary computing but it is necessary first to explore the validity issues that were mentioned earlier in this review.
- The development of high quality and reliable assessment instruments and tasks that are in alignment with the learning objectives of the new computing curriculum. Attention should be given to concept maps which are widely used in primary and secondary education in other fields and to code tracing and comprehension tasks.
- The development and evaluation of automated assessment tools for text-based programming or the investigation of how the ones already used in higher education can be adapted for the purposes of secondary education.
- The development of both formative and summative assessment tasks for computer programming languages and computational thinking or a deeper evaluation of the ones described in this review on students' learning.
- The development of an online shared repository with a variety of assessment questions and tasks.

- The development of supportive mechanisms that would educate and develop teachers' knowledge on assessment practices.
- Rigorous and large-scale empirical research studies.

Taking everything into account, assessment in computing should come in different forms that will help students develop their knowledge and understanding and obtain a range of cognitive skills. It must be realised that assessment is an opportunity for learning and, thus, should provide students with the necessary information that would improve their performance. As such, it should include a variety of means that offer information on the outcome of the education process and should be an appropriate measurement of the curriculum's objectives. Giordano et al. (2015:120) argue that assessment should be "evidence-based clearly linked to a competency framework".

Teachers' involvement is considered necessary in this endeavour. Research in the UK should initially focus on a study similar to the one conducted by Yadav et al. (2015) in the USA; this research will investigate teachers' current assessment practices and the problems that they face employing assessment techniques and tools to cover the new learning objectives of the curriculum. As Yadav et al. (2015) point out, teachers are often leading best practices on every aspect of teaching and learning and, thus, it would be an excellent starting point to collect their experience towards the design and implementation of assessment tasks that would be appropriate for computing education in schools.

References

Ahmadzadeh, M., Elliman, D. and Higgins, C., (2005) An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, 37(3), 84-88

Ala-Mutka, K., (2005) A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2), 83–102.

Alaoutinen, S., Smolander, K., (2010) Student self-assessment in a programming course using bloom's revised taxonomy. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 155-159. ACM

Anewalt, K., (2005) Using peer review as a vehicle for communication skill development an active learning. *Journal of Computing Sciences in Colleges.*, 21(2),148–155

Anderson, L.W., Krathwohl, D., (Eds.) (2001) *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman.

Andrade, H.G., (2000) Using rubrics to promote thinking and learning. *Educational leadership*, 57(5), 13-19.

Barney, S., Khurum, M., Petersen, K., Unterkalmsteiner, M. and Jabangwe, R., (2012) Supporting Students Improve with Rubric-Based Self-Assessment and Oral Feedback. *IEEE Transactions on Education*, 55(3), 319-325

Baughin, J.A., Brod, E.F., and Page, D.L., (2002) Primary trait analysis: A tool for classroom-based assessment. *College Teaching*, 50(2), 75-80.

Becker, K., (2003) Grading programming assignments using rubrics. In *ACM SIGCSE Bulletin* 35(3), 253-253. ACM.

Biggs, J.B., Collis, K.F., (1982) *Evaluating the quality of learning: the SOLO taxonomy.* New York: Academic Press

Biggs, J., (1999) What the student does: Teaching for enhanced learning. *Higher education research & development*, 18(1), 57-75

Black, P., Wiliam, D., (2009) Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability*, 21(1), 5-31

Bloom, B., Englehart, M. Furst, E., Hill, W., and Krathwohl, D. (1956) *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. New York, Toronto: Longmans, Green

Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P. and Franklin, D., (2013) Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education,* March, 215-220. ACM.

Borda, E.J., Burgess, D.J., Plog, C.J., DeKalb, N.C. and Luce, M.M., (2009) Concept maps as tools for assessing students' epistemologies of science. *Electronic Journal of Science Education*, 13(2), 160-185

Boud, D., (1995) *Enhancing Learning through Self-Assessment*. London: Kogan Page

Boud, D., Falchikov, N., (2007) *Rethinking assessment in higher education.* London: Kogan Page

Brennan, K., Resnick, M., (2012) New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association,* 1-25. Vancouver, Canada.

Brusilovsky, P., Sosnovsky, S., (2005) Individualized Exercises for Self-Assessment of Programming Knowledge: An Evaluation of QuizPACK, *ACM Journal on Educational Resources in Computing*, 5(3).

Bryce, R., Mayo, Q., Andrews, A., Bokser, D., Burton, M., Day, C., Gonzolez, J. and Noble, T., (2013). Bug catcher: a system for software testing competitions. In *Proceeding of the 44th ACM technical symposium on Computer science education,* March, 513-518. ACM.

Bull, J., McKenna, C., (2003) A Blueprint for Computer -Assisted Assessment. Routledge.

Burke, Q., Kafai, Y.B., (2012) The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 433–438. ACM

Burnett, P.C., (1999) Assessing the structure of learning outcomes from counselling using the SOLO taxonomy: An exploratory study. *British Journal of Guidance and Counselling*, 27(4), 567-580.

Chatzopoulou, D.I., Economides, A.A., (2010) Adaptive assessment of student's knowledge in programming courses. *Journal of Computer Assisted Learning*, 26(4), 258-269

Chi, M.T., (1996) Constructing self-explanations and scaffolded explanations in tutoring. *Applied Cognitive Psychology*, 10(7), 33-49

Chin, D., (2005) Peer assessment in the Algorithms course. *ITiCSE*, June 27-29, 69-73, Monte de Caparica, Portugal

Clark, N., (2004) Peer testing in software engineering projects. In *Proceedings of the Sixth Australasian Conference on Computing Education*, 30:, 41-48. Australian Computer Society, Inc.

Clark, N., (2012) Peer group testing in software engineering projects. In *Proceedings of The Australian Conference on Science and Mathematics Education (formerly UniServe Science Conference)*, November, 9:, 14-19

Davies, R., Berrow, T., (1998) An evaluation of the use of computer peer review for developing higher-level skills. *Computers in Education*, 30(1), 111-115.

Denner, J., Werner, L. and Ortiz, E., (2012) Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58(1), pp.240-249.

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

Dreher, C., Reiners, T., and Dreher, H., (2011) Investigating factors affecting the uptake of automated assessment technology. *Journal of Information Technology Education*, 10:, 161-181

English, J., English, T., (2015) Experiences of using automated assessment in computer science courses. *Journal of Information Technology Education: Innovations in Practice*, 14:, 237-254

Fernandez-Medina, C., Pérez-Pérez, J.R., Álvarez-García, V.M. and Paule-Ruiz, M., (2013) Assistance in computer programming learning using educational data mining and learning analytics. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, July, 237-242. ACM.

Foley, S., (2013) Student views of peer assessment at the International School of Lausanne. *Journal of Research in International Education,* 12(3), 201-213

Freeman, L.A., Urbaczewski, A., (2001) Using concept maps to assess students' understanding of Information Systems. *Journal of Information Systems Education,* 12(1), 3-9

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

García-Beltrán, A., Martinez, R., (2006) Web assisted self-assessment in computer programming learning using AulaWeb. *International Journal of Engineering Education,* 22(5), 1063-1069.

Gayo-Avello, D. and Fernández-Cuervo, H., (2003) Online self-assessment as a learning method. In *Proceedings of the 3rd International Conference on Advanced Learning Technologies,* 254-255. IEEE.

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

Grover, S., Cooper, S. and Pea, R., (2014) Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education,* 57-62. ACM

Hamer, J., Purchase, H.C., Denny, P. and Luxton-Reilly, A., (2009) Quality of peer assessment in CS1. In *Proceedings of the fifth international workshop on Computing education research workshop,* August, 27-36. ACM

Hamm, R. W., Henderson, K.D., Repsher, M.L. and Timmer, K.L., (1983) A tool for program grading: The Jacksonville University scale. In *Proceedings of the 14th SIGCSE Technical Symposium on Computer Science Education*, 248-252. ACM

Hanrahan, S.J., Isaacs, G., (2001). Assessing self-and peer-assessment: The students' views. *Higher Education Research & Development*, 20(1), 53-70.

Hattie, J., Brown, G.T., (2004) *Cognitive processes in asTTle: The SOLO taxonomy*. Ministry of Education.

Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O., (2010) Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, 86-93. ACM.

Izu, C., Weerasinghe, A. and Pope, C., (2016) A Study of Code Design Skills in Novice Programmers using the SOLO taxonomy. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, August, 251-259. ACM.

Kay, J., Li, L. and Fekete, A., (2007) Learner reflection in student self-assessment. In *Proceedings of the ninth Australasian conference on Computing education*, 66:, 89-95. Australian Computer Society, Inc

Keppens, J., Hay, D. (2008) Concept map assessment for teaching computer programming, *Computer Science Education*, 18(1), 31-42

Kinchin, I.M., Hay, D.B. and Adams, A., (2000) How a qualitative approach to concept map analysis can be used to aid learning by illustrating patterns of conceptual development. *Educational Research*, 42(1), 43-57

Koh, K.H., Basawapatna, A., Nickerson, H. and Repenning, A., (2014) Real time assessment of computational thinking. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium*, July, 49-52. IEEE.

Krathwohl, D., (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice*, 41(4), 212-218.

Kuechler, W.L., Simkin. M.G., (2003) How Well Do Multiple Choice Tests Evaluate Student Understanding in Computer Programming Classes? *Journal of Information Systems Education*, 14(4), 389-399

Lamprianou, I., Athanasou, J., (2009) *A Teacher's Guide to Educational Assessment*. Sense Publishers: Rotterdam

Lin, S.S.J., Liu, E.Z.F., and Yuan, S.M., (2001) Web-based peer assessment: feedback for students with various thinking-styles. *Journal of Computer Assisted Learning*, 17:, 420–432

Lister, R., Leaney, J., (2003) Introductory programming, criterion-referencing, and bloom. *ACM SIGCSE Bulletin*, 35(1), 143-147.

Lister, R., (2005) One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple-choice exam for first semester programming students. *Seventh Australasian Computing Education Conference* (ACE2005), 155-164

Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C., (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education*, Bologna, Italy, June 26 – 28, 118-122.

Liu, J., Pysarchik, D.T, and Taylor, W.W., (2002) Peer review in the classroom. *Bioscience*. 52(9), 824–829

Liu, E.Z.F., Lin, S.S., Chiu, C.H. and Yuan, S.M., (2001) Web-based peer review: the learner as both adapter and reviewer. *IEEE Transactions on education*, *44*(3), 246-251

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

Markham, K., Mintzes, J.J., (1994) The concept map as a research and evaluation tool: Further evidence of validity. *Journal of Research in Science Teaching*, 31(1), 91-101

McMillan, J.H., Hearn, J., (2008) Student self-assessment: The key to stronger student motivation and higher achievement. *Educational Horizons*, 87(1), 40–49.

Miller, N.E., Peterson, C.G., (1980) A method for evaluating student written computer programs in an undergraduate computer science programming language course. *SIGCSE Bulletin*, 12(4), 9-17

Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M., (2013) Learning computer science concepts with scratch. In *ICER '10: Workshop on Computing education research,* 69–76.

Moen, P., (2009) Concept Maps as a Device for Learning Database Concepts. In *7th Workshop on Teaching, Learning and Assessment in Databases,* Birmingham, UK, HEA

Moreno-León, J., Robles, G., (2015) Computer programming as an educational tool in the English classroom a preliminary study. In *Global Engineering Education Conference (EDUCON),* 961-966. IEEE

Mühling, A., (2016) Aggregating concept map data to investigate the knowledge of beginning CS students. *Computer Science Education,* 26:, 176-191

Murphy, L., Tenenberg, J., (2005) Knowing what I know: an investigation of undergraduate knowledge and self-knowledge of Data Structures. *Computer Science Education*, 15(4), 297-315

Murphy, L., Fitzgerald, S., Lister, R. and McCauley, R., (2012) Ability to explain in plain English linked to proficiency in computer-based programming. In *Proceedings of the ninth annual international conference on International computing education research,* September, 111-118. ACM.

Mustapha, A., Samsudin, N.A., Arbaiy, N., Mohammed, R. and Hamid, I.R., (2016) Generic Assessment Rubrics for Computer Programming Courses. *Turkish Online Journal of Educational Technology-TOJET*, 15(1), 53-68.

Novak, J.D., Gowin, D.B., (1984) *Learning how to learn*. Cambridge, England

Novak, J.D., (1990) Concept maps and vee diagrams: Two metacognitive tools for science and mathematics education. *Instructional Science,* 19:, 29-52.

Nulty, D.D., (2011) Peer and self-assessment in the first year of university. *Assessment & Evaluation in Higher Education*, 36(5), 493-507

Patton, C., (2012) Some kind of weird, evil experiment: student perceptions of peer assessment. *Assessment & Evaluation in Higher Education,* 37(6), 719-731.

Pathak, S., Brusilovsky, P., (2002) Assessing student programming knowledge with web-based dynamic parameterized quizzes. In *Proceedings of ED-MEDIA,* June, 24-29

Pears, A., Seidman, S., Eney, C., Kinnunen, P. and Malmi, L., (2005) Constructing a core literature for computing education research. *ACM SIGCSE Bulletin*, 37(4), 152-161

Pieterse, V., (2013) Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research,* 45-56. Open Universiteit, Heerlen

Popham, W.J., (1997) What's wrong-and what's right-with rubrics. *Educational leadership*, 55:, 72-75

Portelance, D.J., Bers, M.U., (2015) Code and Tell: Assessing young children's learning of computational thinking using peer video interviews with ScratchJr. In *Proceedings of the 14th International Conference on Interaction Design and Children*, 271-274. ACM.

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

Porter, L., Zingaro, D. and Lister, R., (2014) Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*, July, 51-58. ACM

Ragonis, N., (2012) Type of questions–the case of computer science. *Olympiads in Informatics*, 6:, 115-132

Roberts, T., (2006) The use of multiple choice tests for formative and summative assessment. *Eighth Australasian Computing Education Conference* (ACE2006), 175-180

Sajjadi, M.S., Alamgir, M. and von Luxburg, U., (2016) Peer Grading in a Course on Algorithms and Data Structures: Machine Learning Algorithms do not Improve over Simple Baselines. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, 369-378. ACM

Sant., J.A., (2009) Mailing it in: email-centric automated assessment. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conference. on Innovation and technology in computer science education*, 308–312, New York, NY, USA. ACM

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

Scott, T. (2003) Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing in Small Colleges*, 19(1), 267-274.

Searby, M., Ewers, T., (1997) An evaluation of the use of peer assessment in higher education: A case study in the School of Music, Kingston University. *Assessment & Evaluation in Higher Education*, 22(4), 371-383.

Seiter, L., Foreman, B., (2013) Modelling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, August, 59-66. ACM.

Seiter, L., (2015) Using solo to classify the programming responses of primary grade students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, February, 540-545. ACM

Selby, C., Dorling, M. and Woollard, J., (2014) Evidence of assessing computational thinking. *Author's original*, 1-11.

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E. and Whalley, J.L., (2008) Going SOLO to assess novice programmers. In *ACM SIGCSE Bulletin*, 40(3), 209-213. ACM

Sheldrake, R., Mujtaba, T. and Reiss, M.J., (2014) Calibration of self-evaluations of mathematical ability for students in England aged 13 and 15, and their intentions to study non-compulsory mathematics after age 16. *International Journal of Educational Research*, 64:, 49-61.

Shuhidan, S., Hamilton, M., and D'Souza, D., (2009) A taxonomic study of novice programming summative assessment. In *Proceedings of the Eleventh Australasian Conference on Computing Education*, 95, 147-156. Australian Computer Society, Inc

Shuhidan, S., Hamilton, M. and D'Souza, D., (2010) Instructor perspectives of multiple-choice questions in summative assessment for novice programmers. *Computer Science Education*, 20(3), 229-259

Sitthiworachart, J., Joy, M., (2003) Web-based peer assessment in learning computer programming. In *Proceedings of Advanced Learning Technologies*, July, 180-184. IEEE

Smith, L., Cordova, J., (2005) Weighted primary trait analysis for computer program evaluation. *Journal of Computing Sciences in Colleges*, 20(6), 14-19

Smith, M.K., Wood, W.B., Adams, W.K., Wieman, C., Knight, J.K., Guild, N., and Su, T.T., (2009). Why peer discussion improves student performance on in-class concept questions. *Science*, 323:, 122–124.

Smith, J., Tessler, J., Kramer, E. and Lin, C., (2012) Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research*, September, 93-98. ACM.

Simon, Snowdon, S., (2011) Explaining program code: giving students the answer helps-but only just. In *Proceedings of the seventh international workshop on Computing education research*, August, 93-100.

Soika, K., Reiska, P. and Mikse, R., (2012) Concept Mapping as an Assessment Tool in Science Education. In *Concept Maps: Theory, Methodology, Technology. Proceedings of the Fifth International Conference on Concept Mapping.* Available at http://cmc.ihmc.us/cmc2012papers/cmc2012-p188.pdf (accessed at 20/02/2017)

Somervell, H., (1993) Issues in assessment, enterprise and higher education: the case for self-peer and collaborative assessment. Assessment and Evaluation in Higher Education, 18:, 221-233

Sosnovsky, S., Shcherbinina, O., and Brusilovsky, P., (2003) Web-based parameterized questions as a tool for learning. In *Proceedings of World Conference on E-Learning, E-Learn*, November, 7-11

Spiller, D., (2012) Assessment matters: Self-assessment and peer assessment. The University of Waikato. Available at http://www.waikato.ac.nz/tdu/pdf/booklets/8_SelfPeerAssessment.pdf. (accessed at 18/02/2017)

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

Stoddart, T., Abrams, R., Gasper, E. and Canaday, D., (2000) Concept maps as assessment in science inquiry learning-a report of methodology. *International Journal of Science Education*, *22*(12), 1221-1246.

Sudol, L.A., Stehlik, M., and Carver, S., (2012) Code comprehension problems as learning events. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 81-86.

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

Tiantian, W., Xiaohong, S., Peijun, M., Yuying, W. and Kuanquan, W., (2009) Autolep: An automated learning and examination system for programming and its application in programming course. In *First International Workshop on Education Technology and Computer Science*, 43-46. IEEE.

Topping, K.J., Smith, E.F., Swanson, I., and Elliot, A., (2000) Formative peer assessment of academic writing between postgraduate students. *Assessment and Evaluation in Higher Education*, 25(2) 150–169.

Topping, K.J., (2009) Peer Assessment. *Theory Into Practice*, 48(1), 20-27,

Tseng, S.C., Tsai, C.C. (2007) On-line peer assessment and the role of the peer feedback: A study of high school computer course. *Computer & Education*, 49:, 1161-1174

Vickerman, P., (2009). 'Student perspectives on formative peer assessment: an attempt to deepen learning?', *Assessment & Evaluation in Higher Education*, 34(2), 221-230.

Vihavainen, A., Vikberg, T., Luukkainen, M., and Pärtel, M., (2013) Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 117-122. ACM

Wei, W., Yue, K.B., (2016) Using Concept Maps to Assess Students' Meaningful Learning in IS Curriculum. In *Conference on Information Systems and Computing Education EDSIG*, Las Vegas.

Werner, L., Denner, J., Campe, S., and Kawamoto, D.C., (2012) The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 215-220. New York: ACM.

Wolfe, W.J., (2004) Online student peer reviews. In *Proceedings of the 5th Conference on Information Technology Education*, 33-37, ACM.

Woodford, K., Bancroft, P., (2005) Multiple choice questions not considered harmful. *Seventh Australasian Computing Education Conference* (ACE2005), 109-115

Wohlin, C., (2014) Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, May, 38-48. ACM.

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

Yin, F., (2006) Applying methods of formative and summative assessment to problem-based learning in computer courses. *The China Papers*, 42-45. Available at http://science.uniserve.edu.au/pubs/china/vol6/IT2.pdf (Accessed at 20/02/2017)

Zingaro, D., Porter, L., (2013) Peer Instruction in computing: The value of instructor intervention. *Computers & Education*, 71, 87-96.

Zingaro, D. and Porter, L., (2014) Peer instruction: a link to the exam. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 255-260. ACM

Zingaro, D. and Porter, L., (2015) Tracking Student Learning from Class to Exam using Isomorphic Questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 356-361.

Appendix 1: Research studies per theme

Theme	Topic	Articles included
1. Assessment approaches	**Self-assessment**	• García-Beltrán, A., Martinez, R., (2006). Web assisted self-assessment in computer programming learning using AulaWeb. *International Journal of Engineering Education*, 22(5), 1063-1069. • Gayo-Avello, D. and Fernández-Cuervo, H., (2003). Online self-assessment as a learning method. In *Proceedings of the 3rd International Conference on Advanced Learning Technologies*, 254-255. IEEE. • Murphy, L., Tenenberg, J. (2005) Knowing what I know: an investigation of undergraduate knowledge and self-knowledge of Data Structures," *Computer Science Education*, 15(4), 297-315
	Peer-assessment	• Sitthiworachart, J. and Joy, M., (2003) Web-based peer assessment in learning computer programming. In *Proceedings of Advanced Learning Technologies*, 180-184. IEEE • Clark, N., (2004). Peer testing in software engineering projects. In *Proceedings of the Sixth Australasian Conference on Computing Education*, 30:, 41-48. Australian Computer Society, Inc.. • Smith, J., Tessler, J., Kramer, E. and Lin, C., (2012). Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research*, September, 93-98. ACM. • Tseng, S.C., Tsai, C.C. (2007) On-line peer assessment and the role of the peer feedback: A study of high school computer course. *Computer & Education*, 49, 1161-1174 • Liu, E.Z.F., Lin, S.S., Chiu, C.H. and Yuan, S.M., (2001) Web-based peer review: the learner as both adapter and reviewer. *IEEE Transactions on education*, *44*(3), 246-251 • Chin, D. (2005) Peer assessment in the Algorithms course. *ITiCSE*, June 27-29, 69-73, Monte de Caparica, Portugal • Sajjadi, M.S., Alamgir, M. and von Luxburg, U., (2016) Peer Grading in a Course on Algorithms and Data Structures: Machine Learning Algorithms do not Improve over Simple Baselines. In *Proceedings of the Third ACM Conference on Learning@ Scale*, 369-378. ACM
2. Strategies for developing assessment instruments	**Bloom's Taxonomy**	• Scott, T. (2003) Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing in Small Colleges*, 19 (1), 267-274

		• Lister, R. and Leaney, J., (2003) Introductory programming, criterion-referencing, and bloom. *ACM SIGCSE Bulletin*, *35*(1), 143-147 • Starr, C.W., Manaris, B. and Stalvey, R.H., (2008). Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin*, 40(1), 261-265 • Thompson, E., Luxton-Reilly, A., Whalley, J.L., Hu, M. and Robbins, P., (2008) Bloom's taxonomy for CS assessment. In *Proceedings of the tenth conference on Australasian computing education*, 78, 155-161 • Alaoutinen, S., Smolander, K., (2010) Student self-assessment in a programming course using bloom's revised taxonomy. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 155-159. ACM • Chatzopoulou, D.I. and Economides, A.A., (2010). Adaptive assessment of student's knowledge in programming courses. *Journal of Computer Assisted Learning*, *26*(4), 258-269
	SOLO **axonomy**	• Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education*. (Bologna, Italy, June 26 - 28, 2006), 118-122. • Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E. and Whalley, J.L., (2008) Going SOLO to assess novice programmers. In *ACM SIGCSE Bulletin*, 40(3), 209-213. ACM • Seiter, L., (2015) Using solo to classify the programming responses of primary grade students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 40-545. ACM • Izu, C., Weerasinghe, A. and Pope, C., (2016), August. A Study of Code Design Skills in Novice Programmers using the SOLO taxonomy. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, 251-259. ACM. • Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M., (2010) Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264 • Shuhidan, S., Hamilton, M. and D'Souza, D., (2009) A taxonomic study of novice programming summative assessment. In *Proceedings of the Eleventh Australasian Conference on Computing*

		Education, 95, 147-156. Australian Computer Society, Inc
	Isomorphic and parameterised questions	• Porter, L., Bailey Lee, C., Simon, B. and Zingaro, D., (2011), August. Peer instruction: do students really learn from peer discussion in computing? In *Proceedings of the seventh international workshop on Computing education research*, 45-52. ACM • Zingaro, D., Porter, L., (2013). Peer Instruction in computing: The value of instructor intervention. *Computers & Education*, 71, 87-96. • Zingaro, D. and Porter, L., (2014) Peer instruction: a link to the exam. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 255-260. ACM • Zingaro, D. and Porter, L., (2015) Tracking Student Learning from Class to Exam using Isomorphic Questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* 356-361. • Porter, L., Zingaro, D. and Lister, R., (2014) Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*, July, 51-58. ACM • Pathak, S. and Brusilovsky, P., (2002) Assessing student programming knowledge with web-based dynamic parameterized quizzes. In *Proc. of ED-MEDIA*, 24-29 • Brusilovsky, P. and Sosnovsky, S. (2005) Individualized Exercises for Self-Assessment of Programming Knowledge: An Evaluation of QuizPACK, *ACM Journal on Educational Resources in Computing*, 5(3).
	Rubrics and primary trait analysis	• Barney, S., Khurum, M., Petersen, K., Unterkalmsteiner, M. and Jabangwe, R., (2012) Supporting Students Improve with Rubric-Based Self-Assessment and Oral Feedback. *IEEE Transactions on Education*, 55(3), 319-325 • Mustapha, A., Samsudin, N.A., Arbaiy, N., Mohammed, R. and Hamid, I.R., (2016). Generic Assessment Rubrics for Computer Programming Courses. *Turkish Online Journal of Educational Technology-TOJET*, 15(1), 53-68. • Smith, L. and Cordova, J., (2005) Weighted primary trait analysis for computer program evaluation. *Journal of Computing Sciences in Colleges*, 20(6), 14-19
3. Assessment tools	**Automated tools for programming**	• Tiantian, W., Xiaohong, S., Peijun, M., Yuying, W. and Kuanquan, W., (2009) Autolep: An automated learning and examination system for programming

		and its application in programming course. In *First International Workshop on Education Technology and Computer Science*, 43-46. IEEE. • Vihavainen, A., Vikberg, T., Luukkainen, M. and Pärtel, M., (2013) Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 117-122. ACM • English, J., English, T. (2015) Experiences of using automated assessment in computer science courses. *Journal of Information Technology Education: Innovations in Practice*, 14:, 237-254 • Bryce, R., Mayo, Q., Andrews, A., Bokser, D., Burton, M., Day, C., Gonzolez, J. and Noble, T., (2013). Bug catcher: a system for software testing competitions. In *Proceeding of the 44th ACM technical symposium on Computer science education*, March, 513-518. ACM. • Burke, Q., Kafai, Y.B. (2012) The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 433–438. ACM • Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P. and Franklin, D., (2013) March. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education*, 215-220. ACM. • Sant. J.A (2009) "mailing it in": email-centric automated assessment. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conf. on Innovation and technology in computer science education*, 308–312, New York, NY, USA. ACM
	Assessing computational thinking	• Werner, L., Denner, J., Campe, S. & Kawamoto, D. C. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 215-220. New York: ACM. • Brennan, K., Resnick, M., (2012) New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*, 1-25. Vancouver, Canada. • Koh, K.H., Basawapatna, A., Nickerson, H. and Repenning, A., (2014). Real time assessment of computational thinking. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium*, July, 49-52. IEEE. • Seiter, L. and Foreman, B., (2013) Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth*

		annual international ACM conference on International computing education research, 59-66. ACM. • Grover, S., Cooper, S. and Pea, R., (2014) Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 57-62. ACM • Moreno-León, J. and Robles, G., (2015). Computer programming as an educational tool in the English classroom a preliminary study. In *Global Engineering Education Conference (EDUCON), 2015 IEEE* (pp. 961-966). IEEE • Portelance, D.J. and Bers, M.U., (2015) Code and Tell: Assessing young children's learning of computational thinking using peer video interviews with ScratchJr. In *Proceedings of the 14th International Conference on Interaction Design and Children*, 271-274. ACM. • Denner, J., Werner, L. and Ortiz, E., (2012) Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, *58*(1), pp.240-249.
4. Assessment instruments	**Concept maps**	• Mühling, A. (2016) Aggregating concept map data to investigate the knowledge of beginning CS students, *Computer Science Education*, 26:, 2-3, 176-191 • Moen, P., (2009). Concept Maps as a Device for Learning Database Concepts. In *7th Workshop on Teaching, Learning and Assessment in Databases*, Birmingham, UK, HEA • Wei, W., Yue, K.B., (2016) Using Concept Maps to Assess Students' Meaningful Learning in IS Curriculum. In *Conference on Information Systems and Computing Education EDSIG*, Las Vegas. • Freeman, L. A., Urbaczewski, A. (2001) Using concept maps to assess students' understanding of Information Systems. *Journal of Information Systems Education*, 12(1), 3-9
	Code comprehension	• Sudol, L.A., Stehlik, M. and Carver, S., (2012). Code comprehension problems as learning events. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 81-86. • Lopez, M., Whalley, J., Robbins, P. & Lister, R. (2008) Relationships between reading, tracing and writing skills in introductory programming, In *Proceedings of the 4th International Computing Education Research Workshop* (ICER'08), 101-112.

		• Murphy, L., Fitzgerald, S., Lister, R. and McCauley, R., (2012) Ability to'explain in plain english'linked to proficiency in computer-based programming. In *Proceedings of the ninth annual international conference on International computing education research*, September, 111-118. ACM. • Simon, Snowdon, S., (2011). Explaining program code: giving students the answer helps-but only just. In *Proceedings of the seventh international workshop on Computing education research*, August, 93-100.
	Debugging	• Fernandez-Medina, C., Pérez-Pérez, J.R., Álvarez-García, V.M. and Paule-Ruiz, M., (2013) Assistance in computer programming learning using educational data mining and learning analytics. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, July, 237-242. ACM. • Ahmadzadeh, M., Elliman, D. and Higgins, C., (2005) An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, 37(3), 84-88
	Multiple-Choice and quizzes	• Lister, R. (2005). One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple-choice exam for first semester programming students. *Seventh Australasian Computing Education Conference* (ACE2005), 155-164 • Roberts, T. (2006). The use of multiple choice tests for formative and summative assessment. *Eighth Australasian Computing Education Conference* (ACE2006), 175-180 • Woodford, K., Bancroft, P. (2005) Multiple choice questions not considered harmful. *Seventh Australasian Computing Education Conference* (ACE2005), 109-115 • Shuhidan, S., Hamilton, M. and D'Souza, D., (2010) Instructor perspectives of multiple-choice questions in summative assessment for novice programmers. *Computer Science Education*, *20*(3), 229-259 • Denny, P., Luxton-Reilly, A. and Simon, B., (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research*, September, 113-124 • Project Quantum • Yin, F., (2006) Applying methods of formative and summative assessment to problem-based learning in computer courses. *The China Papers*, 42-45 • Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P. and Clayborn, L., (2015) *Sowing the*

		seeds: A landscape study on assessment in secondary computer science education. Computer Science Teachers Association., NY, NY. • Kuechler, W. L. & Simkin. M. G. (2003). How Well Do Multiple Choice Tests Evaluate Student Understanding in Computer Programming Classes? *Journal of Information Systems Education*, 14,(4), 389-399