Supporting teachers to assess F–10 Digital Technologies
Literature review

Computer Science Education Research (CSER) Group, The University of Adelaide, supported by Education Services Australia (ESA).

June 2017

Work produced by Rebecca Vivian, Con Lozanovski & Katrina Falkner.

Introduction

This paper outlines assessment requirements as described by the Australian Curriculum (Australian Curriculum, Assessment and Reporting Authority, 2010). It draws together available information from Australian departments of education and training and associations, as well as the latest peer-reviewed research on assessing Digital Technologies. This paper provides some examples and approaches to assessment for Digital Technologies, linking to resources and tools that teachers may adopt in the classroom or utilise in the design of assessment activities.

Assessment and the Australian Curriculum

With any assessment planning, it is important that teachers start with their reporting requirements, and build assessment from there. In this section we provide an overview of general information about assessment and reporting, as advised by ACARA (2010); however, it is important for teachers to follow reporting requirements for their state or territory.

The Australian Curriculum achievement standards provide a focus for teachers in initial planning and programming of teaching and learning activities (ACARA, 2010). They provide a guide for what is expected to be taught by the end of each year band. The achievement standards describe the quality of learning (depth of understanding, extent of knowledge and skill level) expected of students at each year level from Foundation to Year 10 (The Cross Sectoral Assessment Working Party, 2011). Each achievement standard makes explicit the quality of learning required for students to progress to the next level. Teachers, using a range of assessment strategies, will ensure that each student has a portfolio of work that demonstrates evidence of what the student has achieved.

According to the Queensland Curriculum and Assessment Authority (2015), Level C is the achievement standard in its original form, with A–B and D–E being variations of achievement above and below (see Table 1: Reporting standards A–E assessment). Unless provided by their state or territory, teachers can work with this achievement standard as the first starting point, and develop progressions for either side.

Table 1: Reporting standards A–E assessment (Queensland Curriculum and Assessment Authority, 2015)

A	B	C	D	E
Evidence in a student's work typically demonstrates a **very high level** of knowledge and understanding of the content (facts, concepts, and procedures), and application of skills.	Evidence in a student's work typically demonstrates a **high level** of knowledge and understanding of the content (facts, concepts, and procedures), and application of skills.	Evidence in a student's work typically demonstrates a **sound level** of knowledge and understanding of the content (facts, concepts, and procedures), and application of skills.	Evidence in a student's work typically demonstrates a **limited level** of knowledge and understanding of the content (facts, concepts and procedures), and application of skills.	Evidence in a student's work typically demonstrates a **very limited level** of knowledge and understanding of the content (facts, concepts and procedures), and application of skills.

Table reproduced from Queensland Curriculum and Assessment Authority. © The State of Queensland (Queensland Curriculum and Assessment Authority) 2017.

Teachers use the achievement standards at the end of a period of teaching to make judgements about student learning (whether the student has achieved below, at or above the standard). Teachers draw on assessment data they have collected as evidence during the course of the teaching period to support their judgements, as well as to review their own practice.

Assessment of student learning takes place at different levels and for different purposes, including:

- **ongoing formative assessment** within classrooms for the purposes of monitoring learning and providing feedback, for teachers to inform their teaching, and for students to inform their learning.
- **summative assessment** for the purposes of twice-yearly reporting by schools to parents and carers on the progress and achievement of students.

Teachers design and implement assessment approaches and strategies depending on their purpose. There is a variety of assessment strategies available, including anecdotal records, authentic tasks, checklists or scales, conferences, contracts, games, diagnostic inventories, peer evaluation, portfolios, rubrics, self-evaluations, simulations, learning journals and teacher observations.

Designing assessment activities

Assessment is the process of gathering and interpreting evidence to make judgements about student learning and is the link between learning outcomes, content and teaching and learning activities (The Cross Sectoral Assessment Working Party, 2011). The design of high quality instruments and tasks can help to ensure effective and authentic assessment delivery. It is advised that assessment design should focus on what students know and understand, and what they need to know to progress (Victorian Department of Education and Training, 2017). Assessment should align with the achievement standards and content descriptions in the Australian Curriculum: Digital Technologies, and what is taught and learnt in the classroom.

It is recommended that teachers focus on assessing one or two content descriptions from the Australian Curriculum: Digital Technologies in one unit, with selected content descriptions that might also be covered explicitly within a unit from another learning area (Victorian Department of Education and Training, 2017). This can help teachers to focus assessment and provide multiple opportunities for students to demonstrate their achievement within a unit.

The Victorian Department of Education and Training (2017) provides some useful guidance for designing assessment, which includes starting by adding the achievement standard(s) to the top of the unit plan, and highlighting any relevant parts. Teachers must also ensure that the assessment criteria for achievement also includes what achievement looks like at the level above the achievement standard and below, and ensure that the lesson sequence includes activities that enable students to work at their appropriate level.

Teachers can involve students in the process of developing assessment from an early phase. Students could be invited to co-design assessment rubrics or activities, or to identify the ingredients for success. It is recommended that teachers share rubrics or criteria and assessment materials and activities with students early so that they know what they are being assessed on and how (Victorian Department of Education and Training, 2017).

Assessment approaches and strategies

In this section, we discuss some of the most recent strategies in digital technologies literature (also known as computer science education), in relation to approaches to assessment.

Programming assignments

In addition to open-ended programming tasks, specific assignments that test students' ability to draw on their knowledge of a programming concept, problem-solving skills and programming skills can be used as formative assessments after they have learned the required content and have had a chance to practise their skills and knowledge (Grover, Cooper and Pea, 2014). For example, Grover, Cooper and Pea suggest the following visual programming assignments, designed to specifically test programming concepts through open-ended tasks:

- a 'spirograph' (implementing *nested loops*)
- a polygon generator, depending on a size and shape specified by the user (implementing *user inputs and variables*)
- 'four-quadrant art', which colours the screen in different colours depending on the position of the sprite (implementing *conditionals with compound Boolean conditions*)
- two-paddle pong (implementing *conditionals within repeat-until loops*)
- 'guess my number' game (implementing a *combination of constructs taught*).

For teachers looking to develop assignment activities, the Scratch (https://scratch.mit.edu/tips) or ScratchJR (scratchjr.org/teach/activities) cards available online could provide the seed of an idea. For teachers in secondary years, Nifty assignments (nifty.stanford.edu) at the introductory programming level provide more complex ideas, including general-purpose programming and object-oriented programming assignments.

Grover, Cooper and Pea recommend that rubrics could be one way to evaluate students' assignments. Other evaluation techniques could involve checklists in which the teacher marks the presence of certain constructs, or peer review in which students evaluate their peers in terms of achieving a particular assignment goal based on a set of criteria.

Artefact analysis

The production of a programming project can provide an engaging summative assessment activity for students to demonstrate their knowledge and application of skill. Projects could include 'programming assignments' with a more defined goal and set of constraints, or could be more open-ended tasks in which the student has more autonomy over the project designs. The programming project then forms an artefact for analysis. When students are required to iteratively test and evaluate their projects, projects can also form part of the formative assessment process, as students use the feedback to improve their projects and to identify gaps in their knowledge that need to be filled in order to progress.

One example of a free tool that can support the evaluation of visual programming projects in Scratch (MIT Lab, 2016) is Dr Scratch (2014). This tool could be used by teachers in formative and summative processes, or by students as they undertake self- or peer-evaluation. The authors (Moreno-León et al, 2017) have developed a framework that underpins Dr Scratch for assessing projects, linking computational thinking and programming elements and activities (see Table 2: Computational thinking and programming). The authors have identified three indicators of progression, which inform their program analysis scoring system, based on their review of other previous computational thinking assessments. Although this tool can only be used to assess projects within the Scratch environment, the scoring system may be useful in guiding the development of rubrics and supporting teachers in what to look for within other visual programming environments. The authors (Moreno-León et al, 2017) caution that this tool is not designed to replace evaluators or teachers, but rather to assist teachers and learners in assessment tasks.

Table 2: Computational thinking and programming (Moreno-León et al, 2017)

Element	Scoring system (progression of complexity introduced into visual programs)
Logical thinking	1. IF 2. IF Else 3. Logic operations
Data representation	1. Modifiers of object properties 2. Variables 3. Lists
User interactivity	1. Green flag 2. Keyboard, mouse, ask and wait 3. Webcam, input sound
Flow control	1. Sequence of blocks 2. Repeat, forever 3. Repeat until
Abstraction and problem decomposition	1. More than one script and more than one sprite 2. Use of custom blocks 3. Use of 'clones' (instances of sprites)
Parallelism	1. Two scripts on green flag 2. Two scripts on key pressed or sprite clicked 3. Two scripts on receive message, video/audio input, backdrop change
Synchronisation	1. Wait 2. Message broadcast, stop all, stop program 3. Wait until, when backdrop changes, broadcast and wait

Table reproduced from Moreno-León et al (2017). © 2017 Association for Computing Machinery, New York, NY, USA.

Using expert evaluators on students' Scratch projects (Moreno-León et al, 2017), the authors were able to find reasonable correlations between human expert scores and those produced by Dr Scratch; however, the authors acknowledge that further refinement is needed to increase accuracy. A limitation of Dr Scratch is that the tool does not assess fundamental aspects of programming, such as debugging, design or remixing skills, nor does it assess other aspects of computational thinking, such as creativity or correctness (Moreno-León et al, 2017). These are areas that educators and learners using the tool could build in for additional consideration.

Giordano et al (2015) identify other automated tools to support artefact assessments across visual programming environments, Scratch and App Inventor, and the Java programming language. However, their application in the classroom by teachers to evaluate achievement standards has not been explored extensively beyond university and research contexts.

While a student may produce a working programming project, it might not necessarily imply that they understand programming constructs or be able to explain how their design works. Cognitive interviewing is one assessment approach that can support artefact analysis; this is referred to as 'artefact-based interviews' (Brennan and Resnick, 2012). Although Brennan and Resnick have used artefact-based interviews to learn about students' general use of Scratch, their questions relating to project creation can also provide some starting points for teachers to design interview questions. They include questions relating to project framing, such as 'How did you get the idea for your project?' and project process, such as 'How did you get started making your project? What happened when you got stuck?' They then delve further into students' understanding of programming constructs by looking at the code with the student and inviting the student to explain how it works.

Cognitive interviewing (think aloud)

Cognitive interviews (or think-alouds) can provide insight into learners' deeper understanding of how something works, or their level of understanding of concepts and content.

In cognitive interviews with young children, authors Brennan and Resnick found that when prompted to explain how their code worked in visual programming, learners were not always able to articulate their understanding. For example, in one study the authors conducted a cognitive interview with a young child who appeared to have created a complex Scratch project. However, when asked to explain his code, the learner was unable to explain any of it. Upon discussion with the child, they discovered that he had replicated programming blocks from another project. The authors caution that this is a 'strong reminder that the presence of a code element in a project is not necessarily an indicator that the designer possesses a deep understanding of the code element' (Brennan and Resnick, 2012).

Other researchers (Grover and Basu, 2017) have found that, despite completing a course on introductory programming using Scratch, students still held misconceptions with the use of variables and had difficulty with grasping other introductory programming aspects, such as how loops and Boolean operators worked. Grover and Basu used cognitive think-alouds with incorrect responses on quiz questions. They invited learners to talk through their reasoning and problem-solving processes and in doing so, they were able to more easily identify the types of misconceptions learners held and why. Further, the researchers found that incorporating assessment activities that require students to describe what is happening in their code can help students to better understand programming concepts and realise their own misconceptions. For example, working through the process of what is happening in each iteration of a loop (including tracing variable values) can help students understand how sequences of actions inside loops are repeated and how to use variables and expressions in the context of loops.

Rather than including all programming concepts (loops, decisions, etc), in cognitive interviews, teachers can focus on a select few key elements that appear within the achievement standard and content descriptions in the particular year level that they are working with.

Test questions

Question types can include open response, true/false, multiple-choice, an activity or a game (Giordano et al, 2015). In their review, Giordano et al provide a comprehensive list of repositories available around the world for supporting digital technologies subject matter and, in particular, visual programming, computational thinking, and general-purpose and object-oriented programming. Some examples include Nifty assignments (nifty.stanford.edu), Project Quantum (community.computingatschool.org.uk/resources/4382), and the Bebras Computational Thinking Challenge (bebras.edu.au).

In addition to programming activities, it is important that teachers measure students' conceptual understanding. Various types of assessment can help to identify misconceptions and problematic programming concepts that students will require in future learning (Grover and Basu, 2017). Teachers can assess students' computational thinking, and understanding of programming concepts and code by using test questions that involve students reading and tracing code and predicting the output.

A common base for assessing computational thinking with quizzes and tests has been the adoption of Bebras Computational Thinking Challenge questions within the classroom for learning activities, demonstrations of content, and for formative assessment (Dagien and Sentance, 2016). In their review of Bebras tasks for curriculums, the authors were able to identify that tasks were able to broadly fall into one of the five categories:

1. algorithms and programming
2. data, data structures and representations (including graphs and data mining)
3. computer architecture and processes (including anything to do with how the computer works – scheduling, parallel processing)
4. communications and networking (including cryptography and cloud computing)
5. interaction (including human–computer interaction), systems and society.

The categories identified align with the content descriptions in the Australian Curriculum: Digital Technologies, thus providing a pool of questions that could be used by teachers. The authors go further to align the Bebras 2015 tasks with fundamental computational thinking skills of abstraction, algorithmic thinking, decomposition, evaluation and generalisation, providing sets of question examples that align with each computational thinking skill. A limitation identified by the researchers is that the planning of lessons around relevant Bebras tasks can only be achieved if Bebras tasks are available online and if the content is explicitly signposted. Currently, teachers need to download available questions. However, the authors highlight that future work for Bebras will involve developing a two-dimensional categorisation system that educators can adopt and that can facilitate countries in developing databases of Bebras questions that align with their own curriculum (Dagien and Sentance, 2016).

Teachers can also develop their own quizzes for digital technologies that are aligned with the content descriptions and achievement standard for their year level. These can be used for formative or summative assessment purposes. Multiple-choice quizzes or open-ended questions can include snippets of code, based on the programming language that students are using in learning activities. This can help learners develop familiarity with code tracing and the ability to understand an algorithm in visual programming or pseudocode (Grover, Cooper and Pea, 2014).

Grover and Basu (2017) developed a series of questions that emphasised the definition of focal knowledge, skills and abilities required of students. Their assessment questions included a blend of questions in the context of the Scratch programming language, which students had been using, as well as other questions that assessed broader algorithmic thinking and problem-solving skills required for coding in Scratch. Examples of their assessments included a code comprehension task that involved students reading, tracing and predicting the output of code by writing their response, as well as questions that tested students' knowledge of programming concepts, such as logical expressions. Two examples of the questions are below, in Figure 1 and Table 3.

Figure 1: Sample question included in Grover and Basu's assessment activity (2017)

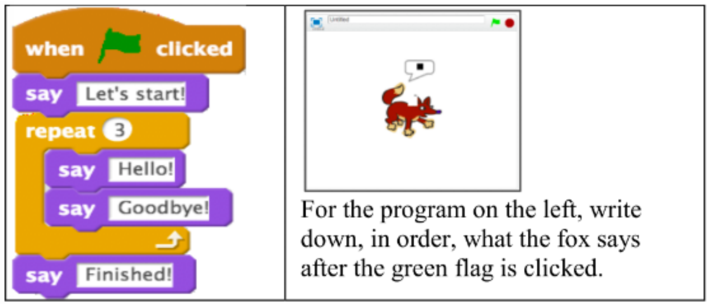
```
when [flag] clicked
say [Let's start!]
repeat (3)
    say [Hello!]
    say [Goodbye!]
say [Finished!]
```
For the program on the left, write down, in order, what the fox says after the green flag is clicked.

Figure reproduced from Grover and Basu (2017). © 2017 Association for Computing Machinery, New York, NY, USA.

Table 3: Sample question included in Grover and Basu's assessment activity (2017)

Logical expression	Words	
(Starts with a D) **AND** (ends with an E)	❒ Dance ❒ Delicious	❒ Soccer ❒ Share
(Starts with a D) **AND** does **NOT** (end with an E)	❒ Dance ❒ Delicious	❒ Soccer ❒ Share
(Starts with a D) **OR** (ends with an E)	❒ Dance ❒ Delicious	❒ Soccer ❒ Share
(Starts with at D) **OR** does **NOT** (end with an E)	❒ Dance ❒ Delicious	❒ Soccer ❒ Share

Table reproduced from Grover and Basu (2017). © 2017 Association for Computing Machinery, New York, NY, USA.

The authors designed twice as many questions as required and invited a panel of experts to review the items. In the school setting, teachers can work with colleagues to design and review questions and select from a pool of questions that are deemed to best fit the purpose and content being assessed.

Other educators have set quizzes with pseudocode (algorithms in plain English) and jumbled blocks of code that students have to piece together in the correct order (known as Parson puzzles) (Grover, Cooper and Pea, 2014). Flowcharts can also be included in quizzes; students can select the correct flowchart or identify an error or missing component in the design.

Giordano et al (2015) advise that when educators design questions to be re-used across a number of contexts, the questions should be as independent as possible from syntax-related constraints. Questions should ideally be contextualised to learners' real life situations (changing with age, social background, context, etc), or be based on a familiar or engaging topic. Questions should also scaffold learners through higher level cognition process and discovery. Giordano et al suggest that some examples of questions could include:

- a sequence of questions scaffolding a specific topic (eg recursion) using a visual language
- a puzzle to illustrate a computer science concept
- a game suitable to teach parallelism and concurrency to middle school students
- a data structure problem related to questions posed in different ways.

Giodano et al's (2015) review of two research works on computer science questions identified a range of 12 question types for assessing students' programming and understanding of constructs. These included:

- tracing a given solution
- analysing code execution
- finding the purpose of a given solution
- examining the correctness of a given solution
- estimating efficiency
- completing a given solution
- manipulating instruction
- programming-style questions
- developing a solution

- developing a solution using a given module
- designing questions
- transforming a solution.

Not all question types are suitable for a particular format (eg multiple-choice) and it is up to the question designer to select the format that is most appropriate for the question being solved.

Problem-based puzzle projects

Learning activities, such as puzzles and games, can form the basis of engaging assessment activities. Similar to the Bebras Computational Thinking Challenge, CS Unplugged (Bell, Witten and Fellows, 2015) is a series of computational thinking activities, including games and puzzles, that can be done without a computer. These activities can be used within lessons, but also as potential formative assessment activities.

Rodriguez (2015) and Rodriguez et al (2017) have extended the suite of CS Unplugged activities to include assessments. The authors have designed two 'project' samples designed to be similar in form but with variations, so that a concept tested by one question in the first project (pre-test) would be tested in a similar question in the second project (post-test). The questions are open-ended, so that students aren't led into a desired outcome. Each project has its own theme or common story, which puzzles and challenges are designed around. In the examples, the authors have designed a 'Pets' theme and a 'Carney' (carnival) theme. The authors created a set of questions per theme and a rubric, with three levels (proficient, partially proficient and unsatisfactory) for each question to aid assessment. Each question takes cues from the CS Unplugged materials and applies them to a new context, requiring students to draw on knowledge acquired.

Table 4 below shows each topic of the comprehensive project along with its associated 'story' for the Pets and Carney final versions, the computational thinking skills, and the categories for Bloom's taxonomy.

Table 4: Question topics aligned with computational thinking and Bloom's taxonomy

Topic	Pets story	Carney story	Computational thinking skills	Bloom's
Character encoding	Melanie Mouse secret message	Detective secret message	Data representation	Remembering Understanding
Search	Delilah Dog shoe search	Odin book search	Algorithmic thinking	Evaluating
MST*	Tunnelling ants	Railroad	Abstraction Algorithmic thinking	Applying
FSA*	Delilah Dog schedule	Fortune-telling robot	Abstraction	Creating Analysing
Binary numbers	Animals hiding	Carney workers hiding	Data representation	Remembering Understanding

Table reproduced from Rodriguez et al (2017). © 2017 Association for Computing Machinery, New York, NY, USA.
*MST = Minimal spanning tree, FSA = Finite state automata.

The authors (Rodriguez et al, 2017) have made their resources available on their website (http://csunplugged.mines.edu). These resources could be selected by teachers looking to test specific computational thinking skills, or the model could be used to inspire the creation of other 'projects', including include puzzles and problems around a common classroom theme.

Rubrics

A rubric can be in the form of a matrix or grid, containing a benchmark of various levels of achievement. Teachers use rubrics as a way to evaluate and grade students' work against an explicit set of criteria and expected standards of performance.

Familiar frameworks, such as the Structure of Observed Learning Outcomes (SOLO) taxonomy, provide a foundation to guide the development of rubrics for assessing digital technologies activities and projects. In one such example, Ginat and Menashe (2015) describe a SOLO taxonomy for assessing algorithmic design, building on that produced by Whalley's code-writing taxonomy. Their taxonomy and examples are situated within the context of general-purpose programming, suitable for secondary activities. In their paper, Ginat and Menashe (2015) walk through their evaluation reasoning of real programming responses using pseudocode samples, thus providing a guide for others developing similar rubrics or using their rubric to assess students' work.

Table 5: SOLO taxonomy for assessing algorithmic design and code writing (Ginat and Menashe, 2015)

	Prestructural [P]	Unistructural [U]	Multistructural [M]	Relational [R]	Extended abstract [A]
Algorithmic design	Substantially lacks knowledge of selection and implementation of generic design patterns	Directly translates the specifications into a straightforward implementation of a generic design pattern	Translates the specifications into flexible manipulation of a generic design pattern; or a simple elementary composition of more than one generic pattern.	Produces a valid well-structured solution that involves the composition of two or more design patterns, integrated in a non-simple, interleaved manner to form a logical whole.	Insightfully capitalises on hidden task characteristics; and/or a generalised structure that encapsulates abstraction beyond the required solution.
Code writing	Substantially lacks knowledge of programming constructs or response is unrelated to the question	Represents a direct translation of the specifications. The code is in the sequence of the specifications.	Represents a translation that is close to a direct translation. The code may have been reordered to make a valid solution.	Provides a valid well-structured program that removes all redundancy and has a clear logical structure. The specifications have been integrated to form a logical whole.	Uses constructs and concepts beyond those required in the exercise to provide an improved solution.

Table reproduced from Ginat and Menashe (2015). © 2015 Association for Computing Machinery, New York, NY, USA.

As SOLO levels are related to both tasks and solutions, the authors unpack the types of tasks at various SOLO taxonomy levels to provide a framework for, not just the assessment of achievement, but also the design of assessment activities (Ginat and Menashe, 2015).

These types include:

- **unistructural tasks:** Require a direct application of a generic pattern, such as simple counting
- **multistructural tasks:** Requires direct application of two or more generic patterns in a simple composition, such as reversing an input list
- **relational tasks:** Requires an interleaved composition of patterns, such as combining search and counting
- **extended abstract tasks:** When a task enables abstract aspects, which reflect generalisation and that may occur with parameteriation of specified task elements, such as a task grouping girls and boys with provided information.

Frameworks, such as the SOLO taxonomy, can provide a foundation from which educators can design rubrics and assessment activities that flag indicators of progression and take into account performance across achievement standard elements.

Rubrics have also been developed around the focus of concepts or constructs (Baille et al, 2010; Sherman and Martin, 2015). For example, Sherman and Martin develop a ‘mobile computational thinking’ framework for assessing students’ mobile apps (using MIT App Inventor). In their framework they identify ‘computational thinking’ and ‘mobile computational thinking’ dimensions for assessment (see Table 6).

Table 6: Computational thinking and mobile computational thinking dimensions (Sherman and Martin, 2015)

General computational thinking	Mobile computational thinking
Naming	Screen interface
Procedural abstraction	Events
Variables	Component abstraction
Loops	Data persistence
Conditionals	Data sharing
Lists	Public web services
	Accelerometer and orientation sensors
	Location awareness

Table reproduced from Sherman and Martin (2015). © 2015 by the Consortium for Computing Sciences in Colleges.

Their rubric measures along a four-point scoring system. The following is an example of the marking progression across one of the dimensions:

Table 7: Screen interface

1 point	2 points	3 points	4 points
Single screen with five or fewer visual components that do not programmatically change state.	Single screen with more than five visual components that do not programmatically change state.	Single screen, where some components programmatically change state based on user interaction with the app.	Two or more screens; screens may be implemented as screen components, or by programmatically changing visibility of groups of visual components.

Table reproduced from Sherman and Martin (2015). © 2015 by the Consortium for Computing Sciences in Colleges.

In their trial of the rubric with introductory programming students, they found that the breadth of the rubric's dimensions allowed for more subtle variations to be isolated, particularly looking at the presence of specific concepts. They have found it to be a useful rubric guide for assessing apps developed with App Inventor by MIT; however, this could also be extended to other app development environments for secondary students.

Baille et al (2010) have further developed rubrics that are general-purpose programming assignment-specific. They begin with a faculty (or school-wide) generalised programming rubric template as a starting point and format for assessment to ensure consistency and objectivity. They then take the template and customise the progressions to include assignment-specific details. Having fine-grained details can further support teachers in the objective evaluation of projects against a criteria and better support consistency between students and classes.

Rubrics are also very familiar to the primary classroom and can be a useful way to objectively evaluate students' visual programming projects. A number of educators within the ScratchED (scratched.gse.harvard.edu) community have shared their own developed rubrics for use with Scratch activities.

Conclusions

A number of assessment approaches that have been investigated by computer science education researchers are suitable to implement with the Australian Curriculum: Digital Technologies. However, a review of available literature reveals that the majority of works relate to the assessment of computational thinking and programming. Future work will investigate research on assessment from mathematics and more general education research. This will inform assessment practices for a broader curriculum, including topics such as data representation and manipulation, collaboration and project management, and the acquisition of content knowledge beyond programming constructs, such as explaining how networks work and identifying digital systems and skills in relation to safety and privacy when communicating online and with others.

References

Australian Curriculum, Assessment and Reporting Authority (ACARA). (2010). Implications for teaching, assessing and reporting. Accessed 20 May 2016, www.australiancurriculum.edu.au/overview/implications-for-teaching-assessing-and-reporting

Bailie, F., Marion, B., & Whitfield, D. (2010). ‘How rubrics that measure outcomes can complete the assessment loop’. *Journal of Computing Sciences in Colleges*, 25(6), 15–25. Retrieved from http://dl.acm.org/citation.cfm?id=1791135

Bell, T., Witten, I., & Fellows, M. (2015). CS Unplugged. Retrieved 13 June 2017, from http://csunplugged.org

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

Dagien, V., & Sentance, S. (2016). ‘Informatics in schools: Improvement of informatics knowledge and perception’. In A. Brodnik & F. Tort (Eds.), *ISSEP* (Vol. 9973, pp. 28–39). Cham: Springer International Publishing. http://doi.org/10.1007/978-3-319-46747-4

Dr Scratch. (2014). Dr Scratch. Retrieved 20 June 2017, from http://www.drscratch.org

Ginat, D. & Menashe, E. (2015). ‘SOLO taxonomy for assessing novices' algorithmic design’. In Proceedings of the 2015 ACM SIGCSE Technical Symposium on Computer Science Education – SIGCSE ’15. (pp. 452–457). New York, New York, USA: DOI: http://dx.doi.org/10.1145/2676723.2677311

Giordano, D., Maiorana, F., Csizmadia, A. P., Marsden, S., Riedesel, C., Mishra, S., & Vinikiene, L. (2015). ‘New horizons in the assessment of computer science at school and beyond’. In *Proceedings of the 2015 ITiCSE on Working Group Reports – ITICSE-WGR ’15* (pp. 117–147). New York, New York, USA: ACM Press. http://doi.org/10.1145/2858796.2858801

Grover, S., & Basu, S. (2017). ‘Measuring student learning in introductory block-based programming’. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education – SIGCSE ’17 (pp. 267–272). New York, New York, USA: ACM Press. http://doi.org/10.1145/3017680.3017723

MIT Media Lab. (2016). Scratch. Retrieved 20 June 2017, from https://scratch.mit.edu

Moreno-León, J., Harteveld, C., Román-González, M., & Robles, G. (2017). 'On the automatic assessment of computational thinking skills: A comparison with human experts'. In *Conference on Human Factors in Computing Systems (CHI)* (pp. 2788–2795). Denver, Colorado. http://doi.org/10.1145/3027063.3053216

Rodriguez, B. R. (2015). *Assessing Computational Thinking in CS Unplugged Activities*. *Master of Science (Computer Science).* Colorado School of Mines. Retrieved from http://hdl.handle.net/11124/169998

Rodriguez, B. R., Kennicutt, S., Cyndi, R. A., & Camp, T. (2017). 'Assessing computational thinking in CS Unplugged activities'. In *Technical Symposium on Computer Science Education (SIGCSE)* (pp. 501–506). Seattle, WA. http://dx.doi.org/10.1145/3017680.3017779

Sherman, M., & Martin, F. (2015). 'The assessment of mobile computational thinking'. *Journal of Computing Sciences in Colleges*, 30(6), 53–59.

The Cross Sectoral Assessment Working Party. (2011). *Teachers' Guide to Assessment*. Australian Capital Territory, Australia. Retrieved from http://www.education.act.gov.au/__data/assets/pdf_file/0011/297182/Teachers_Guide_to_Assessment_Web.pdf

Victorian Department of Education and Training. (2017). Digital Technologies Curriculum: Assessment. Retrieved 20 March 2017, from http://www.digipubs.vic.edu.au/pubs/digitaltechnologies/digital-technologies-curriculum_assessment