Difficulties with design: The challenges of teaching design in K-5 programming

J Waite, P Curzon, D.W Marsh

Queen Mary University of London

S Sentance

King's College London

Abstract

Teachers in England are required to ensure that learners from the age of five are taught about algorithms and program design. Yet, there is evidence that despite teachers reporting that design is important, they are not converting this into classroom practice. This paper describes a survey study, in which we explored teachers' difficulties in using design. We surveyed 207 teachers asking them free-text questions on their use of design in teaching programming and their views of pupils' responses to using design. In the survey, we also investigated teachers' understanding of the term algorithm, an essential concept which may be a contributing factor in their difficulties with design. We provide underpinning data on the difficulties of using design that teachers of pupils aged from 5 to 11 years old (Grades K to 5) have in teaching programming. Difficulties with design identified include pupil resistance, a lack of time, a lack of pupil and teacher expertise, conflicting pedagogical choices and a general confusion over what an algorithm is. There were statistically significant differences in selection of the term 'algorithm' to describe programming artefacts whether a teacher was a specialist or a generalist, what training they had received on programming or design, the age group taught and programming language used. Teachers were more likely to call a complex code snippet an 'algorithm' than a simpler one and more likely to select the term to describe code snippets than a design artefact. We make suggestions of how to alleviate the problems including that teachers are introduced to the idea of ambiguous representations of algorithms and process which refines the representation from ambiguous to unambiguous as the design progresses.

Keywords: K-5 Computing Education, Teachers, Design, Programming

1. Introduction

In September 2014, a new computing curriculum was introduced in England. This includes teaching 5 to 6-year-old children (Grade K to 1) to understand what an algorithm is, how algorithms are implemented as programs and to create simple programs. 7 to 11-year-old children (Grade 2 to 5) are taught to use logical reasoning to explain how simple algorithms work, to be able to detect and correct errors in algorithms and programs, and to design and write programs (Department for Education, 2013a). Similar curriculum changes are being introduced in other countries (Hubwieser et al., 2015). However, there is little research on the pedagogy of teaching algorithms and design in programming to this age group, or even on understanding the problems to be overcome (Waite, 2017). Recent research indicates that teachers think design is 'Essential' or 'Very Useful' but are not converting this view of importance into action of using it in class (Waite et al., 2018b) .

1.1. Aim and contributions

Our aim is to investigate what difficulties teachers are encountering to constrain their introduction of design. Specifically, we investigate, through a teacher survey, what drawbacks teachers think there are of using design, and what negative responses they perceive their pupils have in using it. We also consider whether teachers' understanding of the term 'algorithm', an important aspect of design, is compounding difficulties with design. Our overarching research question, and sub-questions are

- **RQ1: What are K-5 teacher's difficulties in using design?**
  - RQ1-1: What are teachers' views of the difficulties of using design?
  - RQ1-2: Do teachers demonstrate a shared understanding of the term algorithm when viewing images of code snippets and design artefacts?

The contributions are:

- to reveal a series of difficulties teachers have using design;
- to show that teachers do not have a shared understanding of the term 'algorithm'
- to start to address the confusion around algorithms in primary programming, through a model of informal refinement of representations of algorithms from ambiguous to unambiguous as design progresses.

1.2. Report structure

We first outline the literature related to design and algorithms including a summary of our relevant previous research. We then explain the methodology. Following this, quantitative and qualitative results are provided. In the discussion, we link results to literature and synthesise this to present a model of difficulties of using design. We then reflect on how practitioners and educational designers might use our findings and finish by considering our research within a broader context.

2. Background and literature review

In reviewing the literature related to design and algorithms, we start by considering design in programming generally. We then explore what an algorithm is in the contexts of: formal computer science; in the classroom teaching of mathematics, and in teaching computing at different ages. We finally look at representations of algorithms in design and how ambiguous representations are handled.

2.1. Design in programming

The study of the importance of design in programming has a long history for older learners. Soloway (1986) proposed that teaching programming to undergraduates was about teaching problem solving, rather than teaching just the syntax and semantics of a language, recommending that students be taught explicitly that programming is a design discipline (Soloway, 1986, p. 853). A recurrent research theme concerns how expertise in design, in general, develops for adults, including software engineering. (Cross, 2004).

In K-12 contexts, design has until recently been generally lacking both in research undertaken and in learning resources (Rich et al., 2017; Falkner and Vivian, 2015). The reuse of industry design approaches has been suggested for school-aged pupils, such as a simplified engineering design process for very young learners (ages 3 to 5 years old) and agile methods have been successfully tested with German high school students (ages 17 and 18 years old) (Bers, 2017; Kastl et al., 2016). A general framework for developing digital artefacts for any school-aged pupil has been suggested, which combines analysis and exploration with design and construction, but this has not yet been rigorously tested in schools (Schulte et al., 2017). Specific approaches for particular contexts have been trialled such as when teaching the design of games with physical computing components with middle school pupil (aged 12 to 16 years old) (Kafai and Vasudevan, 2015). However, research of design in programming with K-5 learners (aged 5 to 11-years-old), or their teachers, set within the English Computing Curriculum context is rare. Added to this, few K-5 teachers in England are thought to have been taught programming or design themselves, therefore further research, support and training have been called for (The Royal Society, 2017a). Recent research suggests teachers in England think design is useful but avoid doing it (Waite et al., 2018b) . However, there has been no research to understand the reasons behind this. This is a gap we fill.

Irrespective of the vacuum of evidence on how to teach design, teachers in England are required to teach programming including algorithms. The association between design and algorithms is in the English Computing Curriculum: learners of age 5 to 7 are required to implement algorithms as programs (Department for Education, 2013a).

2.2. What is an 'Algorithm'?

Terms have different meanings according to the context and definitions change over time. We first look at the definition of 'algorithm' from a historical perspective, then at its use and meaning in maths classrooms, in computer science in general and in computing classrooms. We look specifically at the representations used for algorithms as distinct from the abstract concept. Finally, we reflect upon representations of algorithms in design.

2.2.1. A Historical Definition

The Oxford English Dictionary defines an algorithm as 'a procedure or set of rules for calculation or problem solving, now esp. with a computer' and its origin dates from the late 17th century (Oxford University Press, 2018). The concept thus derives from before the computer age and originally algorithms were procedures for people to follow, where the representation of such algorithms might be verbal, drawn or written instructions.

2.2.2. In English primary mathematics classrooms

A core part of primary school (K-5) is in learning specific algorithms in this dictionary sense and how to follow them exactly without making mistakes (e.g., algorithms for addition, subtraction, multiplication and division). However, the term algorithm is not commonly used in the current mathematics curriculum. Instead, the terms mental and written methods are used (Department for Education, 2013d). It is recommended such methods are introduced to learners through manipulatives, visual representations and worked examples. Pupils are encouraged to express their understanding verbally and in writing (Hodgen et al., 2018). Representations of classroom mathematical methods are introduced by teachers as they explain methods, such as informal drawing of a number line with arrows depicting the steps needed to answer a question, or a written list of what to do to solve a problem including an annotated example. These representations are important teaching aids. As pupils apply methods, representations of the application are captured for pupil and teacher assessment. For example, a pupil might be video recorded as they use their fingers to work out a sum or photos taken of a pupil using blocks to solve a problem. Not all representations of applied methods are recorded, such as a non-recorded verbal description of a child using number facts to work out an answer. These representations are informal and are not required to adhere to a particular format, be produced in a specific media or depicted at a particular level of detail. Yet, learners are required to develop **reliable** methods, understand which methods to use and why, and apply them **accurately** (Department for Education, 2013d).

There is a clear distinction between the application of a method by a pupil, which must be reliable and accurate, and the representation of the method, which for primary maths is likely to be personally meaningful but ambiguous to others without pupil explanation.

2.2.3. In computer science

In the modern context of computer science, a more computer-centred definition of the word algorithm has been developed. Denning (2017) draws attention to an algorithm being for a machine where 'algorithms are directions to control a computational model (abstract machine) to perform a task' (Denning, 2017, p.4) and each operation has a 'well defined effect that can be carried out by a machine' (Denning, 2017, p.8). An algorithm in this sense is a precise process, where each action within it is unambiguous and definite (one of the five requirements for a procedure to be called an algorithm) (Knuth, 1997, p.5). These actions are selected from a pre-agreed elementary instruction set and for any legal input, when executed, the algorithm will do what it is intended to do, producing the output as required (Harel and Feldman, 2004, p.16). An algorithm can also be understood as an object, rather than a process, in terms of its inputs and outputs, though the algorithm is still precise and unambiguous (Sfard, 1991).

At face value, comparing the contexts of informally documented mathematical methods which five to eleven-year-old pupils are expected to develop to learn about maths

to the formal context of a computer science meaning of an algorithm there appears to be significant differences in terms of ambiguity. However, we will return to this after we have considered representations of algorithms.

2.2.4. In English secondary computing classrooms

In secondary classrooms, learners (age 12-18, Grades 6-12), are required to understand common sorting and searching algorithms (Department for Education, 2013b, p.2) including understanding that a key property of any algorithm is that it must be precise and unambiguous. In guidance on teaching to this age group, structured English (pseudocode) or flowcharts are suggested as forms of representation (Kemp, 2014, p. 8). For older learners, representation becomes more formal as reference language (Cutts et al., 2014) is introduced and in higher education logic as a way to represent algorithms (Harel and Feldman, 2004, p. 113). This third context of older learners in computing lessons maps closely with the formal computer science definition.

2.2.5. In English primary computing classrooms

In an early reference document for teachers of pupils aged 5 to 11 (K-5), published to coincide with the introduction of the 2014 English Computing curriculum the need for a precise algorithm was laid out, but with ambiguous unplugged examples:

> *"An algorithm is a precisely defined procedure - a sequence of instructions, or a set of rules, for performing a specific task (e.g. instructions for changing a wheel or making a sandwich). While all correct algorithms should produce the right answer, some algorithms are more efficient than others."* (p.7 Berry, 2013, p.27)

These common, unplugged activities, such as making a jam sandwich, claim to develop computational thinking, including the understanding that precision is needed. However, these so-called algorithms are unlikely to be carried out by a machine and are therefore in opposition to some definitions of an algorithm as outlined above. Whether unplugged activities support the progression of understanding or develop misconceptions about the nature of algorithms is an open question (Denning, 2017; Curzon et al., 2019). In schools, unplugged algorithms may be causing tension between views that an algorithm must be unambiguous versus often ambiguous unplugged activities that are for people to follow rather than a machine (Figure 1).

The influential Royal Society report preempting the change to the English curriculum defined algorithms, but for K-12 learners, with a further complexity of re-usability being required:

> *"Algorithms: re-usable procedures (often a sequence of steps) for getting something done. For example, plan the shortest delivery route for a lorry, given the required stops on the route."*(Furber et al., 2012, p.19)

In other K-5 teachers' material the context and level of precision was left open:

> *" An algorithm is a sequence of instructions or a set of rules to get something done."* (Berry et al., 2015)

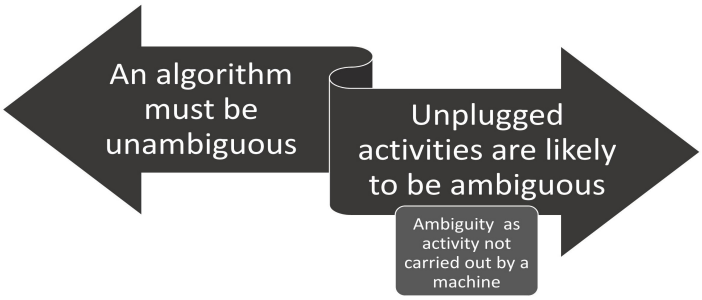


Figure 1: Algorithm precision versus unplugged activity ambiguity

What impact these differing definitions of an algorithm, with some times a focus on precision and other times precision left open, are having on K-5 teachers requires consideration. An aspect that may influence this is time, as a learner's understanding of a concept may change over their school career with simplification occurring at first which is then later addressed. As with maths methods in Section 2.2.2 learners may need to use informal personal representations before moving on to more formal and precise understanding. Moreover, as the design of algorithms is considered, further aspects come into play.

2.3. Representations of an algorithm

Moving on from the question 'What is an Algorithm?' we return to the relationship between algorithms and design including the representation of algorithms within design, the handling of potential ambiguous representations in research studies and research of how representations are refined during the design process.

Kant and Newell (1984) emphasised the importance of design, suggesting an automated design system could be built around their analysis of the human design process. In particular, they concluded that a significant property of such a system should be the progressive states of algorithms within design. They noted:

> *"Algorithm representations must be deliberately ambiguous in order to handle partial states of knowledge and assertions that have not been fully integrated during the design process."* (Kant and Newell, 1984, p.29)

This reference to a representation of an algorithm being ambiguous is important as it distinguishes between an algorithm and its representation during development. Even though a final representation of an algorithm must be unambiguous they make clear that when *developing* algorithms ambiguity is crucial. Managing this ambiguity is an important part of algorithmic thinking and the gradual refinement of the representations of algorithms and so programs.

At higher levels of study, specifying an algorithm through a logic or a formal reference language ensures precision in meaning (Cutts et al., 2014). However, as soon as non-formal instruction elements are introduced, such as using English to describe an

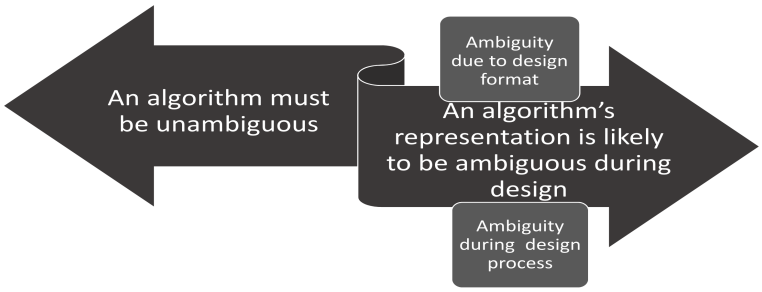


Figure 2: Algorithm precision versus ambiguity of algorithm representation

algorithm, then ambiguity results (Knuth, 1997, p.5). Therefore the format in which a design is developed, which includes the algorithm representation, may bring ambiguity.

Two influencing factors relate to why the representation of an algorithm may be ambiguous, firstly that as a problem is investigated partial states of knowledge and assertions are not yet integrated, and secondly that simply documenting an algorithm in non-formal formats is likely to result in ambiguity (Kant and Newell, 1984, p.29) (Knuth, 1997, p.5). These factors are seen in the development of mathematical methods with young learners as outlined in Section 2.2.2, as learners are exploring their understanding of a problem and using informal personal representations. Therefore, the difference between maths methods and computer science algorithms is not significant, as the maths method is a representation of the mathematical algorithm rather than the algorithm itself. This leads to the conjecture that, as young learners are exploring partial states of knowledge, making and changing their assertions and using informal methods then, as with primary maths, the representation of algorithms in the primary programming design processes is, by nature, ambiguous.

2.3.1. Handling ambiguous representations

There is a tension between algorithm precision when teaching what an algorithm is and ambiguity of algorithm representation during the design process (Figure 2). Two secondary school studies of learners aged 12 and over (Grades 6 onwards) explored the representation of algorithms during the classroom design process. Each dealt with potential ambiguity of representations in a different way. One carefully named the representation of the algorithm a *'verbal description'* and situated the study in an overall framework of levels of abstraction of an algorithm (Armoni, 2013). The second introduced flow charts, a more formal representation format of an algorithm, and mentions refinement of these but did not state how potential ambiguity was explained to learners (Rahimi et al., 2018).

Armoni's use of the term *'verbal description'* as part of her levels of abstraction framework carefully addressed the issue (Armoni, 2013, p.277). It is used to name the representation of an algorithm as learners refine their solutions. For example, there might be a *'verbal description'* of the general behaviour of a sprite which might be implemented as coded scripts, and then following further refinement a sub-level with more detail might be added, implemented as one or two command blocks in Scratch (Armoni, 2013, p.279).

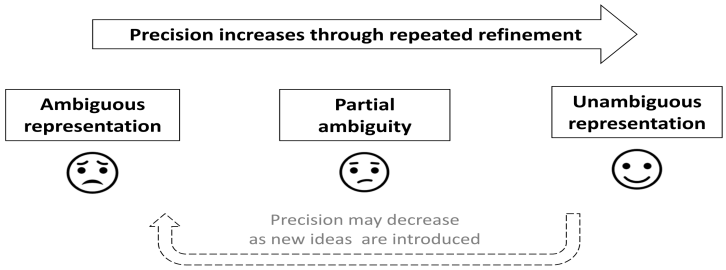


Figure 3: Design refinement process for K-5 programming projects.

Building on Armoni (2013)'s work, Statter and Armoni (2017) used *'verbal descriptions'* (as written notes) within the 'Armoni framework' over a two year period with grade 7 learners (n=80). Initial reports on the study indicate improved general computer science knowledge, knowledge of Scratch and ability to choose the correct level of abstraction when solving problems. The study refers to the 'Armoni framework' as being generally 'top-down' but with emphasis that learners are taught to return to and improve their algorithm as and when needed, the importance being the flexibility to traverse between levels of abstraction (Statter and Armoni, 2017).

Rahimi et al. (2018) presented an explicit development process, including algorithm creation, through their high school lesson plan materials. Rather than written verbal descriptions, flow charts provided a representation of the algorithm in the design phase. To implement solutions specially developed software incorporating PHP (a web authoring scripting language) and Python (a text-based programming language) was developed. Having interviewed four teachers who delivered the lessons and five focus groups of pupils, the authors gave a range of benefits for learners including improved algorithmic thinking, improved problem analysis devising algorithms and improved program development and also that, working on smaller problems promoted systematic and logical thinking. Whether the finally-produced flowcharts were unambiguous, called algorithms, representations of algorithms or some other term is not stated (Rahimi et al., 2018).

2.3.2. Fluent refinement of algorithm representation and the design refinement process

Rahimi et al. (2018) reported that a vague idea was translated to a more explicit picture through an instructional cycle incorporating a *'need to know'* learning of concepts within a project (Rahimi et al., 2018, p.68). In doing this, students learn about design whilst doing design. Similarly, Schulte et al. (2017) propose a design and exploration cycle where students are designing whilst exploring how to design. The authors discuss several ways of looking at the design process, including: the development of competencies versus outcomes; learning processes versus learning achievements; structure or mechanic perspective versus a function or intentional perspective. In this latter view, they suggest that fluency of translating between structure and function is one aspect of competency development. A theme emerges across these studies of projects being used as a context for learning to program, within which the representation of an algorithm becomes more precise through fluent refinements and the learner oscillates between implementation and design, as part of a learning-to-program process (Statter and Armoni, 2017; Rahimi

et al., 2018; Schulte et al., 2017). In the repeated refinement of their representation of their algorithm learners increase the precision of their ambiguous representation to an unambiguous representation As new ideas are added, precision may reduce. Therefore this is likely to be a cyclic process. Figure 3 provides a simple view to introduce the ambiguity of representations of an algorithm to teachers.

2.4. Our Previous Work

The survey here is part of a larger programme investigating the teaching and learning of programming to primary school pupils (Grades K-5). It builds upon a preliminary study in which we interviewed 5 teachers and 50 pupils to explore their use of design in programming projects and planning in other subjects (Waite et al., 2018a). In this earlier interview-based study we investigated common vocabulary used by teachers when talking about programming. We found there was confusion over what an algorithm was. The five teachers interviewed used a variety of conflicting terms for design, algorithm and code, and had a limited vocabulary to describe the code running. To investigate vocabulary use, we analysed teachers' responses to general questions about programming, counting their use of terms. During the interview, we were careful not to mention the term 'algorithm'. Finally, we provided teachers with images of design artefacts, including a storyboard concept map, labelled diagram and state diagram as well as code snippets. We asked teachers what terms they would use to describe the images. We then gave them labels, including the terms 'problem', 'design (including 'algorithm')', 'code' and 'execution'. We asked them to assign the labels to images. From this, discussion then followed on what each term meant. Teachers used the terms 'design', 'algorithm' and 'code' interchangeably, or were unsure of what an 'algorithm' might be for a programming project and several were shocked by the idea that an algorithm might be associated with the design. Progression in programming, as with any topic, is aligned to learners building understanding based on precise vocabulary (Armoni, 2013; Carlisle et al., 2000; Sapir, 1921) in a discourse rich environment(Cutts et al., 2012; Grover and Pea, 2013). Therefore, progression in programming may be being compromised by a lack of teachers' shared understanding of programming vocabulary and specifically the term 'algorithm' (Waite et al., 2018a) .

We developed the survey presented here to discover whether our findings on the confusion over the term 'algorithm' was seen in a wider population of teachers; explicitly including an activity selecting terms to describe code snippets and design artefacts. We reported preliminary results in Waite et al. (2018b) This covered teachers' responses to questions on quantitative data related to confidence to teach writing and programming; usefulness and use of planning and design; and 5 core specific uses of planning and design. As previously reported, from the current survey, 82% of teachers reported design in programming projects as being either 'Essential' or 'Very Useful' and yet just half of these teachers, 44%, converted this view of importance into action of using design in classroom 'Always' or 'Usually'. The contribution of the current paper is to explore why teachers are not introducing design, despite seeing it as important.

3. Method

We created an online survey combining quantitative and qualitative questions. Surveys are a low-cost method of eliciting information from a wide group of participants

where interviews are not practicable on a large scale (Cohen et al., 2011). We based our questions on our teacher interview study findings with a view to investigating whether teacher views were more widely held. Resources developed and data gathered from this study and the rest of this programme of research are available on the Project Website.

3.1. Participant recruitment

Participants were recruited from: a general survey advertised on social media, open to any teacher; and as a pre-course survey for three teacher professional development workshops. The workshops were a 1 hour introductory course on primary programming pedagogy in a local area; a 3 day course for experienced K-5 computing teachers for teachers across the UK; and a distance learning course on primary programming for any primary teacher Data was gathered from June 2017 to April 2018.

3.2. Survey design

Having obtained approval from the university's ethics panel, ethics procedures were followed throughout. Questions were trialled and improved based on several teachers' feedback. Survey questions are given in Appendix A. The survey predominantly compared teachers' uses of design and of planning. We also asked teachers about their perceptions of pupils' views on the uses of design, and the advantages and drawbacks of using designs. Teachers were also asked about their confidence to teach programming, their qualifications and training in programming and design and what resources and programming languages they used. In the survey, to investigate the use of the term algorithm by teachers with relation to programming projects, we included a set of images of code snippets and design artefacts and teachers were asked to select terms which they thought described these images.

Here, we report on sections of the survey related to teachers' reported difficulties of using design and selection of terms for code snippets and design artefacts.

.

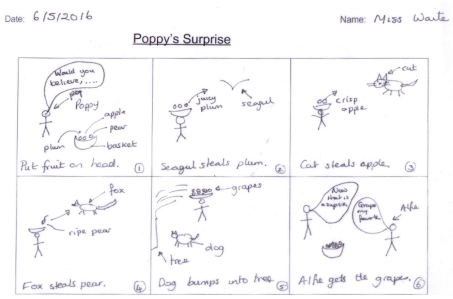


Figure 4: Image of storyboard for an animation presented to teachers in the survey. Teachers were asked to select terms to describe this design artefact.

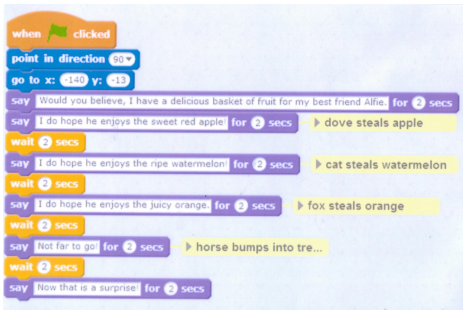


Figure 5: Image of a code snippet for the animation as presented to teachers in the survey. Teachers were asked to select terms to describe this code snippet.

.

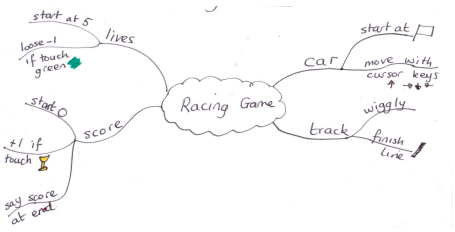


Figure 6: Image of concept map for a game as presented to teachers in the survey. Teachers were asked to select terms to describe this design artefact.

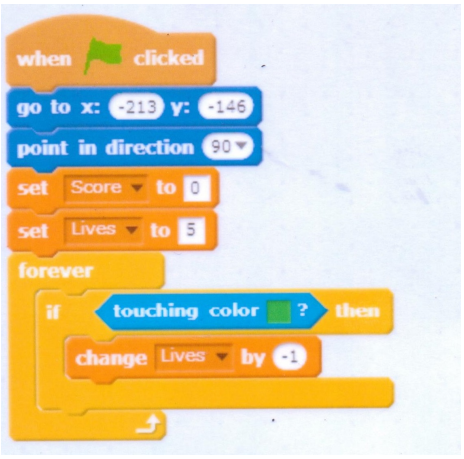


Figure 7: Image of code snippet for the game as presented to teachers in the survey. Teachers were asked to select terms to describe this code snippet.

3.2.1. Term selection for images of code snippets and design artefacts

To provide teachers with example project artefacts in the survey, two simple sample projects were designed and implemented by the lead researcher in Scratch. Scratch is a block-based programming language, commonly used in English K-5 classrooms. As shown in Figures: 4, 5, 6 and 7 we provided participants with four images of project artefacts. Two were associated with a simple animation project and two associated with a racing game project. For each project, there was one image of a design one of some of the code that implemented that design.

A storyboard design was shown for the animation (see Figure 4), and for the game a concept map design was shown (see Figure 6). The code snippet for the animation was simple, just a sequence of commands (see Figure 5). That for the game was more complicated, including variables, a repeat and a selection command (see Figure 7). Teachers were asked to select from a list of terms that they would use to describe the images; they could select more than one term and could add their own free text term.

3.2.2. Programming and Background sections of the survey

Teachers were asked about the uses of design in teaching programming in their classroom. Most of these questions had Likert Scale answers. All questions were mirrored as similar questions on the use of planning in teaching writing.

The final section included general questions on gender, ages taught and length of time teaching. Teachers were asked what drawbacks and advantages there were for the use of design in programming projects and their opinion of pupil's views on using design.

3.3. Qualitative data analysis approach

A thematic qualitative data analysis (QDA) of the free form question responses was conducted (in Nvivo) based on the methodology detailed by Kuckartz (2014). The ob-

jective was to reveal themes which might then be aligned to emergent data from the quantitative data (and vice versa). We first generated high-level categories deductively from the research questions (Kuckartz, 2014). We also created some sub-categories at this stage, based on knowledge of the field (Mayring, 2000). These high-level categories included drawbacks, pupil views, planning in writing, design in programming and sub-categories included task, design, algorithm, code, running the code. Using these initial categories, we coded each free text question adding and amending categories and sub-categories merging and splitting codes inductively (Kuckartz, 2014; Mayring, 2000). For example, pupil resistance to design and time needed to do design were mentioned frequently. The decision to combine themes into general summary categories or to split themes into more finely grained categories was subjective and based on emergent frequency of occurrence. Frequencies of each category are detailed in Table 2.

One researcher did the coding. As the question response data was held at the node level, case-wise inter-coder reliability could not be performed. Therefore a second coder coded two of the questions: firstly the teachers' reported disadvantages of design and secondly their views of pupil's opinion on design. These questions were chosen as they incorporated over half of the data reported on. Following the second coder's coding, the two researchers reviewed and discussed the inter-coder reliability reports to develop a consensus (Kuckartz, 2014, p74-75). There was minimal disagreement in category review and coding, but where this occurred, the two researchers discussed the differences and resolved them through changes to definitions or coding as necessary. A third researcher was available (but unused) to resolve any unresolved queries. The Cohen's Kappa reliability score was calculated as 0.705, which is considered as substantial strength of agreement between researchers (Landis and Koch, 1977, p165).

3.4. Quantitative data analysis approach

SPSS was used to analyse the quantitative questions. The Mann-Whitney U test was used to compare independent groups, such as male and female or generalist and specialist and the Kruskal-Wallis test was used to investigate differences across 3 or more independent groups, such as comparing scales of the degree of use of a programming language (Cohen et al., 2011). For both these tests, cross-tabulations were used to investigate any statistical significance and the effect size calculated. For the Kruskal-Wallis tests, we report the χ^2 statistic and have completed post-hoc analysis through the Bonferroni-correction for pairwise comparison of categories. To grade effect sizes, we used Cohen's classification: if r is 0.1 to 0.3 there is small effect, if 0.3 to 0.5 there is moderate effect, and 0.5 and above is large effect. For paired dependent testing the Wilcoxon Signed-Rank test was used (Meek et al., 2007). For this test, we show the number of cases tied and the changing 'up' or 'down' of ranked responses to report on the underlying data. Statistics and their interpretation was checked by an expert.

4. Results

We have divided the results from the survey into three main sections. First, we provide **participant data**, including gender, training, and resources used. Secondly, we summarise qualitative results from free text questions related to **teachers' negative views of design**. Thirdly we provide quantitative results from questions on the **selection of terms for images**. Participant data is presented as these were the variables

Table 1: Overview of participants, showing the % and number (n) of participants for each category

Gender	Male		Female		No Answer		Totals	
	%	n	%	n	%	n	%	n
Generalist	40%	40	60%	61	0%	0	49%	101
Specialist	38%	40	60%	64	2%	2	51%	106
Overall Totals	39%	80	60%	125	1%	2		207

against which the selection of terms for images was compared and for which we found statistically significant differences.

4.1. Participant data

Participant data reported here includes gender, generalist or specialist teacher, confidence to teach programming, training on programming and design, resources used for teaching programming, programming languages taught and ages of pupils taught. This participant data is presented as these were the variables against which the selection of terms for images was compared. Statistically significant differences were found for all variables which we compared against, except one, confidence to teach programming.

The only participant data presented here that has been reported elsewhere is that on overall numbers, gender and generalist or specialist (see Table 1) and confidence to teach programming (see Section 4.1.2).

4.1.1. Gender, Ages taught and Specialist or Generalist

207 participants completed the survey. Females (60%) were the most represented gender (Table 1). Those teachers who taught both programming and writing we call *generalists*, Those who taught programming but not writing we call *specialists*. 49% of teachers were generalists and 51% specialists with a broadly 40:60 split of male:female for both groups. As shown in Appendix B Table B5, 54% taught grades K-1 (KS1), 87% grades 2 to 5 (KS2). Overall 92% (n=190) taught grades K-1 or 2-5 (K-5).

4.1.2. Confidence to teach programming

27% of teachers reported being very confident to teach programming, 27% quite confident, 25% a little confident, 18% not very confident and 4% not at all confident.

4.1.3. Training

Participants were asked 'What training have you had related to programming?' They could select more than one response and they could add a free form other answer. Responses are shown in Appendix B Table B1. Three quarters of teachers (73%, n=150) reported having taught themselves, through reading books and on the job learning, half of the teachers had had training through courses. Only 6% (n=13) had received training during their teacher training but around a quarter (23%,n=49) reported some form of prior education in programming.

They were also asked 'Have you had any specific training on using design in programming?' with a free text response. The responses are summarized in Appendix B Table B2. The spread of design training by gender was relatively even with 26% (n=21) males,

and 22% (n=28) of females having had training in design, for generalists and specialists there was a greater percentage of specialists but this was not statistically significantly different at 27.9% (n=29) specialists and 19.8% (n=20) generalists.

4.1.4. Programming curriculum materials

Participants were asked a free form text question, "What planning do you use for teaching programming? For example, we follow Somerset Grid for Learning plans, we make up our own and use Barefoot material." 84% (n=173) of teachers answered the question providing 418 distinct answers which we grouped into 80 summary groups. 71 of which were mentioned by less than 8 teachers, with 28% (n=48) having only 1 teacher referencing using it. The responses for those answers where more than 4% (n=8) of the teachers responded for that item are summarized in Appendix B Table B3.The full list of other reported curriculum products is available on the Project Website.

4.1.5. Programming languages used

Teachers were asked 'What programming language do you use for teaching programming. For example, We use ...' with a list of possible answers, and a free text option. Teachers could respond, 'Mostly', 'Sometimes' or not at all. Appendix B Table B4 gives responses for those languages reported by more than 10% of participants. Scratch (97% of participants) was the most popular language, followed by Scratch Jnr (59%).

4.2. Teacher's negative views of design

Qualitative data is now presented on the participants' negative views on design. Teachers were asked a number of free text questions which gave them the opportunity to reveal their thinking including:

- What DRAWBACKS (if any) are there in asking pupils to use a design in a programming project?
- What do you think pupils think about being asked to use a design in a programming project?
- Do you have any other comments about design in programming or planning in writing?

156 teachers reported drawbacks of using design. 18 themes emerged from these comments. 126 teachers reported on negative pupil views. 9 themes emerged from these comments. Themes and numbers of coded segments from both question are presented in Table 2. 6 themes were common across teacher reported disadvantages and teacher reported negative pupil views. Themes were reviewed and 5 overarching reporting groups became evident: pupil resistance, pupil expertise, time constraints, pedagogy choices and teacher expertise. We consider each of the reporting groups in turn and present illustrative teacher comment segments.

Table 2: Themes for questions on drawbacks, pupil negative views giving number of coded segments within reporting groups

Reporting Group	Theme	Drawbacks	Pupil Views	Theme Total
Pupil Resistance	Pupils want to get on and code	36	52	88
	Pupils think design is boring or waste of time	5	58	63
	Pupils rush design	0	2	2
	Pupils don't want to focus	0	2	2
	Pupils don't like making mistakes	2	0	2
				157
Pupil Expertise	Lack of pupil knowledge	42	2	44
	Design created is too ambitious	10	0	10
	Pupils stick to design and won't change it	10	0	10
	Pupils don't use design	7	0	7
	Level of detail is wrong on design	4	1	5
	Design limited by pupil's literacy	4	0	4
	Pupils don't link design to coding process	2	0	2
				82
Time Constraints	Not enough time, design takes too long	43	5	48
Pedagogy Choices	Pupils learn as they go along	15	2	17
	Pupils need something to start with (a model)	5	0	5
	Design limits creativity	3	2	5
	Some lesson resources don't have design	2	0	2
	Teacher prefers pupils to plan on computer as they code	2	0	2
	Teacher prefers verbal planning	1	0	1
				32
Teacher Expertise	Teacher confidence or training	13	0	13

4.2.1. Pupil resistance to design

Our most cited theme for teachers' opinion of negative pupil views was grouped under pupil resistance to design and included responses that: pupils want to get on and code, or think design is boring, or a waste of time. This group also includes the themes of pupils not linking design to the coding process, pupils rushing design, not wanting to focus and not liking making mistakes. Despite this negative view, many teachers gave a balanced view saying some pupils found design useful and that pupils' views changed over time. Teachers mentioned instructional techniques (such as modeling) to support pupils to see the value of design.

Teacher 207, relayed a stark view that pupils did not associate design with computing:

> *"A minority find it useful but I think most find it unnecessary and a chore. They don't link it to computing."* [Teacher 207]

However, Teacher 19, explained how he waited for learners to realise they needed help and then introduced design:

> *"Some consider it [design] as an extra task or unnecessary burden, so they prefer to start their project without any visible design or plan. But when they are confused along the way and the mental picture of the project in their minds or imagination is not very clear, as it was in the beginning of the project, they seek for assistance. Then I will ask for their design or plan. If there is none, I ask them to go and produce a sketch which I will later help to go through and refine the design and ask them to follow the design and rectify any deviation from the plan."* [Teacher 19]

One teacher raised the importance of pupils being helped to reveal their thinking not only for themselves, but also for her to understand:

> *"Some complain [about design], I don't know what they think as I cannot see inside their heads, but almost all of them admit after a while the need for use of designs and plans."* [Teacher 52]

4.2.2. Lack of pupil expertise

The second most common disadvantage theme was around pupils' lack of knowledge or skills. Teachers mentioned that pupils did not know how to implement their ideas, not without exploring at the same time, or did not know what was 'doable', or what level of detail was most effective. Teachers gave differing views on how pupils used design. Some commented on how less confident pupils stick too rigidly to design and this impacts creativity. Another issue was that initial designs became outdated and don't use designs. Teachers mentioned pupils' literacy levels constraining their ability to design and pupils not linking design to the coding process.

Teacher 45, succinctly wrote:

> *"Their ideas can be limited by the programming language and their understanding of it."* [Teacher 45]

suggesting that pupils could not remember what code might do, Teacher 6 reported:

> *"It can be frustrating for some that find it difficult to imagine what will happen without seeing it as they go and some just want to 'get on with it' but it [design] actually gives a better success rate and helps with the frustration of debugging."* [Teacher 6]

Teacher 204 also highlighted the problem that learners don't know what is 'doable':

> *"They are limited in their thoughts however the capabilities of the programming software sometimes hinders what can be achieved over the time frame. e.g. Student plans a 10 minute animation, student wants her app to look like another site or to have advance features"* [Teacher 204]

Teacher 3 mentioned that pupils were not comfortable with 'failing' and also highlighted the ways that pupils of different ability used design:

> *"They need a starting point so I give a basic program say, in Scratch, and they develop their ideas from there. This can mean ideas are restricted in that they reflect the basic idea rather than perhaps focusing on what part interests them and developing that. More able children will try this but many like to stay with tried and tested. Encouraging children to not worry about failing has been hard."* [Teacher 3]

4.2.3. Lack of time to do design

The most commonly cited single drawback for using design in teaching programming was the lack of time to do design. Teachers reported the lack of time in the school schedule for undertaking design, implying design took too long. In these comments the level of detail was sometimes mentioned as pupils included 'too much detail' in their designs. Comments often linked to pupils needing time to explore what was possible, and alluded to creativity, as pupils expanded their design and found 'more interesting ways' to implement their design.

Teacher 206 shared her view of the drawbacks of design writing:

> *"[I] Can see they [designs] are important even though I tend only to do verbal planning - largely due to timetable constraints. Sometimes [I] find pupil's ideas evolve in more interesting ways as they discover more code and capabilities, which can be limited by simplistic designs that they come up with at first"* [Teacher 206]

Teacher 182 wrote of the disadvantages of design:

> *"We often run out of time and it feels rushed"* [Teacher 182]

In what pupils thought of design, Teacher 185 recorded:

> *"[Design] Delays them [from] getting on with things. Slows creativity."* [Teacher 185]

4.2.4. Conflicting pedagogical approaches and a lack of suitable resources

In answering the disadvantage question a theme around conflicting pedagogical approaches was raised. These approaches included: 'just in time designing' (designing as they went along) versus 'exploration first and then design' versus 'design first'. Several teachers also mentioned not having teaching resources that included design elements. Several teachers also commented on design restricting creativity and them preferring pupils to plan verbally or as they coded.

Teacher 202 noted:

> *"Some students struggle to design something they have never seen before. They find it easier to experiment with the actual creation."* [Teacher 202]

Teacher 207 brought together the impact on design of exploration and time constraints as well as raising pupil's attitudes to making mistakes:

> *"[Design is] Not as liberating as just trying things out in programming software; children are more wary of making a mistake. It can be time consuming and adequate time needs to be invested to ensure that it [design] is done properly. It is difficult to find this time as we have a very busy timetable and one hour allocated for teaching computing each week. "* [Teacher 207]

Teacher 3, brought to life Turkle and Papert's 1990 *'bricoleur'* and *'planner'* ideas:

> *"Some like the organisation and ordered approach but some prefer to jump in, try out ideas, select what works then plan. It does help them think about why they are programming as well as what they want to happen, especially thinking about how they order and sequence."* [Teacher 3]

One teacher implied that the resources she used did not include a design phase:

> *"I would make them plan always for programming tasks if I wasn't following the scratch Maths SoW [scheme of work]."* [Teacher 36]

Another wrote that teachers with low confidence might not use design as they were following resources that did not include design:

> *"Teacher's aren't always very confident, so they stick to very structured coding plans rather than more open-ended that would allow for pupil design"* [Teacher 155]

4.2.5. Lack of teacher expertise

Teachers mentioned a lack of their own expertise including a lack of confidence to use design or a lack of training on design. Teacher 35, for example, wrote :

> *"I don't think I have enough instruction myself on how best to create authentic, meaningful reasons to design a project and how to support and guide student lead interest driven project design."* [Teacher 35]

Teacher 10, related confidence to teach design, as pupils might be over ambitious:

> *"[I] Don't feel confident teaching them [design], they [pupils' designs] might be overambitious and unrealistic."* [Teacher 10]

Another teacher responded implying a doubt on the entire purpose of design:

> *"Training to understand the purpose of planning"* [Teacher 103]

Table 3: Descriptive statistics of teacher's selection of terms for each image, showing the % and number (n) of teachers who selected a term, e.g. 54% of teachers selected the term 'Algorithm' for Image 1

Term	Image 1 Simple Code	Image 2 Complex Code	Image 3 Concept Map	Image 4 Story Board
Algorithm	54%(111)	58% (120)	30% (62)	35%(73)
Program	50% (103)	47% (97)	8% (17)	5%(11)
Code	56%(115)	57% (118)	4% (9)	3%(9)
Script	58% (120)	41% (84)	6% (12)	22% (46)
Design	5% (11)	3% (7)	65%(135)	56% (115)
Plan	9% (18)	0	78% (161)	72%(148)
No choice made	1 (2)	2% (4)	.5% (1)	2% (4)
Program, Code or Script	89% (185)	82% (170)	16%(32)	27%(55)
Design or Plan	12% (24)	4% (9)	92%(190)	84%(173)

Table 4: Selection of the term 'algorithm' for either code snippets or either design artefacts

	Selected a design artefact	Did not select a design artefact
Selected a code artefact	31% (n=65)	39% (n=80)
Did not select a code artefact	16% (n=33)	14% (n=29)

4.3. Selection of terms for images

In our earlier interview-based study, we had found a general confusion over vocabulary used to describe programming artefacts. We asked teachers in the survey to select terms to describe images of code snippets and design artefacts. We now provide quantitative results of teachers' selection of terms to describe these images. For each image (see Section 3.2.1), we asked: 'Which of these words would you use to describe this image? (You can choose more than one word if you like.) It is a picture of ...' The terms provided were popular terms from our first study and included: 'A program'; 'Some code'; 'A plan'; 'An algorithm'; 'A script', 'A design' and 'Other' where teachers could enter free text.

4.3.1. Descriptive statistics

Over 80% of teachers selected the terms 'code', 'program' or 'script' for the code snippet images and similarly 'design' or 'plan' for the design artefacts (Table 3). The term 'algorithm' was selected by more than half the teachers for the code snippets, but only around a third of the time for the design artefacts. Nearly twice as many teachers selected the term 'algorithm' for complex code snippets than the concept map.

4.3.2. Comparing the selection of the term 'algorithm' across images

14% did not select the term 'algorithm' for any image, 31% called at least one of the code snippets an 'algorithm' and at least one of the design artefacts an 'algorithm',

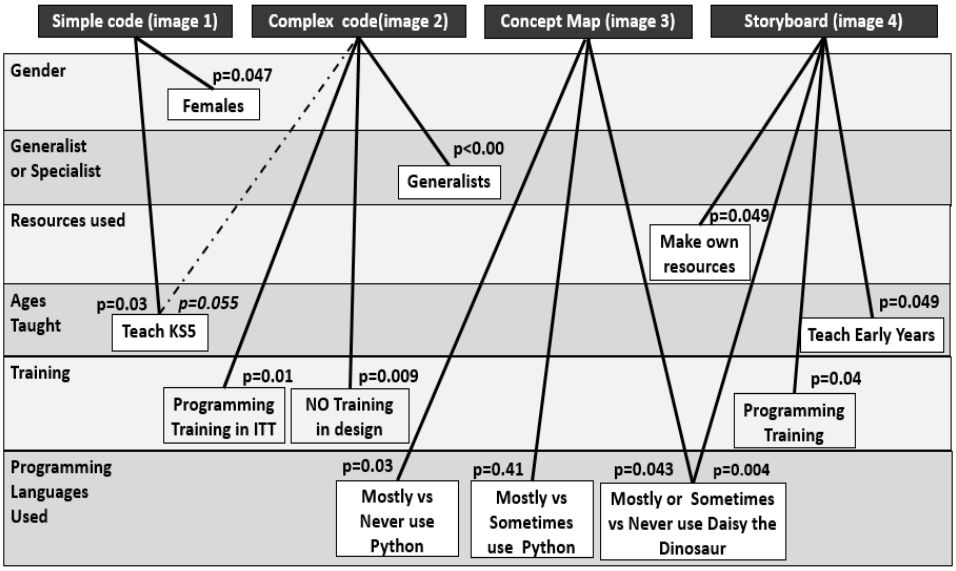


Figure 8: Groups statistically more likely to select the term 'algorithm' for an image.

39% only called one or both code snippets an 'algorithm' and 16% only called a design artefact image an 'algorithm' (see Table 4).

4.3.3. Independent pair and group tests

To investigate whether all teachers, irrespective of gender, confidence, training and teaching setting had the same view of the term 'algorithm' in the image activity, test statistics were used to determine whether there was any statistically significant differences for each of the variables gathered about teachers in the survey of: male and female; generalist and specialist; confidence to teach programming; training in programming; training in design; curriculum resources (used by more than 10% of participants); programming languages (taught by more than 10% of participants) and age of pupils taught. The significant differences are summarized in Figure 8 and described in detail in Appendix C as a narrative descriptions and in Table C1. For each of the variables tested, at least one statistically significant result was found for one or more images, except for confidence to teach programming. The level of confidence reported by teachers appeared to make no difference to the selection of the term 'algorithm' for the four images.

5. Discussion

We first consider possible biases then discuss the two research sub-questions, and the overall research question of **RQ1: What are K-5 teachers' difficulties in using design?** We then provide an overall critical analysis of our most significant findings in light of recent literature to inform thinking beyond the current context.

5.1. Participants and Survey: possible biases

We saw a high percentage of specialist teachers (51%). In the 2017 survey for , The Royal Society Review of Computing in UK schools, mention was made of specialist primary teachers being helpful in school (The Royal Society, 2017b, p.75). However, the percentage of such teachers across schools was not reported. Anecdotally, there may be a trend of specialist primary computing teachers. But it is likely that as with a recent review of computing education in England, that more expert teachers in primary computing are engaging with research activities (The Royal Society, 2017a, p.55). We, therefore, note that there is a potential bias in our population. However, the high percentage of specialists provides us with an opportunity to compare generalists against specialists and investigate this emerging group of expert teachers. Further, if these more expert teachers are confused regarding what is an 'algorithm', or they are having difficulties with design, then it is unlikely that teachers with less expertise will be fairing any better.

5.2. RQ1-1: What are teacher's views of the difficulties of using design?

Summarised in Table 2, five common reporting groups were found across teacher's reported drawbacks of the use of design and their views of pupil's negative opinions.

Firstly, **pupil's resistance** to using design was the most cited difficulty across drawbacks and pupil's views, making up nearly half of the comments. Teachers reported pupils wanting to get on and code, thinking design was boring or a waste of time and pupils not associating design with coding. Yet, the Computing curriculum in England has required K-5 teachers to teach pupils to use design in programming (Department for Education, 2013a) since September 2014.

In research in Physical Education in high schools, Cothran and Ennis (1997) found that pupil resistance to new subject content and pupil expectation about what lessons in Physical Education should be like caused conflict with their teachers. This resulted in some teachers not teaching aspects of the new Physical Education curriculum. These authors reported a successful strategy to support the introduction of changes was teachers enabling pupils to understand why they were being taught new things in new ways.

Pupil's prior experiences and expectations of computing lessons (and before that ICT) may be having a similar impact in classrooms for K-5 pupils as with high school Physical Education in the 1990's. Pupil's in English K-5 computing lessons may have become accustomed to using computers straight away, and not having been required to consider design. Those teachers in our survey who responded that they were overtly introducing design into their classrooms indicated they were encountering pupil resistance, but that they were overcoming this by a range of approaches including helping pupil's see the value of design, a similar approach as suggested by Cothran and Ennis (1997).

Secondly, **time** (or lack of it), was the most cited specific drawback to using design, with teachers explaining that it took time to teach pupils how to design and time to do design in class. In England most primary schools allocate 1 hour a week to computing (The Royal Society, 2017b, p.6) and within that time must teach all aspects of the computing curriculum, including information technology (using word processors, presentation products, data handling software, film-making, finding things out using digital tools etc.) as well as digital literacy (staying safe online, responsible use and ethics etc.)

There is probably little that can be done to provide more time in the busy primary timetable for computing on its own. However, teachers could be provided with programming activities which embed design in them and designs created in other curriculum subjects activities could be reused in computing lessons. For example, if a storyboard has been created for a writing activity, an animation could be produced from it in computing. Similarly, a concept map used in history could be used as a starting point for making a history game in computing. However, careful attention would need to be made of progression across both subjects. Another option is to reduce the introduction of unfamiliar design formats and use those already used by the pupils whilst making it clear to them that they are transferring their existing expertise from one subject to another.

A further drawback related by teachers was on their **lack of expertise in teaching design**. This correlates with our quantitative data where we found less than a quarter of teachers reported any training on design and nearly three-quarters of teachers reported they had learned about teaching programming through personal study (see Section 4.1.3).

This apparent lack of consistent training may be compounded by the resources that teachers have available to them; nearly half of our teachers reported creating their own resources. From the first author's experience of resources, within Barefoot, code-it, Switched on and Wessex resources, design is incorporated in planning, just over half of our teachers reported using one or more of these products (see Section 4.1.4). Linking to our investigation of the term 'algorithm' and analysis of the differences of teacher responses based on resources used, we found no statistically significant difference for teachers using the term algorithm for these different resources, only if they created their own materials (see Figure 8). Further work is needed to review resources more formally to ascertain the degree to which design is included and how.

Fourthly, a quarter of negative and disadvantage comments were that pupils had **limited expertise** of using design. This will be an ongoing issue until design has been introduced earlier in pupils' school careers and the other difficulties have been overcome to enable teachers to teach design.

Finally, tensions related to **pedagogical choices** were reported by teachers as being a difficulty. For some pupils wanting to **learn as they go along** was a drawback associated with using design. From the comments of some teachers, there is indication they are using a *bricoleur* design approach with pupils preferring to design at the point of coding as a 'thought only' design and designs becoming a 'verbal' design if pupils discuss their designs with others or if they are questioned by teachers. Some teachers appear to be mitigating the view that design is seen as a burden through careful teaching of the benefits of more overt forms of drawn or written design. However, how widely adopted this technique is employed has yet to be evidenced. How common these practices might be was not clear from our survey responses and nor is the range of pedagogical choices available to them or the merit of these.

5.3. RQ1-2: Do teachers demonstrate a shared understanding of the term 'algorithm' when viewing images of code snippets and design artefacts?

Teachers did not demonstrate a shared understanding of the term 'algorithm' when viewing images of code snippets and design artefacts. Reflecting first on what common terms were used to describe the images. As shown in Table 3, for the code snippets, the majority of teachers used the terms 'some code', 'program' or 'script' for both the simple code snippet (89%, n=185) and the more complex code snippet (82%, n=170). The

majority of teachers used the term 'design' or 'plan' for the concept map (92%, n=190) and the storyboard (84%, n=173). This selection of terms program, code, design and plan seem unsurprising. However, that a fifth of teachers selected the term 'script' for the storyboard is interesting. This may be because storyboards can be used in writing and the term 'script' is used in relation to 'scripting drama' in the English Primary National Curriculum for the teaching of English (Department for Education, 2013c, p.4). We do not further discuss the potential confusion of this term here, as our focus here is on the use of 'algorithm'.

When selecting the term 'algorithm' for images, responses were complex. As shown in Table 3, the term 'algorithm' was used by just over half the teachers to describe either of the code snippets and around a third of teachers called the concept map an 'algorithm' and a third the storyboard an 'algorithm'.

Selections for the term 'algorithm' for one or both code snippets or one of both design artefacts shows a wide variety of views (Table 4). We look at each combination through the lenses of the concept of a representation of an algorithm (see Section 2.2) and the design refinement process of increasing precision from ambiguous to unambiguous algorithm representations (see Section 2.3.2):

- 31% of teachers selected the term 'algorithm' for both a code and a storyboard or concept map. We do not know whether these teachers understood that the code implements the algorithm and that an ambiguous representation of the algorithm was provided by the design artefacts.

- 39% of teachers called code snippets 'algorithms' and did not do the same for storyboard and concept map images. We do not know whether these teachers thought the lack of precision of the representations of the algorithm in the storyboard and concept map was such that it was arguable to call these artefacts an 'algorithm', but instead an ambiguous representations of the algorithm and that code was an implementation an algorithm.

- 14% of teachers did not select the term 'algorithm' for any image. What these teachers think might be the algorithm for a design needs to be uncovered.

- 16% of teachers selected the term 'algorithm' for one or both of the design artefacts and not for the code snippets. Whether these teachers understood that the code is a representation of the algorithm at a different level of abstraction to the design and that the design is a representation of the algorithm, again is not clear.

Overall though, the main point is clear. The teachers did not share an understanding of what should or could be called an algorithm in a design context. Using a more careful vocabulary to describe representations of algorithms and the degree of precision of these representations may help teachers to come to a higher degree of consensus. However, evidence of this is needed.

The vast majority of comparisons to investigate sub-group selection of the term 'algorithm' were not statistically significantly different. Where there was a difference, as shown in Figure 8, many were with a small effect size, or with p-value very close to the accepted threshold for indicating significance (.05) and some of the differences are very difficult to suggest an explanation for. Therefore, despite a relatively large population of 207 teachers, the variances may be random, as one would expect one in twenty tests to

be near 0.05 by chance. However, the fact that for all variables, except one, there was at least one statistically significant difference is a suspicious pattern. We therefore discuss each variable in turn.

5.3.1. Gender

Gender impacted on just the simple code image, with small effect size. Females were more likely to select 'algorithm' than males. Why is hard to fathom. Confidence in teaching programming is the only other difference we have related to gender, with males more confident to teach programming than females (Waite et al., 2018b) . However, there was no statistically significant difference for the selection of terms for any image by confidence.

5.3.2. Generalists or Specialists

Generalists were more likely to select the term 'algorithm' for the more complex code snippet, again with small effect size. There was no indication in the free text data that gave any underlying rationale for the change in these generalist teachers in calling more complex code an 'algorithm'.

5.3.3. Resources used

Irrespective of whether a teacher used the most popular three named resources: Barefoot, code-it, or Switched on computing there was no statistically significant difference to the selection of terms. Only one curriculum resource response provided a significant difference in use of the 'algorithm' term being selected. Those who created their own resources were more likely to allocate the term 'algorithm' to the storyboard image. Teachers may be gleaning ideas from multiple sources and finding a common theme of storyboards having a series of steps and associating this with algorithms. They are also described in primary material as being a series of steps, as discussed in Section 5.3.5.

5.3.4. Ages Taught - KS5

The age of pupils taught by teachers had an impact on several images. The term 'algorithm' was selected more by Key Stage 5 teachers with the code artefacts than the design artefacts, particularly for the simple code snippet. However, the number of teachers of this age group in our population was small (n=9). In the English computer science A-level specifications, which KS5 teachers use, the term algorithm is not used synonymously with coding (Department for Education, 2014). However, teachers may have interpreted the question knowing that the code is a representation of an algorithm and that the designs were not sufficiently precise to apply the term 'algorithm'. However, three of the nine selected 'algorithm' for the concept map, and two for the storyboard.

5.3.5. Ages Taught and Programming Languages - Early Years and Daisy the Dinasour

Early years teachers and those who taught with Daisy the Dinosaur were more likely to call the storyboard an 'algorithm'. Linking this association to a different thread of questions there may be a connection with teachers' pedagogical content knowledge in other subjects (Shulman, 1986). Teachers were asked about their use of storyboards in both writing and programming. 100% of early years generalist teachers (n=15) used storyboards in writing 'Always', 'Usually' or 'Sometimes'. For generalist teachers of older

pupils, though still common, storyboards were not as popular for writing with 64% using them 'Always', 'Usually' or 'Sometimes'. This difference in use of storyboards in writing was not carried over to the reported use of storyboards in design. In using storyboards in design, 70% (n=35) of early years teachers (generalist and specialist) used storyboards 'Always', 'Usually' or 'Sometimes' with a similar percentage for older learners. 69% of teachers of older pupils used them 'Always', 'Usually' or 'Sometimes'.

Therefore, the difference in use of storyboards is not in teaching programming but in teaching writing. Whether this impacts on the association of the term 'algorithm' and storyboard, or whether the programming languages used and associated resources are having an influence is not clear. However, the use of these common classroom planning tools in other subjects may be having an impact on teacher understanding.

Early Years teachers were more likely to use the simple block-based programming language particularly aimed at route based projects called Daisy The Dinosaur than any other teacher age group (p=.001, n= 207, Z=3.179, r= .0.221) with small effect size.

Several online resources [2] [3] [4] and primary school lesson plans[5] [6] suggest that Daisy the Dinosaur be used with storyboards as algorithms to create animations, including Barefoot, our most highly cited resource. 'Algorithm' appears to have become associated with storyboards for this age group, Early Years, using this type of programming language, Daisy the Dinosaur, a block-based app which is particularly suited for route-based activities. A more careful review of resources, using a standard framework, would further clarify whether resources are driving teachers' understanding of design.

5.3.6. Programming Languages used

The use of Daisy the Dinosaur also impacted the likelihood of selecting 'algorithm' for the concept map. We have not found a Daisy the Dinosaur resource that uses concept maps. However, some of the most popular primary resources (see Section 4.1) are subscription-based, and there may be material which we have no access to.

A possible link here could be the use of concept maps as a design type. In asking teachers about their use of concept maps in writing and design, we found that teachers of older learners were twice as likely to 'Always', 'Usually' or 'Sometimes' use concept maps (42%) than early years teachers (20%). But for design, the percentage of use was similar across both age groups of teachers at 60% and 63% respectively. Therefore concept maps, unlike storyboards, are being used more frequently in design for programming than planning for writing.

Teachers who used Python 'Mostly' or 'Sometimes' compared to 'Never' were more likely to call the concept map an 'algorithm'. We did not find any specific resources that linked Python with concept maps, but this may be because such resources are part of a paid for subscription service, or this finding may be random or due to an entirely different underlying motive.

[2] https://bridgetspgce.wordpress.com/2016/09/13/daisy-the-dinosaur/

[3] https://primarycomputingcurriculum.files.wordpress.com/2013/02/teaching-programming-april-2013.pdf

[4] https://barefootcas.org.uk/wp-content/uploads/2014/10/Sequence-concept-barefoot-computing.pdf

[5] https://www.elmwoodps.co.uk/attachments/download.asp?file=207&type=docx

[6] http://www.chilton.suffolk.sch.uk/storage/download/AwW3Dx9ehD

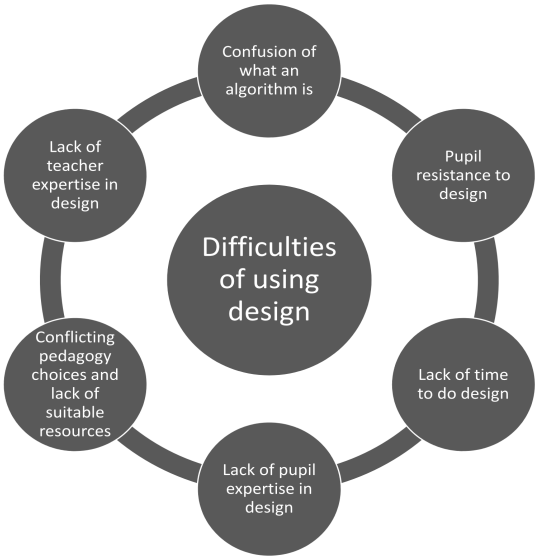


Figure 9: Difficulties of using design.

5.3.7. Training

Training influenced the selection of terms for the complex code image and the storyboard. Teachers with programming training during their initial teacher training were more likely to call the complex code snippet an 'algorithm', as were teachers with NO training in program design. This result seems to indicate an opposite impact of two sources of training that theoretically should result in the same outcome. There is clearly an issue here with consistency across provision. Why the complex code would be any more or less an 'algorithm' than the simple code is not apparent.

Having had any kind of programming training increased the chance of selecting 'algorithm' for the storyboard. This may be a positive outcome of training, perhaps indicating that programming training is supporting teachers to understand that a storyboard represents a sequence of instructions and that this ordering of events or annotation of rules provides an algorithm, albeit a very simple one.

As shown in Figure 8 gender, specialism in teaching programming, training received by teachers, programming languages used, resources used and age groups taught all appear in some way to contribute to whether teachers will select the term 'algorithm' to describe images of code snippets and design artefacts of a storyboard and a concept map. The variation in use of terms, was sometimes in opposite and unexpected ways, suggesting that there is much confusion in the teaching community of what an algorithm is in K-5 programming projects and that teachers need support to improve their understanding.

5.4. RQ1: What are K-5 teacher's difficulties in using design?

A problematic landscape for teachers to navigate is revealed by bringing together our quantitative analysis on teacher's selection of the term 'algorithm' to describe code

snippets and design artefacts with our qualitative analysis of teachers' free text views on difficulties with design, as shown in Figure 9,

We have revealed two sources of contributory factors to difficulties that teachers are having when teaching design in K-5 programming projects. First, there is a lack of a common understanding of the foundational term algorithm including confusion about what an algorithm is and how it relates to design artefacts. Secondly, teachers identified a series of barriers including resistant pupils, limited time and a lack of expertise for both pupils and teachers in using design in programming.

In light of the this, we return to our discussion of the notion of representations of algorithms from Section 2.3 and the handling of ambiguity through a design refinement process (Section 2.3.1). Within the storyboard animation design artefact we presented to teachers (see Figure 4), the algorithm is represented as the ordered images and annotated notes describing the action required. This representation of the algorithm is clearly ambiguous and similar to pupil's personal representations of maths methods as discussed in Section 2.2.2. It requires the pupil to tell us more about what they intend. Therefore the storyboard per se, cannot be called an algorithm.

Considering the concept map design artefact (see Figure 6), representations of the algorithms required for the game project are presented through annotations such as the written rules for managing player movement, maintaining the score and managing the end of the game. However, the representation, as with the ordered images of the storyboard is ambiguous and again could not simply be called the algorithm. Further design decisions are needed and the format of the representation limits precision.

Drawing in the results from the difficulties with design, teachers are conflicted by the pedagogical choices they need to make, with learners needing to explore and develop their expertise as they go along. This supports our discussion in Section 2.3.2 that there is a fluent refinement of the algorithm representation during the design refinement process whereby learners find out what is possible and make design decisions which lead to further precision of their emerging solution.

No wonder then, that our simple question of what term would you use to describe these images has elicited such varying combinations of responses in the use of the term algorithm. To give a full response teachers would need to articulate the nuances of representation, implementation of algorithms and ambiguity of representations. Teachers were given the opportunity in the survey to expand upon the term selected, but no teacher raised these themes. This again is not a surprise, as outlined in Section 2.2, the concept of a representation of an algorithm does not feature in advice provided to teachers. Rather, a simple view that an algorithm is a set of precise instructions or rules, with no mention of what to call the intermediate state of this design component.

In our discussion of these contributing factors, the lack of underlying models on algorithms and design in programming projects for K-5 pupils has become apparent. Models are needed to support discussion with teachers to review and develop the use of design in K-5 classrooms. We therefore suggest that the notion of representations of algorithms be introduced to teachers along with the process of refinement of the precision of representations from ambiguous to unambiguous (see Figure 3). This will support teachers improving their understanding of algorithms and start to overcome their difficulties with design.

5.5. Implications and suggestions for practitioners and educational designers

Our study suggests that program design has been overlooked in the training of English primary computing teachers and is having an impact on the ability of these teachers to deliver this component of the curriculum. Therefore, in England, remedial action is needed in the creation and delivery of program design professional development.

Teachers report a lack of programming resources which include design. Therefore, new lesson planning material should include design, and existing teaching material reviewed and updated to add this important element. However, in doing this, careful consideration needs to be made of adding more to lessons, on the time needed to complete projects. Time could be saved by reusing designs created in other subject activities, or by reusing familiar design formats (again from other subjects). However, we suggest that as teachers become more aware of the potential benefits of design to pupil understanding and pupil independence, then they will see extra design time as time well spent.

Notwithstanding improvements in professional development and lesson plan resources, a pupil and teacher habit of coding without designing has developed, and this will be difficult to overcome. Only when teachers have the confidence, expertise and resources required to incorporate design in their teaching, is it likely that they and their pupils will be persuaded of its value. Therefore, lesson plan materials and teacher CPD should address this negative view of design and support teachers to overcome this such as incorporating activities which highlight the value of design and by exemplifying the benefits of design. However, exactly what underlying concepts relate to design and what progression looks like in design in an open question. Similarly, whether the difficulties encountered by teachers in England is seen in a broader context must be considered.

5.6. Critical analysis of findings within a broader context

5.6.1. Uniqueness

Our study is unique in that our K-5 teachers have been delivering a *radical implementation of computing* since 2013 (Department for Education, 2013a). Indeed England has introduced "perhaps the most radical change of all" (Vahrenhold et al. (2019) p. 547) in its replacement of a predominantly K-8 Digital Literacy computing provision to a predominantly Programming and Computer Science Curriculum. With limited recent, robust and relevant computer science education research to help them (Waite, 2017), as early adopters, our teachers have trail-blazed a difficult path. In a 2017 review of progress implementing classroom computing changes in the UK, the provision for learners was found to be fragile and patchy (The Royal Society, 2017a). Clearly, the journey for teachers in England is not over, but as shown in our study, teachers have a 'story' to tell. They have encountered difficulties which some are starting to overcome. Such experiences could be of great interest to others embarking on a similar journey.

5.6.2. International generalisability

Vahrenhold et al. (2019) calls into question whether findings from one country's unique experience can be translated to other settings, however. The question then is, despite our teachers' unique and radical journey, is their current experience of difficulties with design seen in other settings?

Pockets of evidence of similar difficulties in using design in classroom programming contexts have been reported in studies in other settings. Statter and Armoni (2016),

in investigating the teaching of abstraction through the development of algorithms in Israeli middle school programming lessons, reported, from observation and from talking to teachers, that students tended to rush to start coding without thinking ahead. Student comments on the value of design documentation imply resistance, as they say they did not see the need to do design but that their teacher had forced them. Statter and Armoni (2016, 2017)'s studies do not mention the other aspects which we found as difficulties, such as if teachers or students were confused about what an algorithm was or whether there were issues with time to do design in class.

Time as a difficulty was mentioned by Rahimi et al. (2018). In the development of the Dutch computer science curriculum for secondary students, Rahimi et al. (2018) reported some secondary students saying flowcharts, a design product, was time-consuming and difficult to create. This corresponds to our results that English primary teachers reported time and a lack of pupil expertise as difficulties with design. Rahimi et al. (2018) do not explicitly mention resistance to design; however, this is alluded to in the reporting of a positive impact of the explicit introduction of a design, as students said they were "forced" to think ahead.

In line with our results, Falkner and Vivian (2015) and Rich et al. (2017), in surveys of K-12 resources and K-8 teaching of programming reported a lack of resources, teacher expertise and training for the teaching of program design in Australia, USA, UK and Finland. Unlike our study, Statter and Armoni (2016); Rahimi et al. (2018); Falkner and Vivian (2015); Rich et al. (2017) in Israel, the Netherlands, Australia and the USA, respectively, were not specifically investigating K-5 teachers' reported difficulties with design and yet they each highlighted in some way individual difficulties with design which are in common with our findings. The studies vary in many aspects, including the age group studied, methods used and difficulty reported.

Notwithstanding these differences and Vahrenhold et al. (2019)'s caution when translating findings across country specific settings, we have thus found common difficulties in teaching design in programming activities across geographical spread. Further work across different contexts is needed to show the results generalise.

5.6.3. Relevance and importance

Computing, including programming, has been, or is being introduced, in classrooms to younger pupils across the world. Recently lists of countries adding programming have included the United Kingdom, Finland, Australia, Croatia, Slovenia, Ukraine, Greece, South Korea and some parts of Spain, Italy, Hong Kong, Germany and the USA (Rich et al., 2018; The Committee on European Computing Education (CECE), 2017). In the teaching of programming, there is already a consensus that design matters (Soloway, 1986; Kant and Newell, 1984; Denning, 2003; Armoni, 2013) and there is evidence that explicit teaching of design is effective to improve outcomes (Perrenet et al., 2005; Cutts et al., 2012; Statter and Armoni, 2016, 2017).

Our study provides evidence that despite the importance and effectiveness of including design in the teaching of programming, and mandatory delivery of this for some five years (Department for Education, 2013a), there are barriers for English K-5 programming teachers to overcome to incorporate design in their lessons. This is not because these teachers, despite little training and limited expertise in design, think design is unimportant. Quite the opposite, they report design as being essential or very important (Waite et al., 2018b).

Programming is being introduced to K-5 learners across the world, and design is an important aspect of this. Difficulties in design are compromising the effective teaching of programming. Therefore, as a significant aspect of current curricula development, countries introducing programming to their younger learners should investigate their teacher's views on design and what difficulties are constraining incorporation of this important aspect of teaching of programming. Our work provides a method to facilitate such an investigation, and our data gives a benchmark of one country-specific experience of the difficulties with design against which others can compare.

5.7. Resistance

A difficulty that should be of significant interest to the wider community is that of pupil resistance to design. Turkle and Papert (1990), working with Logo, stated that learners are naturally either planners or *bricoleurs*. This implication that *bricoleurs* are resistant to overt design seems to have now manifested, thirty years later, and be causing much difficulty to our teachers. As stated by one teacher in our survey:

> *"A minority [of pupils] find it [design] useful but I think most find it unnecessary and a chore. They don't link it [design] to computing."* [Teacher 207]

This fixed mindset is in opposition to recommendations that planning is essential in teaching generally and that external representations (e.g., drawings of design) help reduce cognitive load when undertaking complex tasks (Graham et al., 2012; Kirsh, 2010). We agree with Schulte et al. (2017) that learning to program requires a balance between exploration and learning about, and to, design. Therefore investigation of the merits of, and resistance to, design for young learners is paramount, to any community which sees design as an essential aspect of programming. We particularly question whether the high profile focus on coding and getting everyone started through exploration or by copying code, is diminishing the role of design in programming.

5.8. Role and contribution of teachers

Our research has explored what teachers are doing in class. We have uncovered unexpected difficulties in the teaching of design. We have also found early adopter teachers who have accumulated experience and considerable expertise to start to overcome these difficulties. Through years of experience, they are beginning to bridge between the implemented and intended curriculum, wrestling with intrinsic and extrinsic challenges (Finger and Houguet, 2009). These struggles could not have been foretold or overcome without time and experience and have only been revealed by engaging with those working at the chalk-face. This adds weight to Vahrenhold et al. (2019)'s proposal that teachers should be supported to undertake research in their classes. Research capacity and funding for computer science education is necessary not only generally (The Royal Society, 2017a) but specifically to place teachers at the centre of computer science education research.

6. Conclusions

Literature shows that design and algorithms matter in the teaching of programming (Soloway, 1986; Denning, 2003; Armoni, 2013). Reflecting this, design and algorithms are

included in the computing curriculum in England and many other countries (Rich et al., 2018; The Committee on European Computing Education (CECE), 2017). However, there is a clear gap in the literature related to how design and algorithms are now being taught to young learners and what is the best way to do this (Waite et al., 2017; The Royal Society, 2017a).

When learning how to develop algorithms within the design process, certain factors constrain the production of precise representations. These constraints include learning through unplugged activities (Curzon et al., 2019), the need to explore to learn (Schulte et al., 2017), natural limitations of design formats used (Kant and Newell, 1984) and the design process itself (Knuth, 1997). These constraints may be contributing to issues related to resource development. Resources created to support K-5 teachers to implement recent computing curriculum changes lack a common message about what algorithms are, and program design has been lacking in both resources and teacher professional development (Rich et al., 2017; Falkner and Vivian, 2015). Yet, as presented in a previous report of our survey, 82% of the participating teachers reported design in programming projects as being either 'Essential' or 'Very Useful' but just half of these teachers, 44%, converted this view of importance into action of using design in classroom 'Always' or 'Usually' (Waite et al., 2018b).

Our objective was to discover the difficulties that K-5 teachers have with implementing design in programming. We revealed six major difficulties: pupil resistance; a lack of time; a lack of pupil expertise; conflicting pedagogic choices; a lack of teacher expertise and confusion of what an algorithm is and so where algorithms fit in the design process as it moves from ambiguity to unambiguous code. Some teachers mitigate the 'burdon' of design through careful teaching of the benefits of overt forms of drawn or written design. However, it is not clear which format results in most progress, nor what the long term impact of different pedagogical choices have on teaching and learning.

Surveyed teachers disagreed as to whether a simple or complex code snippet, storyboard or concept maps should be called an 'algorithm' or not. Teachers' selection of the term 'algorithm' differed according to many variables, including gender, generalist or specialist teacher, programming language, age of pupils taught, resources used and training received. The reasons for many of these differences are unclear. It seems that teachers' understanding of what an algorithm is varies. Useful concepts to support teachers improve their understanding of algorithms include using the notion of representations of algorithms (see Section 2.3) and the process of refinement of the precision of representations from ambiguous to unambiguous (see Section 2.3.1).

Despite most teachers having received training on programming, the majority of teachers had not received any training on design. There are clearly opportunities to rectify this. Also, despite there being many different resources used by teachers to teach programming over half of the teachers used one of three popular resources. There may be an opportunity to use these resources, embedding aspects of design, to increase teacher training and address the confusion over what an algorithm is.

On a wider stage, our findings are unique to our current educational context. In England, K-5 teachers have been implementing a *radical* curriculum change for five years, and expert teachers are emerging who can inform research in ways not previously possible. How transferable our research, based on teacher views, is to other settings is questionable. However, our findings related to pupil resistance to design are too important to be dismissed as local issues. There are opportunities for those countries who are earlier in

their journey to learn from England's difficulties and avoid them.

7. Further Work

There is a need to support teachers to overcome their difficulties in using design. Therefore our next step will be to work with K-5 expert teachers to iteratively develop a design artefact model and a design toolkit. The model will define the aspects of design in programming projects and the toolkit will support teachers to apply this model. The representation of an algorithm will be a core component of our model and we will define age-appropriate representations of designs and algorithms, what design approaches are available to teachers and what level of precision might be most useful, desirable or practicable for a representation of an algorithm for children aged 5 to 11-years-old in classroom contexts. The model and toolkit will be made available for practitioners, educational resource developers and researchers to use, review and build upon. Considering a wider audience, further research is needed to replicate our study internationally to discover whether the results are generalisable beyond the English context.

References

References

Armoni, M., 2013. On teaching abstraction in cs to novices. Journal of Computers in Mathematics and Science Teaching 32, 265–284.

Berry, M., 2013. Computing in the national curriculum. A guide for primary teachers. Bedford: Computing at School .

Berry, M., Woollard, J., Hughes, P., Chippendal, J., Ross, Z., Waite, J., 2015. Barefoot computing resources. Online. URL: http://barefootcas.org.uk/.

Bers, M., 2017. Coding as a Playground: Programming and Computational Thinking in the Early Childhood Classroom. Taylor & Francis.

Carlisle, J.F., Fleming, J.E., Gudbrandsen, B., 2000. Incidental word learning in science classes. Contemporary Educational Psychology 25, 184–211. doi:10.1006/ceps.1998.1001.

Cohen, L., Manion, L., Morrison, K., 2011. Research methods in education. volume 7th Edition. Routledge.

Cothran, D.J., Ennis, C.D., 1997. Students' and teachers' perceptions of conflict and power. Teaching and teacher education 13, 541–554.

Crick, T., Sentance, S., 2011. Computing at school: stimulating computing education in the UK., in: Proceedings of the 11th Koli Calling International Conference on Computing Education Research, ACM, New York, NY, USA. pp. 122–123. doi:10.1145/2094131.2094158.

Cross, N., 2004. Expertise in design: an overview. Design studies 25, 427–441.

Curzon, P., Bell, T., Waite, J., Dorling, M., 2019. Computational thinking, in: Fincher, S.A., Robins, A.V. (Eds.), The Cambridge Handbook of Computing Education Research. Cambridge Handbooks in Psychology, pp. 513–547. doi:10.1017/9781108654555.018.

Cutts, Q., Connor, R., Michaelson, G., Donaldson, P., 2014. Code or (not code): separating formal and natural language in cs education, in: Proceedings of the 9th Workshop in Primary and Secondary Computing Education, ACM. ACM, New York, NY, USA. pp. 20–28. doi:10.1145/2670757.2670780.

Cutts, Q., Esper, S., Fecho, M., Foster, S., Simon, B., 2012. The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition, in: Proceedings of the ninth annual international conference on International computing education research, ACM. pp. 63–70. doi:10.1145/2361276.2361290.

Denning, P.J., 2003. Great principles of computing. Communications of the ACM 46, 15–20.

Denning, P.J., 2017. Remaining trouble spots with computational thinking. Communications of the ACM 60, 33–39. doi:10.1145/2998438.

Department for Education, 2013a. Computing programmes of study key stages 1 and 2 national curriculum in England. URL: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study. may 5, 2015.

Department for Education, 2013b. Computing programmes of study: key stages 3 and 4 national curriculum in England. Online. URL: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf. may 5, 2015.

Department for Education, 2013c. English programmes of study: key stages 1 and 2 national curriculum in England. Online. URL: http://www.nationalarchives.gov.uk/doc/open-government-licence/.

Department for Education, 2013d. Mathematics programmes of study, key stages 1 and 2: National curriculum in England. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/335158/PRIMARY_national_curriculum_-_Mathematics_220714.pdf.

Department for Education, 2014. GCSE AS and A level subject content for computer science. URL: https://www.gov.uk/government/publications/gce-as-and-a-level-for-computer-science.

Falkner, K., Vivian, R., 2015. A review of computer science resources for learning and teaching with k-12 computing curricula: An Australian case study. Computer Science Education 25, 390–429. doi:10.1080/08993408.2016.1140410.

Finger, G., Houguet, B., 2009. Insights into the intrinsic and extrinsic challenges for implementing technology education: case studies of queensland teachers. International Journal of Technology and Design Education 19, 309–334. URL: https://doi.org/10.1007/s10798-007-9044-2, doi:10.1007/s10798-007-9044-2.

Furber, S., et al., 2012. Shut down or restart? the way forward for computing in UK schools. The Royal Society, London .

Graham, S., Bollinger, A., Olson, C., DAoust, C., MacArthur, C., McCutchen, D., Olinghouse, N., 2012. Teaching elementary school students to be effective writers. What Works Clearinghouse, US Department of Education .

Grover, S., Pea, R., 2013. Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students, in: Proceeding of the 44th ACM technical symposium on Computer science education, ACM. pp. 723–728. doi:10.1145/2445196.2445404.

Harel, D., Feldman, Y.A., 2004. Algorithmics: the spirit of computing. Addison-Wesley, Reading, Mass.

Hodgen, J., Foster, C., Marks, R., Brown, M., 2018. Evidence for review of mathematics teaching: improving mathematics in key stages two and three: evidence review. Education Endowment Foundation, London.

Hubwieser, P., Giannakos, M.N., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., Pal, Y., Jackova, J., Jasute, E., 2015. A global snapshot of computer science education in K-12 schools, in: Proceedings of the 2015 ITiCSE on Working Group Reports, ACM. ACM, New York, NY, USA. pp. 65–83. doi:10.1145/2858796.2858799.

Kafai, Y.B., Vasudevan, V., 2015. Constructionist gaming beyond the screen: Middle school students' crafting and computing of touchpads, board games, and controllers, in: Proceedings of the Workshop in Primary and Secondary Computing Education, ACM. ACM, New York, NY, USA. pp. 49–54. doi:10.1145/2818314.2818334.

Kant, E., Newell, A., 1984. Problem solving techniques for the design of algorithms. Information Processing & Management 20, 97–118.

Kastl, P., Kiesmüller, U., Romeike, R., 2016. Starting out with projects: Experiences with agile software development in high schools, in: Proceedings of the 11th Workshop in Primary and Secondary Computing Education, ACM. ACM, New York, NY, USA. pp. 60–65. doi:10.1145/2978249.2978257.

Kemp, P., 2014. Computing in the national curriculum-a guide for secondary teachers. computing at school. Online. URL: https://www.computingatschool.org.uk/data/uploads/cas_secondary.pdf.

Kirsh, D., 2010. Thinking with external representations. AI and Society 25, 441–454. doi:10.1007/s00146-010-0272-8.

Knuth, D.E., 1997. The art of computer programming: sorting and searching. volume 3. Pearson Education.

Kuckartz, U., 2014. Qualitative text analysis: A guide to methods, practice and using software. Sage. doi:10.4135/9781446288719.

Landis, J.R., Koch, G.G., 1977. The measurement of observer agreement for categorical data. Biometrics 33, 159–174.

Mayring, P., 2000. Forum: Qualitative social research sozialforschung, 2. history of content analysis, in: Forum: Qualitative Social Research. Sozialforschung.

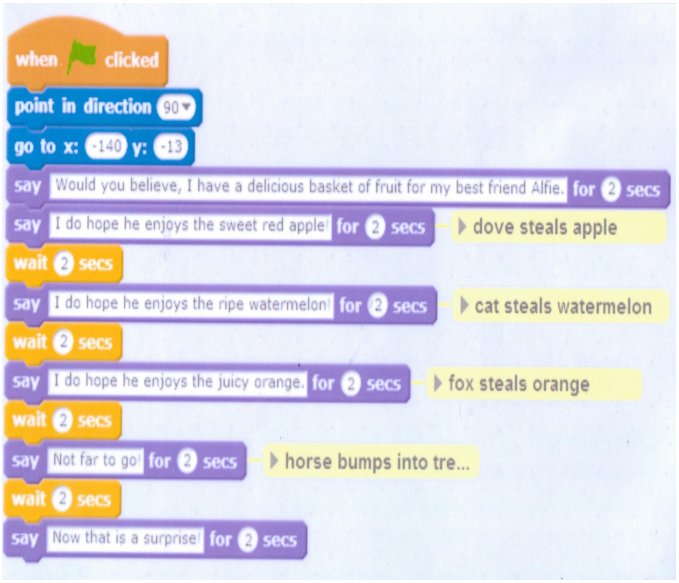
Meek, G.E., Ozgur, C., Dunning, K., 2007. Comparison of the t vs. Wilcoxon signed-rank test for Likert

scale data and small samples. Journal of Modern Applied Statistical Methods 6, 10.
Oxford University Press, 2018. OED online. Online. URL: www.oed.com/viewdictionaryentry/Entry/11125.
Perrenet, J., Groote, J.F., Kaasenbrood, E., 2005. Exploring students' understanding of the concept of algorithm: levels of abstraction. ACM SIGCSE Bulletin 37, 64–68. doi:10.1145/1067445.1067467.
Rahimi, E., Barendsen, E., Henze, I., 2018. An instructional model to link designing and conceptual understanding in secondary computer science education, in: Proceedings of the 13th Workshop in Primary and Secondary Computing Education, ACM, New York, NY, USA. pp. 11:1–11:4. doi:10.1145/3265757.3265768.
Rich, K., Strickland, C., Franklin, D., 2017. A literature review through the lens of computer science learning goals theorized and explored in research, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, ACM, New York, NY, USA. pp. 495–500. doi:10.1145/3017680.3017772.
Rich, P.J., Browning, S.F., Perkins, M., Shoop, T., Yoshikawa, E., Belikov, O.M., 2018. Coding in k-8: International trends in teaching elementary/primary computing. TechTrends , 1–19.
Sapir, E., 1921. Language, An introduction to the study of speech. Harcourt, Brace & World.
Schulte, C., Magenheim, J., Müller, K., Budde, L., 2017. The design and exploration cycle as research and development framework in computing education, in: Global Engineering Education Conference (EDUCON), 2017 IEEE, IEEE. pp. 867–876. doi:10.1109/EDUCON.2017.7942950.
Sfard, A., 1991. On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin. Educational studies in mathematics 22, 1–36. doi:10.1007/BF00302715.
Shulman, L.S., 1986. Those who understand: Knowledge growth in teaching. Educational researcher 15, 4–14.
Soloway, E., 1986. Learning to program = learning to construct mechanisms and explanations. Communications of the ACM 29, 850–858. doi:10.1145/6592.6594.
Statter, D., Armoni, M., 2016. Teaching abstract thinking in introduction to computer science for 7th graders, in: Proceedings of the 11th Workshop in Primary and Secondary Computing Education, ACM. pp. 80–83. doi:10.1145/2978249.2978261.
Statter, D., Armoni, M., 2017. Learning abstraction in computer science: A gender perspective, in: Proceedings of the 12th Workshop on Primary and Secondary Computing Education, ACM. ACM, New York, NY, USA. pp. 5–14. doi:10.1145/3137065.3137081.
The Committee on European Computing Education (CECE), 2017. Informatics Education in Europe: Are We All In The Same Boat? Technical Report. New York, NY, USA.
The Royal Society, 2017a. After the reboot: computing education in UK schools. Online. The Royal Society, 6 – 9 Carlton House Terrace, London SW1Y 5AG. URL: royalsociety.org/computing-education.
The Royal Society, 2017b. After the Reboot: The State of Computing Education in UK Schools and Colleges Final Report September 2017. Technical Report. The Royal Society. URL: https://royalsociety.org/~/media/policy/projects/computing-education/pye-tait-teacher-survey-report.pdf.
Turkle, S., Papert, S., 1990. Epistemological pluralism: Styles and voices within the computer culture. Signs: Journal of women in culture and society 16, 128–157.
Vahrenhold, J., Cutts, Q., Falkner, K., 2019. Schools (K–12). Cambridge University Press. Cambridge Handbooks in Psychology, p. 547–583. doi:10.1017/9781108654555.019.
Waite, J., 2017. Pedagogy in teaching computer science in schools: A literature review (after the reboot: computing education in uk schools). Online. URL: https://royalsociety.org/~/media/policy/projects/computing-education/literature-review-pedagogy-in-teaching.pdf.
Waite, J., Curzon, P., Marsh, D., Sentance, S., Hawden-Bennett, A., 2018a. Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. International Journal Of Computer Science Education In Schools doi:10.21585/ijcses.v2i1.23.
Waite, J., Curzon, P., Marsh, W., Sentance, S., 2017. K-5 teachers' uses of levels of abstraction focusing on design, in: Proceedings of the 12th Workshop in Primary and Secondary Computing Education, ACM. pp. 115–116. doi:10.1145/3137065.3137068.
Waite, J., Curzon, P., Marsh, W., Sentance, S., 2018b. Comparing K-5 teachers' reported use of design in teaching programming and planning in teaching writing, in: Proceedings of the 13th Workshop in Primary and Secondary Computing Education. doi:10.1145/3265757.3265761.

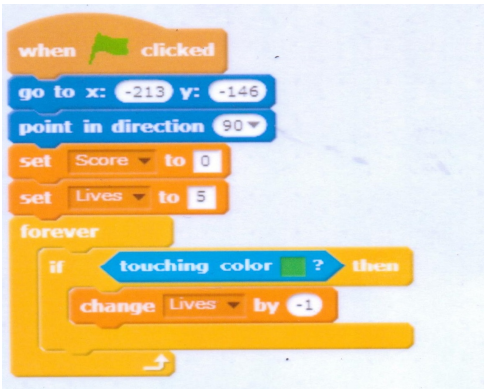
Appendix A. Survey Questions

Survey - Using Design in Primary Programming - Research Project

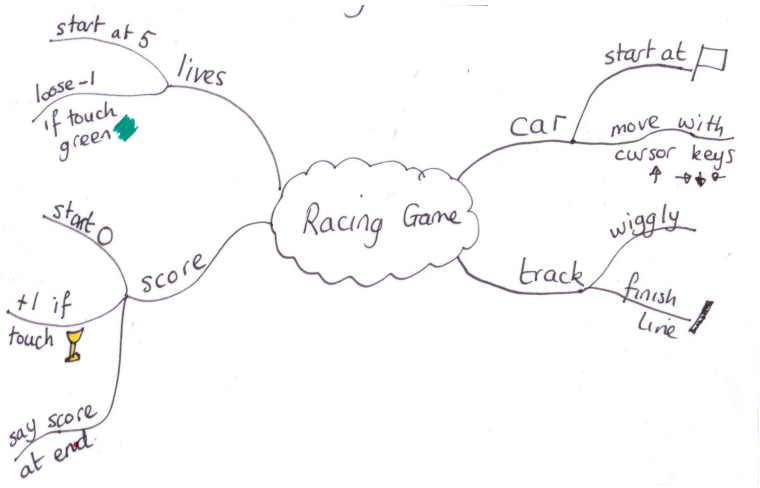
- Do you teach programming? Yes or No (If no skip to section on teaching writing)
- How confident are you teaching programming? (1 to 5 linear scale from very confident to not at all confident)
- What planning do you use for teaching programming? For example, we follow Somerset Grid for Learning plans, we make up our own and use Barefoot material.
- What programming languages do you use to teach programming to your current class?
- Which of these words would you use to describe this image? (You can choose more than one word if you like.) It is a picture of ... (A program, Some code, A plan, An algorithm, A script, A design, other please add)



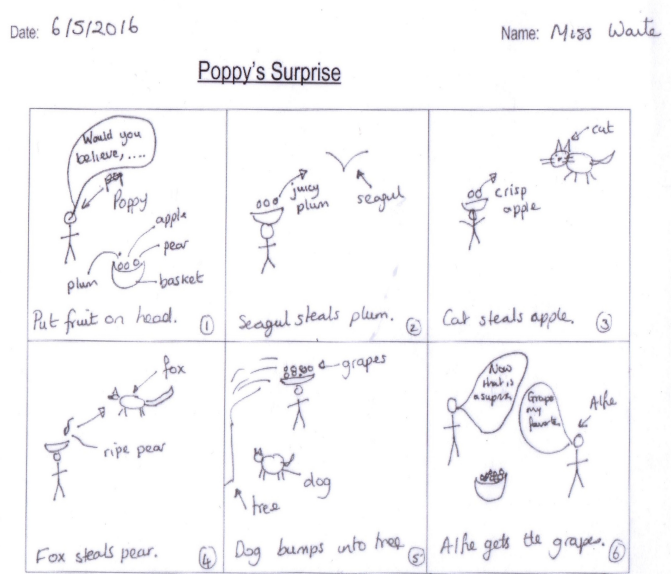
- Which of these words would you use to describe this image? (You can choose more than one word if you like.) It is a picture of ... (A program, Some code, A plan, An algorithm, A script, A design, other please add)



- Which of these words would you use to describe this image? (You can choose more than one word if you like.) It is a picture of ... (A program, Some code, A plan, An algorithm, A script, A design, other please add)



- Which of these words would you use to describe this image? (You can choose more than one word if you like.) It is a picture of ... (A program, Some code, A plan, An algorithm, A script, A design, other please add)



- What training have you had related to programming?
- Have you had any specific training on using design in programming? (No, Yes) If Yes, please detail
- When teaching programming, do you require your pupils to consider or use a design at some point? By design we mean a high-level plan of what program is intended to do. A design could be verbal, written or drawn, informal or formal. (1 to 5 scale from Always to Never)
- Based on your recent practice of teaching programming projects, did you expect pupils to ....
  - create a design before they start coding?
  - create a design at some point, but not straight away?
  - annotate their design with useful code snippets?
  - refer to their design when coding?
  - tick off what they have done so far on their design?
  - change their design once they have started coding?
  - mark things that did not work on their design?
  - note on their design their confidence to implement the different parts?
  - evaluate their design as they go along?
  - evaluate their design when they have finished their coding?
- For programming projects, do you
  - mark the designs created by pupils?

  - keep designs created by pupils?
  - use a design as a contract to support pupils when programming in pairs?
  - use designs to differentiate work?
  - use the designs created by pupils to work out what to teach next?
  - demonstrate to pupils (model) the creation of designs?
  - demonstrate to pupils the use of designs?
  - demonstrate to pupils the evaluation of designs?

- What formats of design do you use for programming projects? (Verbal, Written notes, Storyboards, Concept/Mind Maps, Labelled diagrams, Pseudocode, Flowcharts, Other)
- Do you create different types of design for different types of programming projects? e.g. storyboards for animations, lists for quizzes. If yes explain

  Identical questions were asked for teaching writing and are available on the Project Website.

  General views on design and planning

- Do you think design in programming projects is useful? (1 to 5 from Very useful to not at all useful)
- What DRAWBACKS (if any) are there in asking pupils to use a design in a programming project?
- What ADVANTAGES (if any) are there in asking pupils to use a design in a programming project?
- What do you think pupils think about being asked to use a design in a programming project?

  Identical questions were asked for teaching writing and are available on the Project Website.

- Do you have any other comments about design in programming or planning in writing?

  About your teaching

- What age of pupil do you teach?
- How long have you been teaching programming?
- How long have you been teaching writing?
- What are your teaching roles?
- To which gender identity do you most identify?
- What is your age?

Appendix B. Participant data

Table B1: Participant training in programming, showing the % and number (n) of participants reporting each category

Training type	%	n
Books and On the job	73%	150
Self Taught	73%	150
Conferences, short courses	47%	97
At University	18%	38
BCS certificate	12%	24
A Level in computing	9%	19
GCSE in computing	6%	13
None	9%	18
Prior education (GSCE, A Level or At University)	23%	49
During Initial Teacher Training (ITT)	6%	13
Personal study (Books, on job or self taught)	73%	152
Courses (Conference, short course or BCS certificate)	50%	105

In response to the questions on training, Computing At School (CAS) courses were mentioned by teachers and is summarised as CAS Courses. CAS is the English computing teacher's association which provides support and professional development for educators through a network of regional hubs (Crick and Sentance, 2011).

Table B2: Participant training in design, showing the % and number (n) of participants reporting each category

Training type	%	n
At University	9%	18
CAS courses	6%	13
Worked in Industry	2%	5
Self Taught	2%	5
Other short courses	2%	5
Online courses	1%	3
Other	1%	2
Any form of Training in Design	23%	49
No form of Training in Design	77%	158

In Table B2 the Computing At School (CAS) courses included Master Teacher Training, Diving Deep into Primary Programming Course, BCS Certificate and CAS Conference workshops. Other short courses, included a local authority course and an industry-sponsored course. The University courses were all computer science related rather than initial teacher training.

[7] https://barefootcas.org.uk/

Table B3: Curriculum planning used, showing the % and number (n) of participants who reported each category

Curriculum planning used	%	n
Create own resources	42%	86
Barefoot[7]	34%	71
code-it [8]	17%	35
Switched on computing [9]	14%	29
Combine a variety of products	11%	23
Wessex planning[10]	11%	22
Online e.g. Twitter	6%	13
Computing At School resources [11]	6%	12
Scratch resources	4%	8
Barefoot or code-it or Switched-on or Wessex	55%	114
code-it or Switched-on or Wessex	38%	78

Table B4: Programming languages taught, showing the % and number (n) of participants who reported 'Mostly', 'Sometimes' and Accumulated (Acc.)('Mostly' or 'Sometimes') for each category e.g. 17% of Generalists reported 'Mostly' using Scratch, 79% 'Sometimes' and 96% reported either 'Mostly' or 'Sometimes' using Scratch

Prog Language	Generalist			Specialist			Total
	Mostly % (n)	Sometimes % (n)	Acc. % (n)	Mostly % (n)	Sometimes % (n)	Acc. %(n)	%(n)
Scratch[12]	17% (17)	79%(80)	96% (97)	17%(18)	81% (84)	98% (104)	97% (201)
Scartch Jnr [13]	32% (32)	25% (26)	57% (58)	30%(32)	31% (33)	61% (65)	59% (123)
Purple Mash [14]	23% (23)	14% (14)	37% (37)	22% (24)	11%(11)	33% (35)	35% (72)
Blockly [15]	24% (24)	12% (12)	36% (36)	34% (36)	20% (21)	53% (57)	45% (93)
Kodu [16]	28% (28)	11% (11)	39% (39)	32% (33)	18% (19)	49% (52)	44% (91)
Python [17]	21% (21)	8% (8)	29% (29)	30% (32)	18% (19)	49% (51)	39%(80)
Hopscotch [18]	23% (23)	7% (7)	30% (30)	28% (29)	7% (7)	34% (36)	32%(66)
Daisy [19]	24% (24)	7% (7)	31% (31)	26% (27)	6% (6)	31% (33)	31% (64)
Espresso [20]	14%(14)	3%(3)	17% (17)	20%(21)	10%(10)	29% (31)	23%(48)
HTML			3% (3)			20% (21)	12%(24)

[8] http://code-it.co.uk/
[9] https://www.risingstars-uk.com/series/switched-on-computing
[10] https://slp.somerset.org.uk/sites/edtech/SitePages/Primary%20Computing/New%20Wessex%20Planning.aspx
[11] https://community.computingatschool.org.uk/resources/landing
[12] https://scratch.mit.edu/
[13] https://www.scratchjr.org/
[14] https://2simple.com/purple-mash/
[15] https://blockly-games.appspot.com/
[16] https://www.kodugamelab.com/
[17] https://www.python.org/
[18] https://www.gethopscotch.com/

Table B5: Who teaches who by Key Stage within generalist/specialist groups and overall, showing the % and number (n) of teachers who reported each category

Key Stage	Pupil age	US Grades	Generalist % (n)	Specialist % (n)	Overall Total % (n)
Early Years	3-4 years old	Pre-K	15% (15)	37% (39)	26% (54)
Primary					
KS1	5-7 years old	K-1	43% (44)	65% (69)	54% (113)
KS2	8-11 years old	2-5	90%(91)	84%(89)	87% (180)
Secondary					
KS3	12-14 years old	6-8	7%(7)	25%(27)	16% (34)
KS4	15-16 years old	9-10	5% (5)	13%(14)	9%(19)
KS5	17-18 years old	11-12	3%(3)	6%(6)	4% (9)
Adult			4%(4)	26%(28)	15% (32)
University			3%(3)	9%(10)	6% (13)

With regard to programming languages shown in Table: B4, participants reported 57 'other' programming languages in the free text option. Of these, 65% (n=37) were mentioned by no more than 1 teacher, a third (n=17) were mentioned by 2 to 7 teachers, Logo was mentioned by 12 teachers, the BBC Micro:bit was mentioned by 18 with a spread of associated languages (including PXT, block based, micropython or no specific programming language mentioned) and HTML was mentioned by 12% (n=24) teachers. HTML was therefore added as quantitative data. Data on the other programming languages reported by teachers is available on the project website.

[19]Daisy the Dinosaur https://itunes.apple.com/gb/app/daisy-the-dinosaur/id490514278?mt=8

[20]http://www.discoveryeducation.co.uk/what-we-offer/discovery-education-espresso

Appendix C. Quantitative Data - Statistically significant differences for selection of the term 'algorithm' for each image.

Table C1: Significant differences for selection of the term 'algorithm', showing the % and number (n) of teachers who selected the term for each variable and the test statistic

Variables tested showing values compared	Image 1 Simple Code Snippet	Image 2 Complex Code Snippet	Image 3 Concept Map Design	Image 4 Story Board Design	Test Statistic
Female	59% (74)				p=0.047, n=205
Male	45% (36)				Z=1.984, r=0.138
Generalist		70% (71)			p<0.00, n=207
Specialist		46% (49)			Z=3.499, r=.243
Programming Training				38%(72)	p=0.04, n=207
No Programming Training				5%(1)	Z=2.865, r= 0.199
Programming in ITT		92% (12)			p=0.01, n=205
No Programming in ITT		56% (108)			Z= 2.585, r=.179
No Design Training		63% (99)			p=0.009, n=207
Design Training		42% (21)			Z=.2.621,r=.181
Make own resources				43%(37)	p=0.049, n=207
Do not make own				30%(36)	, Z=1.965,r=.0.136
Mostly/Sometimes use DTD			44% (28)		p=0.004,n=207
Never use DTD			23% (34)		Z=2.892,r=0.201[21]
Mostly/Sometimes use DTD				45% (29)	p=0.043,n=207
Never use DTD				31% (44)	Z=2.019,r=0.140 [22]
Mostly Use Python			45% (24)		p=0.03, n=207
Never Use Python			26% (32)		Z=2.57, r=0.179
Mostly Use Python			45%(24)		p=0.041, n=207
Sometimes Use Python			18%(5)		Z=2.465, r= 0.171
Teach Early Years				46% (25)	p=0.049, n=207
Do Not teach Early Years				31% (48)	Z=1.969, r= 0.139
Teach KS5	89%(8)				p=.03, n=207
Do not teach KS5	52%(103)				Z=2.164, r=0.15

To read Table C1, for example, for the Female/ Male comparison for Image 1 (simple code snippet) there was a statistically significant difference between men and women in selection of the term 'algorithm' for Image 1. Women were more likely to call the simple code snippet an 'algorithm' than men. 59% of females reported selecting the simple code snippet as an 'algorithm' whereas 45% of males reported the same.

Looking in detail at each of these findings.

a) Gender - Male vs Female - Image 1 and 2. Females were statistically more likely to select the term 'algorithm' for the simple code snippet than males for the simple code snippet.

b) Generalist vs Specialist - Image 2. Generalists were statistically more likely to select the term 'algorithm' for the complex code snippet than specialists.

c) Training in programming vs No Training on programming - Image 4. Teachers who reported having had some training in programming (n=188) were statistically more likely to select the term 'algorithm' for the storyboard image than teachers who reported they had no training in programming by any method (n=19).

d) Programming training in ITT vs No Programming in ITT - Image 2. Teachers who had been trained in programming in their Initial Teacher Training course (ITT) (n=13) were statistically more likely to select the term 'algorithm' for complex code snippet compared to those who did not report they learned about programming in their Initial Teacher Training.

e)Training in design vs No Training in design - Image 2. Teachers who had not had training in design (n=157) were statistically more likely to select the term 'algorithm' for the more complex code than those who had received training in design, with 63% of them calling this an 'algorithm' compared to 42% of those who had had some form of design training.

f) Make own resources - Image 4. Teachers who reported making their own curriculum resources (n=86) were statistically more likely to select the term 'algorithm' for the storyboard than those who did not make their own curriculum resources.

We used the programming languages which had been reported by more than 10% as a grouping to compare the selection of the term algorithm for each of the images and found statistically significant differences for two programming languages, Daisy the Dinosaur and Python.

g) Use Daisy the Dinosaur - Image 3 and 4. Daisy the Dinosaur is a block-based programming language used for simple route based animations. There is only one character, Daisy, that can be controlled on one set background, no extra artwork can be added. Learners can program Daisy to move, spin, jump and roll as a sequence of commands or with a simple when or repeat command.

Those teachers who recorded teaching programming with Daisy the Dinosaur were statistically more likely to call both the concept map and the storyboard an 'algorithm' than those who did not teach with Daisy the Dinosaur.

h) Use Python - Image 3. Those teachers who reported 'Mostly' (n=53) using Python, compared to those who 'Never' used Python (n= 125) were more likely to select the term 'algorithm' for the concept map. Similarly, comparing teachers who reported 'Mostly' teaching Python to 'Sometimes' (n=27) teaching programming with Python there was the same trend seen (p=0.041, n=207, Z=2.465, r= 0.171).

i) Teach Early Years - Image 4. The Early Years teachers (n=54) were statistically more likely to assign the term 'algorithm' to a storyboard than their colleagues.

j) Teach KS5 - Image 1. There were very few KS5 teachers in our survey (n=9). However, they were statistically more likely to call the simple code snippet an 'algorithm' than other teachers (with 8 of 9 calling it an 'algorithm'). All 9 teachers also called image 2 an 'algorithm', but as more teachers generally called this image an 'algorithm', there was no statistically significant difference (p=0.055, Z=-1.917, n=207) in responses.

Comparing statistically significantly different variables for each image. For each image, for those variables which we found statistically significantly differences we compared each pair of variables to explore potential relationships using the Mann-Whitney U statistic.

i) For Image 1. Comparing Male or Female against teaching K5 learners, there was no statistically significant difference.

ii) Image 2. Comparing Generalists or Specialists against and having had Programming Training in Initial Teacher (ITT) Training, there was a statistically significant difference (p=0.037, n= 207, Z=-2.091, r=.145) with small effect size, only 13 teachers said they covered programming in their ITT. Of these 77% (n=10) were generalists and only 23% (n=3) specialists. 12 of the 13 teachers responded in line with generalists.

Comparing generalists and specialist against having had or not had design training there was no statistically significant difference. Similarly having design training and having had programming training in ITT there was no statistically significant difference.

iii) Image 3. Comparing using Python and Daisy the Dinosaur, there was a statistically significant difference (p=0.025,n=207 Z=-2.239, r=.155) with small effect size. Of the 80 Python users, (either 'Mostly' or 'Sometimes' using Python), 40% (n=32) used Daisy the Dinosaur and 60% did not. In other words, teachers were less likely to use Daisy the Dinosaur if they used Python.

iv) Image 4. Comparing the use of Daisy the Dinosaur and teaching Early Years, there was a statistically significant difference (p=.001, n= 207, Z=3.179, r=.221), with small effect size. 75% (n=115) of non-Early teachers never used the Daisy, and 25% 'Sometimes' or 'Mostly' used Daisy the Dinosaur compared to 52% of Early Years teachers never using the product and 48% 'Sometimes' or 'Mostly' using it. In other words, Early Years teachers were the ones using Daisy the Dinosaur.

However, having had training on programming and making your own resources were statistically significantly different groups (p=0.17, n=207, Z=-2.385, r=,165), with small effect size. 96% of the 86 teachers who had created their own resources had received some form of programming training, implying perhaps that training reveals to teachers the need to look beyond the current curriculum products being offered.