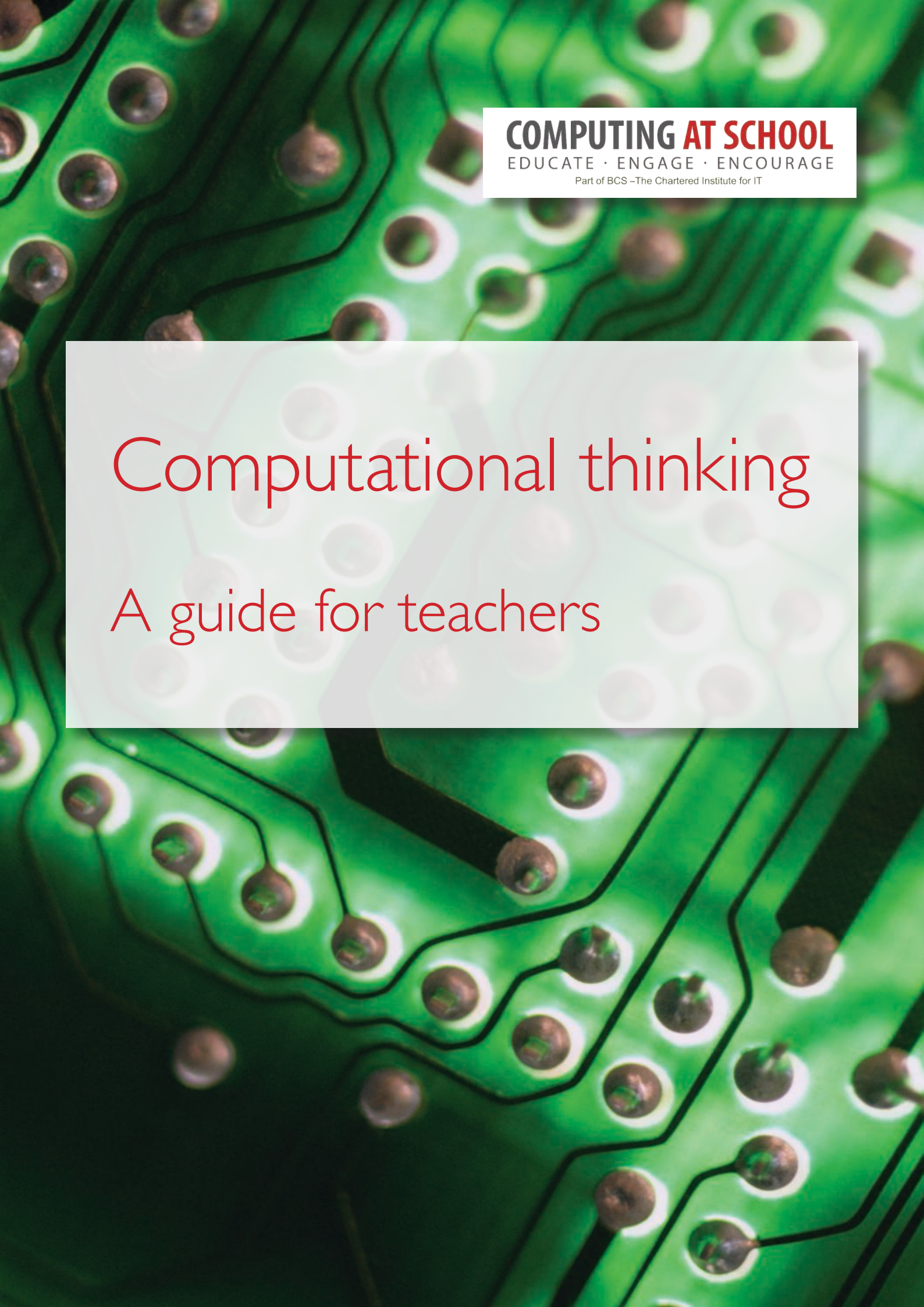


COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
Part of BCS –The Chartered Institute for IT

Computational thinking

A guide for teachers

Computing At School would like to acknowledge the following organisations for their support in the publication of this guide:

Hodder Education - the educational division of Hachette UK

Digital Schoolhouse - inspirational computing for kids http://www.digitalschoolhouse.org.uk

Authors

Andrew Csizmadia
School of Education, Newman University, Birmingham

Prof. Paul Curzon
Queen Mary University of London, School of Electronic Engineering and Computer Science
Teaching London Computing Project http://www.teachinglondoncomputing.org

Mark Dorling
CEO and co-founder Progression Pathways http://www.progression-pathways.co.uk
Digital Schoolhouse London Project http://www.digitalschoolhouse.org.uk

Simon Humphreys
National Coordinator for Computing At School http://www.computingatschool.org.uk

Thomas Ng
West Berkshire Council School Improvement Adviser (ICT & Assessment)

Dr Cynthia Selby
University of Southampton http://www.southampton.ac.uk/education/about/staff

Dr John Woollard
University of Southampton http://www.southampton.ac.uk/education/about/staff

An ebook version of this guide, which can be freely shared with colleagues, is available at:
http://computingatschool.org.uk/computationalthinking

Computational thinking

A guide for teachers

Foreword

Computational Thinking has become the buzz term for many teachers in England with the advent of the new Computing National Curriculum in September 2014. Computing At School has been at the forefront of advising on this change and providing much needed support to both primary and secondary teachers faced with the challenge of bringing into being a new subject in our schools. No-one underestimates that challenge and I have deep respect for the professionalism of the teachers I meet as they take their first steps in meeting this challenge. It is not easy. New vocabulary needs to be learnt, new skills acquired and new ways of teaching adopted. At the heart of the new curriculum is computational thinking and the role it has to play for our 21st century learners. This document sets out a conceptual framework for understanding computational thinking in the new Programme of Study. I trust it will be useful for all teachers, wherever they may be on their journey with the new Computing curriculum.

Simon Humphreys

National Coordinator, Computing At School

Computing At School promotes the teaching of computing in schools. Our aim is to support all teachers and all schools, and to develop excellence in the teaching of computing in their classrooms. We provide resources. training, local conferences and workshops, regional hub meetings, online community forums and so much more! Computing At School is free to join. Sign up and find out about events in your area by visiting us at www.computingatschool.org.uk.

Contents

Introduction

This guide aims to help develop a shared understanding of the teaching of computational thinking in schools. It presents a conceptual framework of computational thinking, describes pedagogic approaches for teaching and offers guides for assessment. It is complementary to the two CAS guides published in November 2013 (Primary) and June 2014 (Secondary) in supporting the implementation of the new National Curriculum and embraces the CAS Barefoot and CAS QuickStart Computing descriptions of computational thinking. Computational thinking lies at the heart of the computing curriculum but it also supports learning and thinking in other areas of the curriculum.

In the English National Curriculum for Computing, computational thinking sits at the heart of the new statutory Programme of Study:

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

(Department for Education, 2014, p. 217).

This new computing curriculum has an enriched computer science element. Computer science is an academic discipline with its own body of knowledge that can equip pupils to become independent learners, evaluators and designers of new technologies. In studying computer science, pupils gain skills, knowledge and a unique way of thinking about and solving problems: computational thinking. It allows the pupils to understand the digital world in a deeper way, just as physics equips pupils to better understand the physical world and a modern foreign language equips pupils to gain a richer understanding of other cultures. Computational thinking also gives a new paradigm for thinking about and understanding the world more generally. Simon Peyton-Jones succinctly explains why learning computer science and computational thinking is a core life skill – as well as being eminently transferable – in a talk filmed at TEDxExeter (http://bit.ly/13pJLCR).

Computational thinking skills are the set of mental skills that convert "complex, messy, partially defined, real world problems into a form that a mindless computer can tackle without further assistance from a human." (BCS, 2014, p. 3)

This guide introduces the nature of computational thinking and provides a vocabulary by which teachers can communicate, understand and teach the important concepts, approaches and techniques associated with computational thinking. It identifies where computational thinking occurs in the computing curriculum and how it might be introduced into the classroom.

The nature of computational thinking

Computational thinking provides a powerful framework for studying computing, with wide application beyond computing itself. It is the process of ***recognising*** aspects of computation in the world that surrounds us and ***applying*** tools and techniques from computing to understand and reason about natural, social and artificial systems and processes. It allows pupils to tackle problems, to break them down into solvable chunks and to devise algorithms to solve them. The term computational thinking was first used by Seymour Papert, though Professor Jeannette Wing popularised the idea in advocating computational thinking for all new university students (Wing, 2006). She defined computational thinking as:

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

(Cuny, Snyder, Wing, 2010, cited in Wing, 2011, p.20)

and

> *"The solution can be carried out by a human or machine, or more generally, by combinations of humans and machines."*

(Wing, 2011, p. 20).

The emphasis is clear. It concentrates on pupils performing a thought process, not on the production of artefacts or evidence. Computational thinking is the development of thinking skills and it supports learning and understanding.

Concepts of computational thinking

Computational thinking is a cognitive or thought process involving **logical reasoning** by which problems are solved and artefacts, procedures and systems are better understood. It embraces:

- the ability to think **algorithmically**;
- the ability to think in terms of **decomposition**;
- the ability to think in **generalisations**, identifying and making use of **patterns;**
- the ability to think in **abstractions**, choosing good **representations**; and
- the ability to think in terms of **evaluation.**

Computational thinking skills enable pupils to access parts of the Computing subject content. Importantly, they relate to thinking skills and problem solving across the whole curriculum and through life in general.

Computational thinking can be applied to a wide range of artefacts including: systems, processes, objects, algorithms, problems, solutions, abstractions, and collections of data or information. In the following discussion of concepts, **artefact** refers to any of these.

Logical reasoning

Logical reasoning enables pupils to make sense of things by analysing and checking facts through thinking clearly and precisely. It allows pupils to draw on their own knowledge and internal models to make and verify predictions and to draw conclusions. It is used extensively by pupils when they test, debug, and correct algorithms. Logical reasoning is the novel application of the other computational thinking concepts to solve problems.

Design and technology pupils, designing a model of a truck, choose materials for different elements of a project. They are employing generalisation when they recognise that the properties of a material used in one situation make it suitable to use in another completely different context. Being able to divide the new project into different parts, requiring different materials, is an example of decomposition. The pupil is using logical reasoning to design a truck.

Pupils use logical reasoning when learning about gravity using a weighted string suspended from the lid of a glass jar. Before tilting the jar, pupils can make predictions about the behaviour of the weighted string. They can then evaluate the results of their tests. They may be able to generalise the behaviour to other situations such as a crane. The novel use in understanding a property of gravity is logical reasoning.

Logical reasoning is key in allowing pupils to debug their code. They can work with peers to evaluate each other's code, to isolate bugs, and to suggest fixes. During this process, they may have opportunities to employ abstraction, evaluation, and algorithmic design. The novel use in correcting mistakes in code requires logical reasoning.

Abstraction

Abstraction makes problems or systems easier to think about. Abstraction is the process of making an artefact more understandable through reducing the unnecessary detail. A classic example is the London Underground map. London is a highly complex system. The representation of London in particular ways (usually maps or pictures) aids different users. The London Underground map is a highly refined abstraction with just sufficient information for the traveller to navigate the underground network without the unnecessary burden of information such as distance and exact geographic position. It is a representation that contains precisely the information necessary to plan a route from one station to another – and no more!

The skill in abstraction is in choosing the right detail to hide so that the problem becomes easier, without losing anything that is important. A key part of it is in choosing a good representation of a system. Different representations make different things easy to do.

For example, a computer program that plays chess is an abstraction. It is a finite and precise set of rules carried out each time that it is the computer's turn. It is far removed from the analogue, emotional, biased and distracted mental processes undertaken by a human player of chess. It is an abstraction because the unnecessary detail of those processes is removed.

Evaluation

Evaluation is the process of ensuring that a solution, whether an algorithm, system, or process, is a good one: that it is fit for purpose. Various properties of solutions need to be evaluated. Are correct? Are they fast enough? Do they use resources economically? Are they easy for people to use? Do they promote an appropriate experience? Trade-offs need to be made, as there is rarely a single ideal solution for all situations. There is a specific and often extreme focus on attention to detail in evaluation based on computational thinking.

Computer interfaces are being continually developed to meet the needs of different users. For example, if a medical device is needed to deliver drugs automatically to a patient, it needs to be programmable in an error-free, quick, simple and safe way. The solution must ensure that nurses will be able to set the dose easily without making mistakes and that it won't be frustrating for patients and nurses to use. In the proposed design there would be a trade-off to be made between speed of entering numbers (efficiency) and error avoidance (effectiveness and usability). The design would be judged on the specification proposed by clinicians, regulators and medical device design experts (criteria) and the general rules relating to good design (heuristics). Criteria, heuristics and user needs enable judgements to be made systematically and rigorously.

These computational thinking concepts have been summarised for the primary phase (applicable across all key stages) by CAS Barefoot Computing http://www.BarefootCAS.org.uk.

Algorithmic thinking

Algorithmic thinking is a way of getting to a solution through a clear definition of the steps. Some problems are one-off; they are solved, solutions are applied, and the next one is tackled. They are solved, solutions are applied, and the next one is tackled. Algorithmic thinking needs to kick in when similar problems have to be solved over and over again. They do not have to be thought through anew every time. A solution that works every time is needed. Learning algorithms for doing multiplication or division at school is an example. If simple rules are followed precisely, by a computer or a person, the solution to any multiplication can be found. Once the algorithm is understood, it doesn't have to be worked out from scratch for every new problem.

Algorithmic thinking is the ability to think in terms of sequences and rules as a way of solving problems or understanding situations. It is a core skill that pupils develop when they learn to write their own computer programs.

Decomposition

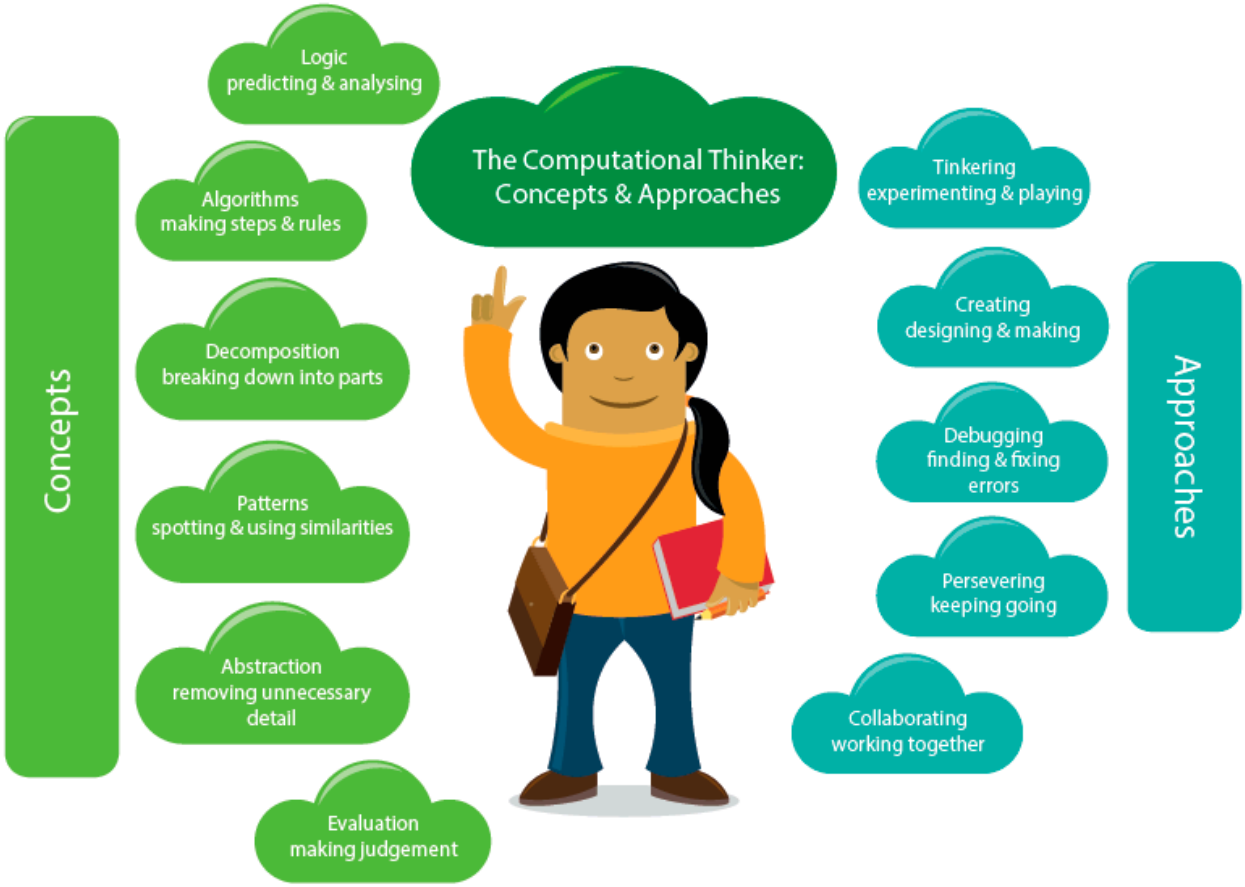
Decomposition is a way of thinking about artefacts in terms of their component parts. The parts can then be understood, solved, developed and evaluated separately. This makes complex problems easier to solve, novel situations better understood and large systems easier to design.

For example, making breakfast can be broken down, or decomposed, into separate activities such as make toast; make tea; boil egg; etc. Each of these, in turn, might also be broken down into a set of steps. Through decomposition of the original task each part can be developed and integrated later in the process. Consider developing a game: different people can design and create the different levels independently, provided that key aspects are agreed in advance. A simple arcade level might also be decomposed into several parts, such as the life-like motion of a character, scrolling the background and setting the rules about how characters interact.

Generalisation (Patterns)

Generalisation is associated with identifying patterns, similarities and connections, and exploiting those features. It is a way of quickly solving new problems based on previous solutions to problems, and building on prior experience. Asking questions such as "Is this similar to a problem I've already solved?" and "How is it different?" are important here, as is the process of recognising patterns both in the data being used and the processes/strategies being used. Algorithms that solve some specific problems can be adapted to solve a whole class of similar problems. Then whenever a problem of that class is encountered, the general solution can be applied.

For example, a pupil uses a floor turtle to draw a series of shapes, such as a square and a triangle. The pupil writes a computer program to draw the two shapes. They then want to draw an octagon and a 10-sided shape. From the work with the square and triangle, they spot that there is a relationship between the number of sides in the shape and the angles involved. They can then write an algorithm that expresses this relationship and use it to draw any regular polygon.



Techniques associated with computational thinking

There are a number of techniques employed to demonstrate and assess computational thinking. Think of this as '**computational doing**'. These are the 'computer science' equivalent of 'scientific methods'. They are the tools by which computational thinking is operationalised in the classroom, workplace and home. They are reflected in the Key Stage 3 classroom practice.

Reflecting

Reflection is the skill of making judgements (evaluation) that are fair and honest in complex situations that are not value-free. Within computer science this evaluation is based on criteria used to specify the product, heuristics (or rules of thumb) and user needs to guide the judgements.

Coding

An essential element of the development of any computer system is translating the design into code form and evaluating it to ensure that it functions correctly under all anticipated conditions. Debugging is the systematic application of analysis and evaluation using skills such as testing, tracing, and logical thinking to predict and verify outcomes.

Designing

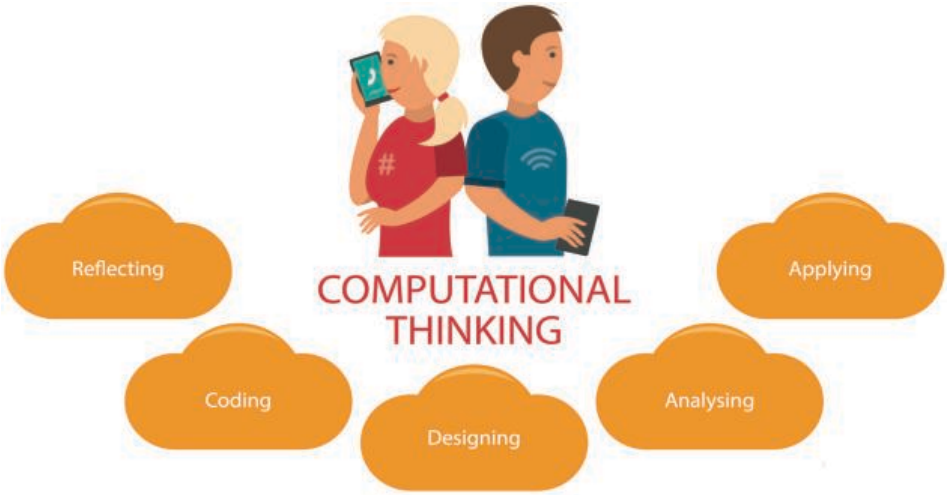
Designing involves working out the structure, appearance and functionality of artefacts. It involves creating representations of the design, including human readable representations such as flowcharts, storyboards, pseudo-code, systems diagrams, etc. It involves further activities of decomposition, abstraction and algorithm design.

Analysing

Analysing involves breaking down into component parts (decomposition), reducing the unnecessary complexity (abstraction), identifying the processes (algorithms) and seeking commonalities or patterns (generalisation). It involves using logical thinking both to better understand things and to evaluate them as fit for purpose.

Applying

Applying is the adoption of pre-existing solutions to meet the requirements of another context. It is generalisation - the identification of patterns, similarities and connections - and exploiting those features of the structure or function of artefacts. An example includes the development of a subprogram or algorithm in one context that can be re-used in a different context.



Concepts

Logic
predicting & analysing

Algorithms
making steps & rules

Decomposition
breaking down into parts

Patterns
spotting & using similarities

Abstraction
removing unnecessary
detail

Evaluation
making judgement

The Computa
Concepts &

ational Thinker: & Approaches

Tinkering
experimenting & playing

Creating
designing & making

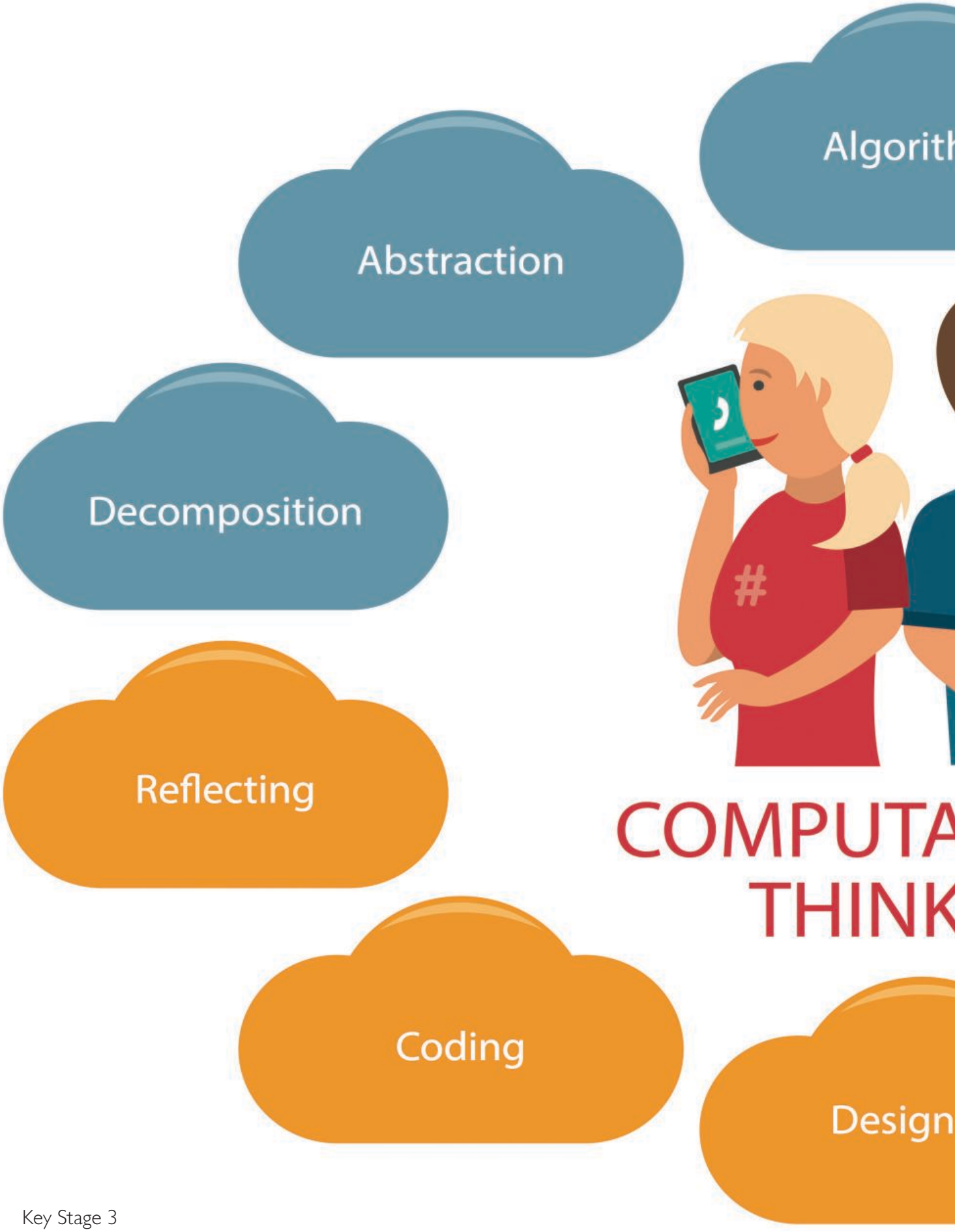
Debugging
finding & fixing
errors

Persevering
keeping going

Collaborating
working together

Approaches

Barefoot gratefully acknowledges the work of Julie Briggs and the eLiM team at Somerset County Council for their contribution to the poster.


Algorith
Abstraction
Decomposition
Reflecting
COMPUTA
THINK
Coding
Design

Key Stage 3


nms
Generalisation
Evaluation
Applying
ATIONAL
KING
Analysing
ing
digital schoolhouse
Inspirational Computing for Kids

Computational thinking in the classroom

Each of the concepts of computational thinking described above (algorithmic thinking, decomposition and so on...) is identified with distinct learner behaviours that may be observed in the classroom.

Algorithmic thinking

Algorithmic thinking is the ability to think in terms of sequences and rules as a way of solving problems. It is a core skill that pupils develop when they learn to write their own computer programs. The following can be observed in the classroom.

- The first set involves formulating instructions to achieve a desired effect.
- Formulating instructions to be followed in a given order (sequence).
- Formulating instructions that use arithmetic and logical operations.
- Writing sequences of instructions that store, move and manipulate data (variables and assignment).
- Writing instructions that choose between different constituent instructions (selection).
- Writing instructions that repeat groups of constituent instructions (loops/iteration).
- Grouping and naming a collection of instructions that do a well-defined task to make a new instruction (subroutines, procedures, functions, methods).
- Writing instructions that involve subroutines that use copies of themselves (recursion).
- Writing sets of instructions that can be followed at the same time by different agents (computers/people, parallel thinking and processing, concurrency).
- Writing a set of declarative rules (coding in Prolog or a database query language).

It also involves:

- Using an appropriate notation to write code to represent any of the above.
- Creating algorithms to test a hypothesis.
- Creating algorithms that give experience-based solutions (heuristics).
- Creating algorithmic descriptions of real world processes so as to better understand them (computational modelling).
- Designing algorithmic solutions that take into account the abilities, limitations and desires of the people who will use them.

Decomposition

Decomposition is a way of thinking about artefacts in terms of their parts. The parts can then be understood, solved, developed and evaluated separately. The following can be observed in the classroom.

- Breaking down artefacts into constituent parts to make them easier to work with.
- Breaking down a problem into simpler versions of the same problem that can be solved in the same way (recursive and divide and conquer strategies).

Generalisation (Patterns)

Generalisation is a way of solving new problems based on previous problem solutions. It involves identifying and exploiting patterns. The following behaviours can be observed in the classroom.

- Identifying patterns and commonalities in artefacts.
- Adapting solutions, or parts of solutions, so they apply to a whole class of similar problems.
- Transferring ideas and solutions from one problem area to another.

Abstraction

Abstraction is the process of making an artefact more understandable by hiding detail. The following behaviours can be observed in the classroom.

- Reducing complexity by removing unnecessary detail.
- Choosing a way to represent an artefact, to allow it to be manipulated in useful ways.
- Hiding the full complexity of an artefact (hiding functional complexity).
- Hiding complexity in data, for example by using data structures.
- Identifying relationships between abstractions.
- Filtering information when developing solutions.

Evaluation

Evaluation is the process of ensuring a solution is a good one: that it is fit for purpose. There is a specific and often extreme focus on attention to detail in computational thinking based evaluation. The following behaviours can be observed in the classroom.

- Assessing that an artefact is fit for purpose.
- Assessing whether an artefact does the right thing (functional correctness).
- Designing and running test plans and interpreting the results (testing).
- Assessing whether the performance of an artefact is good enough (utility: effectiveness and efficiency).
- Comparing the performance of artefacts that do the same thing.
- Making trade-offs between conflicting demands.
- Assessing whether an artefact is easy for people to use (usability).
- Assessing whether an artefact gives an appropriately positive experience when used (user experience).
- Assessment of any of the above against the specification and set criteria.
- Stepping through processes or algorithms/code step-by-step to work out what they do (dry run/tracing).
- Using rigorous argument to justify that an algorithm works (proof).
- Using rigorous argument to check the usability or performance of an artefact (analytical evaluation).
- Using methods involving observing an artefact in use to assess its usability (empirical evaluation).
- Assessing whether a product meets general performance criteria (heuristics).

Examples of algorithmic thinking, decomposition, generalisation, abstraction and evaluation are found across the computing curriculum and across the full range of attainment. Computational thinking is not age dependent and therefore the concepts are not attributed to years or key stages, but they are capability dependent.

Computing At School has published a document called 'Computing Progression Pathways' which sets out the major knowledge areas of computing and gives specific indicators of increasing mastery of the subject relevant to those areas. This document can also be used as an assessment framework. CAS Computing Progression Pathways is based on classroom experiences and identifies the dependencies and interdependencies between concepts and principles as well as between the three subject strands of computer science, digital literacy and information technology.

The Computing Progression Pathways document incorporates the concepts of computational thinking using the initials:

- AL for Algorithm
- DE for Decomposition
- GE for Generalisation and patterns
- AB for Abstraction and representation
- EV for Evaluation

It includes a description of how it can be used to acknowledge progression and reward performance in mastering both the content of the Computing Programme of Study and the concepts of computational thinking. For example, algorithmic thinking is demonstrated not just in the Algorithms and Programming & Development pathways, but also in using search filters (Data & Data Representation orange) and in demonstrating an understanding of the fetch-execute cycle (Hardware & Processing purple). Illustrated below are examples including generalisation - pupils recognising that digital content can be represented in different forms (Data & Data Representation pink).

Data & Data Representation	Hardware & Processing
• Recognises that digital content can be represented in many forms. **(AB) (GE)** • Distinguishes between some of these forms and can explain the different ways that they communicate information. **(AB)**	• Understands that computers have no intelligence and that computers can do nothing unless a program is executed. **(AL)** • Recognises that all software executed on digital devices is programmed. **(AL) (AB) (GE)**
• Recognises different types of data: text, number. **(AB) (GE)** • Appreciates that programs can work with different types of data. **(GE)** • Recognises that data can be structured in tables to make it useful. **(AB) (DE)**	• Recognises that a range of digital devices can be considered a computer. **(AB) (GE)** • Recognises and can use a range of input and output devices. • Understands how programs specify the function of a general purpose computer. **(AB)**

Summary

Computational thinking is an important life skill, which all pupils now need to develop. It is central to both living in and understanding our digitally enriched world. It is therefore a central concept in the new Computing Programme of Study. The framework presented here explains the concept and illustrates it with effective learning experiences that develop the skills, as well as being a way of assessing their development.

There are many existing sources of ideas for classroom activities that potentially develop computational thinking skills in rich ways. Examples include:

- Barefoot Computing http://www.barefootcas.org.uk
- Computing at School http://www.computingatschool.org.uk
- CS4FN http://www.cs4fn.org/
- CS Insidehttp://csi.dcs.gla.ac.uk/
- CSTA Computational Task Force http://www.csta.acm.org/Curriculum/sub/CompThinking.html
- CS Unplugged http://csunplugged.org/
- Digital Schoolhouse http://www.digitalschoolhouse.org.uk
- Teaching London Computing http://www.teachinglondoncomputing.org

For a more intense training experience, the Google MOOC gives detailed and clear explanations of the concepts of computational thinking. The programme can take between 5 hours and 15 hours to complete but provides a rich experience of how computational thinking appears across the whole curriculum and the values it has in commercial activity and learning in general. Computational Thinking for Educators is free

and intended for anyone working with students aged 13 to 18 years, who is interested in enhancing their teaching with creative thinking and problem solving http://goo.gl/7t9vOE

> *"We believe all students should learn computational thinking, regardless of subject, age or access to technology in the classroom. If our students are technology creators, equipped with computational skills, they'll be able to participate and position themselves professionally in a globalized society, helping to solve the biggest challenges using creativity."*

Aida Martinez, Google.

Whatever the source of ideas and ultimately of lessons, it is vital that computational thinking is a central part of them.

Bibliography

The following sources provide valuable insights into the development of computational thinking in computing and in the curriculum as a whole.

BCS, The Chartered Institute for IT. 2014. ***Call for evidence - UK Digital Skills Taskforce.*** Available: http://bit.ly/1Li8mdn [Accessed 15-04-2015].

Department for Education. 2014. ***The National Curriculum in England, Framework Document.*** Reference: DFE-00177-2013. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/335116/Master_final_national_curriculum_220714.pdf [Accessed 15-04-2015].

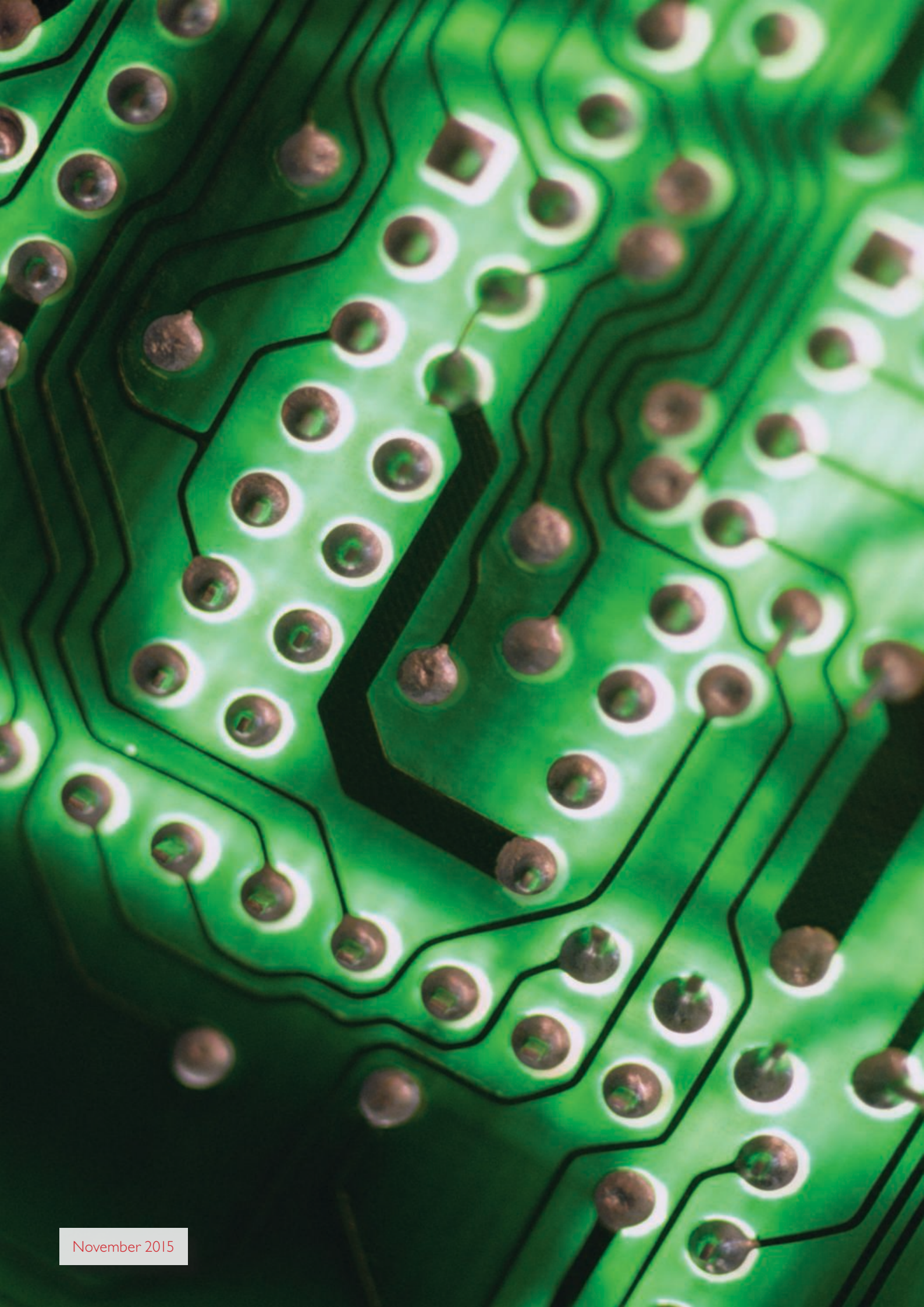
Dorling, M. & Walker, M. 2014. Computing Progression Pathways. Available: http://www.hoddereducation.co.uk/Subjects/ICT/Series-pages/Compute-IT/Series-Box/Progression-Pathways/Progression-Pathways-Grid.aspx [Accessed 28-02-14].

Dorling, M., Selby, C. & Woollard, J. 2015. Evidence of Assessing Computational Thinking. IFIP 2015, A New Culture of Learning: Computing and Next Generations. Vilnius, Lithuania. Available: http://eprints.soton.ac.uk/377856 [Accessed 01-07-2015].

Selby, C. & Woollard, J. 2013. Computational thinking: the developing definition. Definition Available: http://eprints.soton.ac.uk/356481/ [Accessed 01-04-2014].

Wing, J. 2006. ***Computational Thinking.*** Commun. ACM, 49, 3, 33-35. Available: http://dl.acm.org/citation.cfm?id=1118215 [Accessed 15-04-2015].

Wing, J. 2011. Research Notebook: Computational Thinking - What and Why? ***The Link.*** Pittsburgh, PA: Carneige Mellon. Available: http://www.cs.cmu.edu/sites/default/files/11-399_The_Link_Newsletter-3.pdf [Accessed 15-04-2015].



November 2015