Second Draft ICT Programme of Study
Guidance notes

BCS, the Chartered Institute for IT, and Royal Academy of Engineering, Nov 2012

These notes give additional guidance about the thinking behind some of the words and phrases in the Draft Programme of Study. Although written by the working group that drafted the Programme of Study, these notes are not endorsed by DfE. Indeed they should be regarded as just one resource in the rich menu of online resources to support teaching and learning in computing.

Terminology

The Programme of Study deliberately uses technical terms, such as "algorithm", "abstraction", or "data representation", rather than more colloquial forms, to add precision and depth of meaning to a very short document.

Technical terms have precise meanings; you can look them up in Wikipedia and other literature to draw on a rich source of ideas and inspiration. They also usually describe more general ideas than their colloquial counterparts; for example, as students develop they will find that an "algorithm" can be parallel or distributed, something not encompassed by "a sequence of steps".

These guidance notes describe the meanings of some of these terms, as they apply to Key Stages 1-4.

Computational thinking and abstraction

Computational thinking 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 both natural and artificial systems and processes. It is an approach to solving problems that occur not only in writing programs but also in dealing with problems in the physical world. For example, decomposing complex problems into simpler steps and solving these steps one at a time is sequencing, providing alternative solutions to solve different aspects of a problem is selection and solving problems incrementally by taking repeated actions is repetition. Sequencing, selection and repetition are part of the basic toolkit of computer programming but they are also techniques that apply to organising a multitude of common tasks such as planning a journey or decorating a house.

Computational thinking is something that *people* do (rather than computers), and includes the ability to think logically, algorithmically and (at higher levels) recursively and abstractly. Nevertheless, a key property of computational thinking is that it is open to embodiment in programs, and that mechanisation dramatically extends the scale, speed, and "reach" of what can be achieved.

That very scale and speed leads to a key challenge in computational thinking, namely the staggering complexity of the systems we study or build. The main technique used to manage this complexity is **abstraction**. The process of abstraction takes many specific forms, such as **modelling**, **decomposing**, and **generalising**. In each case, complexity is dealt

with by hiding complicated details behind a simple abstraction, or model, of the situation. For example,

- The London Underground map is a simple model of a complex reality — but it is a model that contains precisely the information necessary to plan a route from one station to another.
- A procedure to compute square roots hides a complicated implementation (iterative approximation to the root, handling special cases) behind a simple interface (give me a number and I will return its square root).

Computational thinking values elegance, simplicity, and modularity over ad-hoc complexity.

Modelling

Modelling is the process of developing a representation of a real world issue, system, or situation, that captures the aspects of the situation that are important for a particular purpose, while omitting everything else. *Examples: London Underground map; storyboards for animations; a web "site map"; the position, mass, and velocity of planets orbiting one another.*

Different purposes need different models. *Example: a geographical map of the Underground is more appropriate for estimating travel times than the well-known topological Underground map; a network of nodes and edges can be represented as a diagram, or as a table of numbers.*

A particular situation may need more than one model. *Example: a web page has a structural model (headings, lists, paragraphs), and a style model (how a heading is displayed, how lists are displayed). A browser combines information from both models as it renders the web page.*

Decomposing

A problem can often be solved by decomposing it into sub-problems, solving them, and composing the solutions together to solve the original problem. For example "Make breakfast" can be broken down into "Make toast; make tea; boil egg". Each of these in turn can be decomposed, so the process is naturally recursive. Programming languages allow the programmer to express this kind of decomposition directly using procedures and functions, and real programs almost invariably consist of layer upon layer of procedures, each using the services of the layer below, and hiding complexity from the layer above.

The organisation of data can also be decomposed. For example, the data representing the population of a country can be decomposed into entities such as individuals, occupations, places of residence, etc.

Sometimes this top-down approach is the way in which the solution is *developed*; but it can also be a helpful way of *understanding* a solution regardless how it was developed in the first place. For example, an architectural block diagram showing the major components of a computer system (e.g. a client, a server, and a network), and how they communicate with each other, can be a very helpful way of understanding that system.

Generalising and classifying

Complexity is often avoided by generalising specific examples, to make explicit what is shared between the examples and what is different about them. For example, having written a procedure to draw a square of size 3 and another to draw a square of size 5, one

might generalise to a procedure to draw a square of any size N, and call that procedure with parameters 3 and 5 respectively. In this way much of the code used in different programs can be written once, debugged once, documented once, and (most important) understood once.

Generalisation is the process of recognising these common patterns, and using them to control complexity by sharing common features.

Algorithms

An algorithm is a precise method of solving a problem. Algorithms range from the simple (such as instructions for changing a wheel on a car) to the ingenious (such as route-finding), and cover many different application areas (for example, cookery, drawing three-dimensional graphics; solving systems of constraints, such as a school timetable; understanding images; numerical simulation, and so on).

An algorithm can be expressed as a program in many different programming languages.

There may be more than one algorithm to solve a single problem, differing in their simplicity, efficiency, or generality. For example, to find a path through a maze, one (simple, slow) algorithm might be to simply walk around at random until you find the exit. Another (more complicated) one would involve remembering where had been to avoid going down the same blind alley twice. Another might be to keep you left hand on the wall and walk till you find the exit (faster, but does not work on all mazes, and so less general).

Human factors

It is important to remember that it is people who use IT. Students need to be aware that when designing new IT systems human factors also need to be taken into consideration. Human computer interaction (HCI) includes rules for good system design e.g. having an undo button, checking that it is clear to the user what they need to do at every stage, considering the needs of disabled users etc. These rules can be explored by examining existing IT systems and getting students to evaluate their usability, i.e. what is easy to use and what is more difficult and why, and applying these rules to their own designs.

Students should also be aware of the need to understand the wider social context surrounding developing and deploying IT systems, this can be explained with reference to case studies of systems which failed to deliver the benefits hoped for because of the lack of prior consideration of the disruptive changes then system would bring to the way people work or by exploring how for example social media e.g. twitter and Facebook has change the way people interact with one another both positively and negatively.

Socioeconomic Factors

Digital Divide

It is important to help pupils realise that access to technology can bring benefits and power, but that not everyone has easy access. Lack of access to technologies can disadvantage particular groups or individuals within societies. Exclusive access to data and/or technologies can give advantage to organisations or individuals.

Gender and inclusion

It is important to counter the stereotypes often associated with information technology and computing, e.g. that it is a male-only field. Efforts should be made in, for example, the selection of historical or contemporary case studies to reflect the positive contributions of female practitioners, for example Ada Lovelace, Grace Hopper or Dame Wendy Hall.

Projects topics should also be carefully considered to be inclusive to both genders.

Assistive technology

As with other areas of the curriculum, ICT can be made more accessible to children with some special educational needs or disabilities through the use of assistive technology, from adapted mice or keyboards to screen readers and Braille displays. Within the curriculum, pupils might evaluate whether digital content is accessible to users with SEND, and learn about assistive technology as examples of 'forms of input and output' at KS2 and 'hardware and software components' at KS3.

English as an Additional Language

Technology can also facilitate the inclusion of pupils learning English as an additional language. The user interface of the operating system or application software can be set to languages other than English, and, for example, Scratch programs can be written in languages other than English. Machine translation technology allows instructions, questions and responses to be translated automatically, often with a good degree of accuracy between common languages; teachers may wish to explore the process and accuracy of such services. Machine translation may also be useful for project work in which pupils learn about the opportunities offered by the Internet.

ICT as a preparation for professional life

It is critical, as part of teaching and using ICT within and across the curriculum, to deal with the false perceptions that careers in ICT are only for 'geeks' or that there is a shrinking profession. ICT is a wide ranging, exciting, varied and rich career, providing constant opportunities for growth and involvement in careers across the economy. It is essential to promote this prospect and the added value these skills will provide both them as employees and to their employers no matter their choice of career.

Aims and purpose

- **"Digital literacy, information technology, and computer science".** Each of these elements should be visibly present at every stage in a pupil's ICT education. However, the three are closely related, overlap, and should not be thought of as "silos" into which lessons can be categorised. For example, use of a spreadsheet might be considered information technology, but use of formulae crosses over into an exercise in programming. To take another example, the PoS asks that pupils are taught about how search engines ranks searches (Computer Science). The context is likely to be about finding relevant information to help with a problem; clarity about the problem can make them better at searching (IT). The link can then be made to the wider context of the need for efficiency in finding information for organisations and to discussion of ethics when helping the pupils understand that ranking is based on payments as well as popularity (DL).

- **"Societal value".** This term invites teachers to help their students reflect on the profound effects that pervasive information technology has on the society in which we live.
- **"Create purposeful and usable artefacts"** means that there should be a reason, clearly identified to the pupils, why a system or artefact is created. The pupils should be able to articulate this reason. Furthermore, it is good practice that there should be an identified reason, made explicit to the pupils, for why they are learning any particular thing. To support this it is important that learning should be set within a context.

Key stage 1

- **"Work individually and collaboratively".** A fundamental aspect of computing and information technology in the workplace and beyond is that it is collaborative. Systems are invariably built by teams; pair-programming and peer code review is well-established professional practice; developing software components that can be used by others is the key to modularity; and so on. It is not appropriate to try to *teach* professional software engineering practice at school, but it is important that from the earliest age pupils should have the *experience* of describing their digital creations to others, and working together to develop, critique, and improve them.
- "**Work creatively**". Unlike natural science where we discover facts about the natural world, computer science and information technology are entirely the result of human creativity. It is through the creative processes of making and refining programs and digital media that pupils acquire for themselves a deep understanding of how technology works and the principles that underpin it. In ICT, as in other creative disciplines, theoretical understanding is developed in parallel with increasing practical capabilities.

  Valuing creativity also allows for the ICT to be taught in collaboration or sympathy with other arts subjects, creating a digital basis for personal expression, as well as further academic progression or work in the digital or creative economy (e.g. games, graphics, animation, and interactive technology).
- "**Playfully**". A playful attitude towards technology seems to encourage pupils to develop independence, confidence and understanding, and sits in a long tradition including Froebel, Papert and, more recently, the work of MIT's Lifelong Kindergarten Group. An attitude of playful experimentation and exploration characterizes the work of many software developers and computer scientists.
- "**A range of devices"** here indicates digital devices accepting input, producing output and operating according to a stored program, including desktop computers, mobile phones, digital cameras and smarter photocopiers; the operation of these devices are controlled by computers. Programmable devices are those where the user themselves can create or alter the program, such as a desktop or laptop computer, smart phone or tablet.
- **"Algorithm".** At KS1 an algorithm is likely to be no more than a simple sequence of steps (e.g. open bread bin; cut slice; put bread in toaster; wait; take toast out; eat it).
- **"Simple programs".** These may be sequences of instructions for controlling the movement of a robot (eg Bee Bot or ProBot) or an on screen turtle or sprite.

- **"Organise, store, manipulate, and retrieve data"** includes the efficient and effective use of the computer file system or equivalent cloud-based storage.

Key stage 2

- **"Collecting, analysing, evaluating and presenting data and information",** should make use of the specific capabilities of IT to extend these activities through provisionality, interactivity, automation and increased speed, capacity (e.g. using very large data sets) and range, as well as through joint projects mediated via the Internet. Information is understood here as data to which a specific meaning has been attached through a process of interpretation.
- **In programs**
    - **Sequence** means putting instructions in order to be executed one after another.
    - In a **selection** structure, a question is asked, and depending on the answer, the program choose between two or more possible courses of action. At KS2, selection should include the if..then..else statement. (E.g. **If** the sprite is touching a wall **then** bounce back, **else** move forward.).
    - **Repetition** means repeating a sequence of instructions a certain number of times, or until some specific result is achieved. In programming terms this means loops of all kinds, such as repeat, for, while, until etc. (e.g. move dog 1 step forward; repeat until dog is in kennel then stop).
- **"Various forms of input and output".** Keyboard, mouse, sensors, screen, speakers, microphone**.**
- **"Understand computer networks"** means, at this stage, knowing that a network consists of one or more computing devices connected together, using shared protocols, so they can share data and resources. "**Internet services**" means things like school blogs, web-based spreadsheets, language translation services, pixlr, or email.
- **"Opportunities for communication and collaboration"** is one of the most immediate and visceral impacts of the internet on pupils' lives. Students should personally experience opportunities to communicate and collaborate both internally within the school and, where possible, externally. That experience should in turn inform, and be informed by, reflective discussion about issues such as respectful communication in a context where body language is absent; cultural differences; privacy; and safety.
- **"How they change over time".** As well as thinking about how technology has evolved during their parents' or grandparents' life times, pupils should be taught about some of the key figures and events in the development of ICT, many of them from the UK, including
    - Charles Babbage, Ada Lovelace and the difference and analytical engine
    - Alan Turing, the Turing Test, Turing machines, Enigma and the work of Bletchley Park
    - Tim Berners-Lee and the invention of the Web.

Pupils should also consider some implications of the continuing technological innovation in their and others' lives, perhaps creating digital content to illustrate how they think technology may further change over a period of ten or twenty years.

- **"Appreciate how [search] results are selected and ranked"**. Internet search engines must choose which order to present result in. The "page rank" algorithms used to do so are interesting in their own right, and elementary versions are accessible even at KS2 (e.g. pages to which many other pages point are more highly ranked). Given the enormous influence of search engines, other non-technical factors come into play, notably advertising and censorship.

Key stage 3

- "**Combine multiple applications**". As confident and independent users of IT, pupils should make reasoned choices from a range of applications, combining these effectively to achieve complex goals, including solving problems and creating digital content, to take into account the needs and expectations of their intended users or audience. For example, pupils might start by capturing audio, video and image content using a smart phone, edit images and audio and combine these with video using editing software on a desktop PC, before uploading the edited video to a web service and embedding the streamed video within a website of their own design.

  Another example: Machinima, which is the use of online collaborative gaming environments to create a scripted story that is captured and edited onto video. A well-known example from the game Halo is the 'Red vs. Blue' series. Multiple applications are used by a group to write, story-board and then act out a script in an online collaborative game, stream their efforts in real-time onto a PC, edit the video and add sound/music/VFX, convert to a standard video format and finally upload onto a public repository such as YouTube.

  A third example would be to use programming languages such as Python to create small scripts/macros that can add new items/behaviours to a computer video game. e.g. creating Python 'mods' for Minecraft.

- "**Revise, repurpose**" Pupils should have the opportunity to work with their peers', open source, creative commons and public domain content in a range of media. They should learn how to work creatively, using others' digital work as a starting point whilst respecting copyright. They should also learn how content can be selected, edited and combined with other content, perhaps in other media to produce work fulfilling a different purpose or meeting the needs of a different audience than that on which it is based.

- **"Design, use, and evaluate computational abstractions that model the state and behaviour of real-world problems"**. See "Modelling" above. Here is a concrete example. Suppose you are trying to predict the population of rabbits and wolves on an island. We might model the **state** at any moment by two numbers: the number or rabbits and the number of wolves. We might model the **behaviour** of the rabbits by saying: every six months the rabbit population increases by a factor of 1.3 due to breeding, and decreases in proportion to the wolf population by the wolves eating them. And so on. This model is an **abstraction** because it ignores masses of detail (fur, blood, etc), including some that might be relevant (food supplies for the rabbits). But the model is **computational** in the sense that we can write a program,

or a spreadsheet, that explores how the two populations evolve under various assumptions of breeding and predation rate.

Simulation and modelling of real-world situations offers a rich source of motivating examples of computational thinking, and helps to make concrete the idea of computation as a lens through which to understand the world (see "Purpose" section of the POS). Simulations can be used at every key stage, not only KS3, and may involve spreadsheets, games, or full programming languages.

- **"Be able to use at least two programming languages, at least of one of which is textual"**. Key objectives here are to learn that programming can be done in a wide variety of ways; that different languages support different styles; and that once you know two or three languages, learning more languages becomes easy. When pupils see more than one language, they will begin to generalise, and learn lessons that are not tied to the particular syntax or semantics of any one language.

  There is a broad range of possibilities, depending on the resources of the school:

  - Visual languages such as Kodu or Scratch are an excellent way in.
  - The formula language of spreadsheets is certainly a textual programming language, albeit a limited one
  - Several languages (e.g. Logo) control the behaviour of a turtle or robot, but introduce the idea of a textual language.
  - HTML and CSS have a much more specialised and declarative flavour (they do not specify a sequence of steps), but they certainly embody the ideas of precise syntax, and finding and correcting errors; and they lead naturally to the world of Javascript.

- Schools should offer opportunities to learn more fully-fledged languages supporting procedures, data structures, and types, and that have rich eco-systems of libraries and programming environments. Examples: Greenfoot (Java), Small Basic, Visual Basic, etc.**"How they interact and how they affect cost and performance".** Here the PoS goes beyond asking for simple recall of a block diagram of a computer (CPU, RAM, disk, network). For example, RAM is fast but volatile (loses data if power is lost) expensive; disk is cheap and non-volatile but slow. Even RAM is not fast enough to keep up with the CPU, so a cache sits between the CPU and RAM to keep frequently used data in particularly fast storage. (Analogy: keeping the book you are reading on the bedside table.) It may be helpful to have order-of-magnitude numbers on these cost/performance trade-offs; for example a CPU may take hundreds of cycles while the RAM performs a single memory fetch. Changes in performance over time (e.g. Moore's law) are also relevant here because they have historically been so large.

- **"How instructions are executed"**. A basic understanding of the von Neumann fetch-execute model is intended here. A CPU fetches one instruction at a time and executes it; instructions are held in memory, just like any other data; the program counter gives the address of the next instruction to execute. Simple analogies may be useful here; for example, a knitting pattern is a "program" and a knitting machine is the "computer" that executes it; the knitting machine carries out the knitting instructions in a totally simple-minded way. Despite this simplicity, the resulting garment is a work of art whose creativity is all in the knitting pattern.

- **"Explain how data of many sorts can be represented and manipulated in the form of binary digits"**. Pupils learn the base-2 numbers system in Mathematics in Year 6, so they should be ready to understand data representation by KS3. Pupils should be able to count and do simple arithmetic in binary, but they should also explore the specifics of some particular data representations. Examples: numbers, text, images, sound.
- **"Understand how computers can be connected to, monitor, and control physical systems"**. There are clear opportunities here for linkage to D&T and Science.

Key stage 4

The working party found particular challenges in drafting a programme of study for Key Stage 4.

- By the time they reach Key Stage 4 pupils should have the opportunity to specialise somewhat, as they do in Science. We think it essential that students taking such KS4 qualifications (e.g. in Computer Science, or in Digital Media) be considered to be fulfilling the statutory KS4 ICT programme of study, even though it may not cover the whole range of ICT.
- Other students will choose to focus their attention elsewhere, and will choose to take no KS4 qualifications in ICT. In the past some schools, obliged by law to provide ICT for all, have provided a token but compulsory ICT course for unwilling students. Such classes have little purpose, and drag the entire subject into disrepute.

The working party's solution is as follows

- The draft Programme of Study specifies that at KS4 "**All pupils must have the opportunity to take qualifications in aspects of information technology and computer science, which lead to progression to higher levels of study or a professional career".** These words are carefully chosen:
    - "**Opportunity**" means that pupils must be able to take KS4 qualifications in ICT subjects, but should not be obliged to do so. "Option blocks" at KS4 are already very constrained and highly cherished. We do not think it is right to require every pupil to take a KS4 qualification in ICT.
    - "**Aspects**" is intended to mean that a KS4 qualification might not cover the whole range of the subject as studied in KS1-3. For example: a GCSE in Computer Science, Information Technology, or Digital Media; or combinations thereof; or a more professionally-oriented qualification in Network Management or Database Administration.
    - "**Qualifications**" means that, in a period when pupils are very focused on GCSEs and Diplomas, schools cannot merely offer a token course in ICT leading to nothing of value.
    - "**In information technology and computer science**". We would like every good school to offer a *range* of KS4 qualifications in ICT subjects, including both CS and IT variants; but we are aware that not every school can do so immediately.

      If a school is unable to offer a range of qualifications, then they must provide the opportunity to study a qualification covering topics from IT, CS and DL

that would allow a student to progress to IT, Computer Science or Digital Media post 16. This parallels the Science curriculum where students taking joint or single Sciences have the pre-requisite knowledge to study full Physics, Chemistry and Biology at a higher level (e.g. A level).

- "**Progression to higher levels of study or a professional career". I**n line with the recommendations of the Wolf report, it is essential that any qualification offered by a school must be valued by employers and higher education institutes, and must support progression into A levels, further education, or a professional career.

- Students who choose not to take any such KS4 qualification should nevertheless develop the capability and knowledge they acquired in KS1-3. This requirement is stated briefly, and schools are free to deliver is in a variety of ways. We believe that delivering it across the curriculum in other subjects would be perfectly acceptable.

Technology Enhanced Learning

At all Key Stages, information and communication technology should be used to enhance teaching and learning right across the curriculum: call it Technology Enhanced Learning (TEL). When there is a clear focus on learning rather than technology, systems such as the Web, interactive whiteboards, virtual learning environments, video conferencing, blogs, wikis, podcasts, video, and mobile devices can have a transformative impact on both learning and teaching. Pupils' use of such technology both draws on and enhances their digital skills.

The purpose of using technology in this way should be to improve learning in that subject, and not a back-door way to teach ICT. The technology *serves* learning; it is not the *object* of learning. It follows that:

- The Programme of Study for ICT focuses only on ICT as a discipline in its own right, and not on TEL.
- The use of technology to support learning in other subjects (English, say, or Geography) should be assessed by Ofsted as part of the school's teaching and learning in that subject, not as part of its delivery of ICT. Nevertheless, it would be extraordinary for any subject to make no use of technology - which should be recognised in the statutory requirements for these subjects within the National Curriculum.
- The extent and nature of the use of technology in other subjects should be driven exclusively by the needs of those subjects, and not by the needs of the ICT curriculum; except, of course, if a school consciously chooses to teach some aspect of the ICT programme of study through those subjects.