Orienting learners and teachers in introductory programming classes: the ABC Framework

Quintin Cutts
University of Glasgow
Glasgow, Scotland
Quintin.Cutts@glasgow.ac.uk

ABSTRACT

This practice paper presents a framework that has been successfully used in introductory programming classes to orient students to the nature and purpose of programming, and teachers to the multiple aspects of programming education. Orientation was one of the five key difficulties for novices identified in du Boulay's landmark 1986 paper, and it can still be an issue for both incoming students and new computing teachers. The framework, known as *ABC*, presents computing as a modelling activity, with: a multitude of possible problem/task domains, or Application Areas (A); sets of Building Blocks (B), such as programming languages and other computing systems, which can be used for model building; and a set of skills enabling the Creative Construction (C) of a solution, or model, of a problem/task in a particular application area, using a particular building block system. How the ABC Framework can be used to help orient students and structure learning and teaching is presented, as well as insights derived from students and teachers of an introductory programming course where this approach is used.

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**.

KEYWORDS

introductory programming, orientation, difficulties, framework

ACM Reference Format:
Quintin Cutts. 2024. Orienting learners and teachers in introductory programming classes: the ABC Framework. In *Computing Education Practice (CEP '24), January 05, 2024, Durham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3633053.3633063

1 INTRODUCTION

This practice paper concerns the adoption of the ABC Framework, presented here, for understanding and organising the various aspects of programming, and of learning and teaching programming. ABC points to much of the foundational research in learning to program over the years and so will most likely seem familiar, particularly in relation to du Boulay's and Soloway's early works, e.g. [2, 8]. In its very simplicity, hopefully, lies ABC's greatest strength, enabling us to integrate much of what we know about programming into a coherent and immediately understandable framework.

Sections 2, 3 and 4 present our context and drivers for change, an explanation of ABC, and its use in our course. Section 5 presents students' experience and teachers' use of ABC, giving deeper insight into its value. The paper concludes with next steps.

2 CONTEXT AND THE NEED FOR CHANGE

Our context is an "Introduction to Computational Thinking" course with an enrolment of around 150 students who have no prior experience of programming (often referred to as a CS-0 class). Around half the class have declared that computer science will be their degree subject; for the rest, this is an elective option. The class runs for 11 weeks with six contact hours per week. The course is primarily a programming course, as an example of computational thinking that captures the key aspects of the phrase as identified by Wing in her later 2008 article [10] including both abstraction and automation: learning to write simple programs to control a computer gives experience in abstraction and automation.

Our programming course, like many others, and like most text books, has tended in the past to be structured by the programming language constructs, rather than by the problem domains to which those constructs could be applied. For an intending computer science student, it is perhaps acceptable to focus on the mechanics of programming languages; but for an elective student, taking the course as an option alongside their intended degree subject, the value or potential application of what they are learning about computation should be emphasised.

Furthermore, the actions that we take as programmers, the actions of computational thinking, are perhaps not all explicitly identified. We may have talked in our course about top-down decomposition, or patterns, or debugging – but we didn't have good frameworks to introduce these topics properly. Via a combination of live-coding and other forms of worked example, along with the students working on programming tasks, we have hoped that these problem-solving, or computational thinking, skills are acquired. But we know that they aren't, for many of our students.

In short, we recognise a need to provide more motivation and orientation for our students, as well as a framework for discussing the activities involved in computational thinking / programming.

3 THE ABC FRAMEWORK

ABC has been born out of a lack of easily understandable connective tissue between a stated high-level goal such as learning to program, and the low-level knowledge and exercises presented to students. Such a structured connection is needed both for teachers, to ensure that they are attending to all aspects of the knowledge and skills

CEP '24, January 05, 2024, Durham, United Kingdom

ACM ISBN 979-8-4007-0932-6/24/01…$15.00
https://doi.org/10.1145/3633053.3633063

involved in programming, and for students, to help orient their learning and make sense of all the low-level detail.

At the heart of the Framework is the recognition that programming (as a key exemplar of computational thinking, that is, what we do as computing people) is fundamentally a *modelling* activity. Implicit in modelling is the *task domain*, aspects of which we want to model, and the *modelling domain*, aspects of which we are going to use to build models, and finally, the *modelling skills* that we use to analyse tasks in their domain, drawing out key attributes, which we then fashion into a model using the tools of the modelling domain. These three are the basis of the ABC framework, so called when the two domains and one set of skills are renamed, respectively, *Application Areas*, (A), *Building Blocks*, (B), and *Creative Construction*, (C). These are depicted in Figure 1, where A and B are separate worlds, divided by the horizontal line, and C is a process that involves moving between the two worlds.

By making the three aspects explicit, teachers are prompted to focus on application area and building blocks both separately and in combination, as well as creative construction: do the students thoroughly understand the A they're working in (necessary for effective C); is it a personally interesting area; have they thoroughly learned about the individual building blocks, B; do they see the relationship between recurring sub-tasks in the various As they're seeing and related templates in the B they're using; is time being spent on the skills involved in C?

We expect ABC to help address two of du Boulay's five landmark difficulties in learning to program [2], namely orientation, and the pragmatics of programming. The framework also precisely locates two more – notional machines and notation – in the B domain.

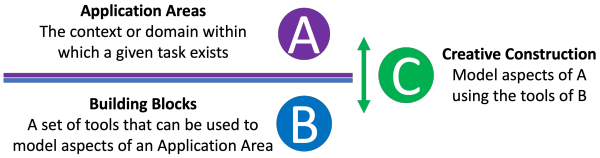

Figure 1: The ABC Framework

4 WHAT IS THE NEW PRACTICE?

The course has been changed in the following ways based on thinking around the ABC Framework:

Structured by application area. The high-level structuring of the course is now via five topic areas that introduce different application areas - hence turning around a traditional focus on programming language constructs (or building blocks). The delivery of each topic requires the use of a steadily increasing set of programming language constructs. Both the topics and the constructs they introduce are given in Figure 1. The aim is that students will appreciate the relevance of programming across their lives, even at the relatively simplistic level covered in an introductory course.

Made students aware of ABC, with reminders. The ABC is introduced in the first lectures of the course as a way of explaining the nature of computational thinking and programming as modelling activities. ABC is initially exemplified with modelling using Lego, mathematical modelling (creating a model to predict the number of cars in a road network), and modeling a human information process (carrying out long multiplication with pencil and paper). Additionally, the learning of mathematics in schools is used to emphasise the difference between application areas and building blocks, by highlighting how so much of maths was simply about learning how the building blocks worked, with little to no mathematical modelling of real world tasks or problems. The 5 different application areas (topics) are introduced to the students, and our use of Python as the building blocks system. As each different topic is reached in the course, students are reminded of the ABC and our progress.

Mapped the development of building block knowledge. A single-page map of all the building block concepts to be learned in the course is introduced early, with most greyed-out and only the ones that have already been covered in black type. The aim is to make clear the distinction between the building blocks world that they are learning about, and task/topic specific understanding that may be needed in the particular topic currently being studied. Furthermore, we amplify the idea of a "restricted toolset": at any point in the course, the programming tasks we present to the students should be solvable with only the building blocks currently covered. We discuss the value of learning to work in this way – to make do with the tools available – because each set of building blocks (that is, each programming language, for example) will have *different* blocks, and one must learn to work with what one has.

Highlighted commonalities across building block systems. As well as noting the differences between building block systems, we also assure the students that although they are only learning Python in this course, there is much overlap with other programming languages. While we are confident that most instructors will say something like this, the introduction of the ABC gives an easy language to talk about this.

Used modelling as the foundation for code comprehension. We use the clear distinction between the application area and the building blocks domain to introduce the two keys ways in which program code must be understood – on the one hand, it is a collection of instructions that will be applied to a (virtual) machine, while on the other hand, it is a model of the activity described in the application area. These are the Structural and Functional aspects, respectively, of Schulte's Block Model of code comprehension [7].

Emphasising the skills involved in creative construction. The C part of the framework encourages the explicit identification of skills involved in developing programs. Two examples are: identification and discussion of patterns / plans as a way of linking the task and modelling domains, drawing on Soloway's notions of goal and template respectively as making up a plan [8]; and debugging, beyond syntax errors, as an exercise in comparing the intended program model in our head with the actual program model we have created and finding out how these differ. Of course, if the two models *are* the same, yet we aren't getting the right output, then we have analysed the problem incorrectly and come up with the wrong intended model.

Table 1: Application Area Topics and related Building Blocks

Application Area Topic	Description	Building Blocks
Processing text - You are a data wrangler	Manipulating tabular data in text files for data cleaning and simple operations	Expressions, variables, control flow (sequence, selection, repetition), file handling, interactive texual I/O, lists
Generating graphics - You are a games designer	Drawing 2D colour graphics incorporating data from files. Simple guessing games	Function calls, actual parameters and return values, side effects. Using a library (Turtle graphics)
Organising data - You are a data scientist	Modelling more complex data and performing detailed data analysis	Dictionaries. Complex data models using both lists and dictionaries
Developing apps - You are a software engineer	Introducing the challenges of working with larger projects	Creating functions, seeing Turtle as a framework, event loop and callback functions
Your task - The Free Programming Project	An opportunity for students to pick their own application area and task within it	Consolidating the learning of all constructs encountered so far, integrated into the solution of a unique task

Included the students' own application areas of interest. The fifth topic area is an opportunity for the students to pick their own application area and task within it. This final topic gives the students a level of agency and ownership within the course, and enables them to develop their skills in creative construction, working on an entirely new task, developing an appropriate solution in the task domain and then modelling that in Python.

Made (some) tutors aware of ABC. Some of the tutors were aware of the ABC framework from an optional tutor training course run by the department, and were able to work with it during their tutorial/laboratory sessions. This continuity of approaches across the different teaching staff seen by students is important.

5 DOES IT WORK?

The value of ABC was discussed with two tutors and two lecturers on the course, and informal and formal feedback was sought from students, with input from eight students, some at the end of one run of the course, others a few weeks into the next. Standard ethics procedures were followed in acquiring and processing this data. This is obviously not a rigorous evaluation, but the comments of instructors and students indicate high value in the approach.

The Framework has helped instructors see the "big picture" of programming education: *"[ABC] hasn't dramatically altered my perspective on programming, but it has helped me better realise the complexity of the programming process, and how to break it down"* and *"If I were to design a programming course from scratch, I would draw considerably from the ABC Framework. I see it as a structured curriculum approach that mirrors programming as a modelling process, giving equal emphasis to problem domains, modelling domains, and the development of problem-solving and modelling skills."* Ensuring that students see modelling as crucial to the true essence of programming was emphasised.

The instructors noted the importance of making the application area explicit, both for interest and programming skill. For example, one said *"I like ABC from the point of view of a teacher, because it makes me think explicitly about application areas – and it's important to be diverse when picking them – to arouse interest and enthusiasm in students with a range of backgrounds."* Another noted *"If a student does not know anything about A and does not know how to solve the problem without the computer, no understanding of Python or any other language is going to enable them to solve the problem."* This underlines the importance of considering whether given tasks are set in application areas familiar or attainable by students. For example, success in maths is correlated with success in programming: is this because maths and computing are related, or because many of the *tasks* in programming classes require mathematical knowledge?

On the building blocks, one of the tutors wrote *"the students need to know what tools they have available to solve it and how they work. This knowledge is separate from the problem they are solving. It does not change from problem to problem. This knowledge is constant for all exercises. A for loop always behaves the same way. They need to have a good understanding of the tools in order to use them correctly, predict how they will behave and enable them to debug when the result is not correct."* There is a clear emphasis here that we need to understand a building block thoroughly before we can use it in creative construction.

Non-productive student behaviours were identified and explained in terms of the ABC components: *"Students tend to read a problem and jump right into trying to solve it (go straight to C) without first considering A (do they know what the problem is, and how it can be solved without a computer) or thinking of all their tools in B, (they tend to take the first tool that comes to hand and use it for the job without considering what the tool does or doesn't do and whether it is the correct tool for the job or if there are better ones.) This leads to hacky code where the student keeps writing code until it forces the output to be correct (or at least appears to be)"*

This need for students to pay attention to what they know was amplified by the second tutor who talked about students mid-course who have trouble writing a program from scratch. He described how paying attention to A and B was a crucial part of the creative construction process. First, *"What we would do is try and nail down, 'what does the program need to do' and write it all down."* This was about setting objectives for the program to achieve, a C activity but requiring sound A knowledge. Action around the building blocks was more explicit: *"I got them to write down EVERYTHING they knew how to program. They were relatively new programmers so really we *could*, sort of, just put everything they knew onto a bit of paper: for loops, while loops, functions, etc. This became a bit more interesting when they started thinking of it as actual building blocks, so one would say something like, 'I can use a for loop to count occurrences of an element in a list' which we put down as another *thing* we could do."* Finally, with their understanding of A and B clear, they could

much more successfully tackle C: *"Then, with the objectives on one bit of paper and a whole load of blocks on another, we put them next to each other and said 'how can we fit these together: how can we use some of the blocks we have on B to match the list of tasks on A?' This part was the part they were always the most scared of, but they were always able to do it quickly and easily because they had everything they needed in front of them."*

Student feedback mainly centred on the holistic nature of the approach. For example, *"I think the way the course is taught, how we learn to use python for mathematical purposes, then expanding with lists and dictionaries learning how to use it for practical everyday situations, then the turtle library for animation/games etc, makes one understand it is multipurpose. This structure is brilliant! In the span of four months one sees its broad practical side, how with one program/system one can do so much."* This student clearly valued the explicit attention paid to different application areas, and the comments match well to what one of the instructors said about ABC: *"it has the potential to foster a deeper appreciation of the nature of programming and its interconnection with diverse disciplines, making students understand that programming is not an isolated skill but an adaptable tool applicable to multiple domains."*

New students were asked whether their view of programming had changed as a result of ABC. One wrote *"It reminds me of '知行合一' in Chinese, — it means 'unity of knowledge and action' in English. In fact, we have already studied [a lot of] math so far, [so] we have enough blocks to use, but we don't know how to build a structure with actual [value]. Maybe it's not the best, but it's a good start point of engineering journey."* Another said their view had changed a lot: *"My only previous experience was on Scratch in a very unstructured class so I hadn't previously considered problem solving (A) we just memorised/copied (B) from the worksheet and B was already partially done with those blocks you drag about."* And a third (who appears to have programmed before): *"Yes, a little – it helped me put words to how I've thought through programming problems."*

Students noted enjoyment of the course, although this can't be directly ascribed to ABC: *"It was such a pleasant surprise to see that there is a fun way to learning computing hahaha!"*, *" I wanted to express how much I enjoyed the class this semester. Before coming to the university last year I would have never considered to do computing science ... but when looking back at the beginning of the semester I am amazed to what my skills have developed to"*, *"Incredibly accessible to students who have never coded before. Taught the skills at a very manageable and stimulating speed"*, *"Even though I had not done any coding before, the skills were built up slowly and all the information that I had learnt tied together beautifully in the end with the final project which gave the freedom to the learners to choose their favourite way to show what they had learned this semester"*, and finally, *"I had attended 2 years of a software development course 10 years ago, and left it feeling like I couldn't solve anything. After 1 semester here I already feel much more capable than I ever thought I would. I have never had a fixed mindset, but I did not expect the way I solve problems to change this much in 3 months."*

6 WHERE DID THE IDEA COME FROM?

The ABC Framework is a simplification of a national framework for schools computing education [1]. Broadly the same structure is also evident in the *Realistic Mathematics Education* (RME) movement, in which the real world application of mathematics, or its use as a modelling tool, is balanced against mathematical modelling concepts and detail [9]. As noted earlier, much of what we know about programming education can be sign-posted from ABC, to help both teachers and learners. As well as Soloway's and du Boulay's work, some examples are: Papert's *microworlds* [6] to represent different application areas; pattern-oriented instruction [5] and sub-goal labelling [4] for learning and teaching of aspects of creative construction; and tracing [11] and notional machines [3] in the building blocks area.

7 WHAT NEXT?

Following on from the tutor comments above, the full tutor group will be made aware in future of the ABC approach used in the lecture side of the course. Also, the structure of the C aspects, while better than before, still needs further formalisation.

ACKNOWLEDGMENTS

ABC originated in discussion with Richard Connor, and later, Judy Robertson. Maria Kallia and Jeremy Singer enriched the idea in class. Steve Draper provided valuable comments for this paper.

REFERENCES

[1] Richard Connor, Quintin Cutts, and Judy Robertson. 2017. Keeping the Machinery in Computing Education. *Commun. ACM* 60, 11 (oct 2017), 26–28. https://doi.org/10.1145/3144174

[2] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[3] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. 2020. Notional Machines in Computing Education: The Education of Attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 21–50. https://doi.org/10.1145/3437800.3439202

[4] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-Labeled Instructional Material Improves Performance and Transfer in Learning to Develop Mobile Applications. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 71–78. https://doi.org/10.1145/2361276.2361291

[5] Orna Muller. 2005. Pattern Oriented Instruction and the Enhancement of Analogical Reasoning. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, WA, USA) *(ICER '05)*. Association for Computing Machinery, New York, NY, USA, 57–67. https://doi.org/10.1145/1089786.1089792

[6] Seymour Papert. 1981. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York, NY. http://www.amazon.fr/exec/obidos/ASIN/0465046274/citeulike04-21

[7] Carsten Schulte. 2008. Block Model: An Educational Model of Program Comprehension as a Tool for a Scholarly Approach to Teaching. In *Proceedings of the Fourth International Workshop on Computing Education Research* (Sydney, Australia) *(ICER '08)*. Association for Computing Machinery, New York, NY, USA, 149–160. https://doi.org/10.1145/1404520.1404535

[8] Elliot. Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (sep 1986), 850–858. https://doi.org/10.1145/6592.6594

[9] Marja Van Den Heuvel-Panhuizen. 2003. The didactical use of models in realistic mathematics education: An example from a longitudinal trajectory on percentage. *Educ. Stud. in Mathematics* 54 (2003), 9–35.

[10] Jeanette M Wing. 2008. Computational thinking and thinking about computing. *Phil. Trans. R. Soc. A.* 366, 1881 (2008), 3717–3725.

[11] Benjamin Xie, Greg L. Nelson, and Amy J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 344–349. https://doi.org/10.1145/3159450.3159527