Fostering Computational Thinking skills with a Tangible Blocks Programming Environment

Tommaso Turchi
Human Centred Design Institute
Brunel University London
United Kingdom
tommaso.turchi@brunel.ac.uk

Alessio Malizia
Human Centred Design Institute
Brunel University London
United Kingdom
alessio.malizia@brunel.ac.uk

***Abstract*—Computational Thinking has recently come into the limelight as an essential skill to have for the general public and in many disciplines outside of Computer Science; it encapsulates those thinking skills integral to solving complex problems using a computer, thus widely applicable in our technological society. It is influencing how people think about the world and shaping research across many different areas. Several public initiatives such as the Hour of Code have succeeded in introducing it to millions of people of all ages and backgrounds using Blocks Programming Environments like Scratch, thanks to their ability of lowering the barriers of programming and enhance learning. In this paper we present our arguments for fostering Computational Thinking skills using a Blocks Programming Environment augmented with a Tangible User Interface, namely by exploiting objects whose interactions with the physical environment are mapped to digital actions performed on the system.**

Keywords—Computational Thinking; Tangible User Interface; End-User Programming

I. Content and claims

From the simple task of turning on the engine of your car to the vastly more complicate process surrounding the management of a flight, it is unmistakably clear how much our society depends on software: technology surrounds every aspect of our lives, thus computational literacy and coding are becoming much needed skills to possess for an ever wider audience. Moreover, since people will always strive to play a more active role in their lives, this need is undoubtedly becoming their wish, as evidenced by initiatives such Code.org's Hour of Code[1]: this successful global initiative involves millions of students of different ages starting with 4-year old and aims at introducing coding to a wide audience with different backgrounds. This and similar initiatives – as with coding itself – are not just about programming though: they endeavor to foster Computational Thinking skills, namely all those thinking abilities lying at the hearth of Computer Science, such as problem solving, abstraction, and pattern recognition.

Wing [1] first defined Computational Thinking (CT) as a set of thinking skills, habits, and approaches that are needed to solve complex problems using a computer and widely applicable in our information society. It covers far more than programming itself, including a range of mental tools reflecting fundamental principles and concepts of Computer Science, such as abstracting and decomposing a problem, recognizing similar ones and being able to generalize their solutions.

In and of itself, programming has proven to be an excellent way of developing CT skills [2]; to this end, many End-User Development (EUD) tools and techniques have been employed with the aim of lowering programming barriers and fostering the spread of computational literacy, often with mixed results [3].

One of if not the most well-known and effective EUD technique for lowering programming barriers and teach how to code is the use of Visual Languages (VLs): they at least reduce syntactic problems by encapsulating a traditional programming language – generally domain-specific, i.e. tailored to a given application domain – with a graphical representation of its instructions, often incorporating some nifty tweaks to effectively communicate the underlying semantical rules at a glance; for instance, the available components can be depicted by different puzzle blocks, allowing users to combine them together to build a program: type constraints can be communicated to the users by using different shapes, showing whether an output is compatible with an input, something that still requires assistive tools (e.g. a type checker) even to experienced professional programmers. Representing program syntax trees as compositions of visual blocks is a recent trend in designing VLs, as witnessed by the success of Blocks Programming Environments like Scratch[2], Blockly[3], and App Inventor[4], to name a few.

Since its introduction, many have attempted to define more precisely what Computational Thinking actually means [4]; having not come to an agreement yet, the only consensus reached so far involves the concepts of abstraction and decomposition:

- *Abstraction* refers to modeling problems and systems by capturing their essential properties, considering only the common features while overlooking their differences, since the latter won't be relevant from the analysis' point of view. Modeling in terms of abstraction's layers allows focusing just on one layer at a time together with the formal relations between its adjacent layers; moreover, moving to a higher layer enables not having to worry about the underlying details, thus providing with an easy and effective way of analyzing one element from different perspectives.
- *Decomposition* involves thinking about problems or artifacts (e.g. systems, processes, or algorithms) in terms of their inner components: one can then understand, solve and evaluate them separately, making problems easier to solve and artifacts easier to design.

1 https://hourofcode.com
2 https://scratch.mit.edu
3 https://developers.google.com/blockly/
4 http://appinventor.mit.edu

As stated before, these two concepts are an integral part of Computational Thinking as well as heavily relied on by coding; effectively manipulating highly abstract concepts is often a challenge for inexperienced users, who usually need to be trained and practice these skills for quite a while before being able to properly master them.

Visual blocks are a natural fit for visualizing different components and their relationships (decomposition); they are also particularly suited to easily move between different abstraction layers, allowing a set of blocks to be clustered as one when shifting to a higher level of analysis (abstraction). Thus, they allow users to shift from a problem representation to another by manipulating the same metaphor, unlocking different perspectives of the same process and easing the conceptualization of the many components' roles and structure.

Moreover, they allow to spot similarities and foster the repurposing of known solutions to different problems by identifying similar structures; promoting components' reuse helps identifying common features to problems in different contexts, practicing in quickly solving new problems based on previously solved ones (i.e. generalization [5]).

Another way of aiding users in understanding those abstract concepts often involved by coding and hence foster their CT skills stems from the constructivist theory of Jean Piaget [6], describing how human capabilities evolve during the first years of life: at ages 7 to 11 children are in what he called *concrete operational stage*; they can think logically in terms of objects, but have difficulty replacing them with symbols: they can solve problems in a logical fashion, but are typically not able to think abstractly or hypothetically. The following *formal operational stage* enables them to replace objects with symbols, generalizing and manipulating abstract concepts by using proportional reasoning and deriving cause-effect relationships. The shift from concrete to formal operational should occur by age 12, but a famous study [7] found that most College freshmen in physics courses are still in the concrete operational phase, thus being incapable of grasping abstract concepts not firmly embedded in their concrete experience.

That being the case, we argue that exploiting our innate dexterity for objects' manipulation in the physical world could be an effective way of aiding concrete operational thinkers to grasp abstract concepts involved by coding, thus fostering their Computational Thinking skills.

Physical manipulation sits at the core of a new digital interaction paradigm designed with the aim of providing users with an easy to use interface [8] that can be used even by inexperienced people: this paradigm is known as Tangible User Interfaces [9]. Employing it in a Blocks Programming Environment – thus pairing it up with a well-known technique aiming at lowering programming barriers and allowing end users to program – could foster their CT skills by supporting them with a concrete representation of the abstract concepts involved.

Summarizing, we believe that exploiting the benefits of a tangible interaction in conjunction with visual blocks will leverage on human's natural ability of manipulating objects in the real world, aiding end users in grasping highly abstract concepts while fostering their Computational Thinking skills.

II. Relevance

This work is relevant to the scope of the conference given the recent focus of the entire area over fostering Computational Thinking skills [10] and the new research community arisen around Blocks Programming Environments [11].

III. Presentation

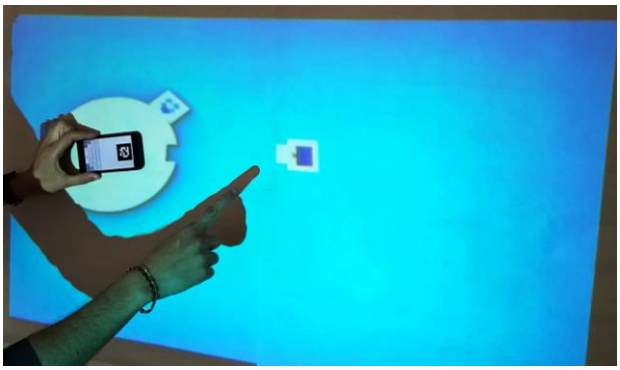

Fig. 1. An example of a workflow being assembled using TAPAS.

In a recent study [12] we introduced TAPAS (TAngible Programmable Augmented Surface), a Blocks Programming Environment that allows users to develop simple workflows by assembling different services together using the movements of their smartphones, as depicted in figure 1. The interaction is carried out using tangible objects and the digital blocks are projected over the surface, making it fun and easy to use.

TAPAS will be showcased as a fully-working installation during the conference, allowing anyone to use their smartphones to interact with the system by using a Web application.

References

[1] J. M. Wing, 'Computational thinking', Commun ACM, vol. 49, no. 3, p. 33, Mar. 2006.

[2] G. Orr, 'Computational thinking through programming and algorithmic art.', SIGGRAPH Talks 2009, pp. 1–1, 2009.

[3] D. Weintrop, 'Blocks, text, and the space between: The role of representations in novice programming environments', presented at the Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on, 2015.

[4] C. Selby and J. Woollard, 'Computational thinking: the developing definition', 2013.

[5] P. Curzon, M. Dorling, T. Ng, C. Selby, and J. Woollard, 'Developing computational thinking in the classroom: a framework', 2014.

[6] J. Piaget and B. Inhelder, The psychology of the child. 1969.

[7] K. A. Williams and A. Cavallo, Reasoning Ability, Meaningful Learning, and Students' Understanding of Physics Concepts. Journal of College Science Teaching, 1995.

[8] A. Bellucci, A. Malizia, P. Díaz, and I. Aedo, 'Don't touch me', New York, New York, USA, 2010, p. 391.

[9] H. Ishii and B. Ullmer, 'Tangible bits', presented at the the SIGCHI Conference, New York, New York, USA, 1997, pp. 234–241.

[10] 'Foreword VL/HCC 2015', 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. vii–viii, 2015.

[11] 'Foreword', presented at the Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE, 2015.

[12] T. Turchi, A. Malizia, and A. Dix, 'Fostering the adoption of Pervasive Displays in public spaces using tangible End-User Programming', presented at the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2015, pp. 37–45.