EAI Endorsed Transactions
on Ambient Systems

Research Article ICST.ORG

A Human-Centred Tangible approach to learning Computational Thinking

Tommaso Turchi[1,*], Alessio Malizia[1]

[1]Human Centred Design Institute, Brunel University London, UB8 3PH, United Kingdom

Abstract

Computational Thinking has recently become a focus of many teaching and research domains; it encapsulates those thinking skills integral to solving complex problems using a computer, thus being widely applicable in our society. It is influencing research across many disciplines and also coming into the limelight of education, mostly thanks to public initiatives such as the Hour of Code. In this paper we present our arguments for promoting Computational Thinking in education through the Human-centred paradigm of Tangible End-User Development, namely by exploiting objects whose interactions with the physical environment are mapped to digital actions performed on the system.

Received on 18 July 2016; accepted on 20 July 2016; published on 23 August 2016

Keywords: Computational Thinking, Education, Tangible User Interface, End-User Development

Copyright © 2016 Tommaso Turchi and Alessio Malizia, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi:10.4108/eai.23-8-2016.151641

1. Introduction

Given how our entire society is increasingly surrounded by technology, programming is becoming an essential skill to master for the general public: from the amount of software involved in managing a flight, to the simple task of turning on the engine of your car, it is unmistakably clear how much we rely on software in every part of our lives. That being the case, people will always strive to play a more active role in their life, resulting in a clear overall heightened interest in coding: take for instance the Hour of Code[1], a successful global initiative organised by Code.org (a non-profit founded in 2013 and supported, among others, by Mark Zuckerberg and Bill Gates) involving millions of students of different ages starting with 4-year old, aiming at introducing coding skills to a wide and heterogeneous audience.

Programming is no longer simply a job skill, but becoming a *literacy*, enabling people to acquire a new way of thinking and seeing the world, fostering abilities like problem solving, abstraction, and pattern recognition to name but a few: in a word, the so-called Computational Thinking (CT) skills. They allow to break a complex problem down into small chunks and express them to a non-human entity such as a computer [1].

In the following, we highlight the current and future role of CT in Education, and present our arguments for fostering its development through pairing End-User Development (EUD) with Tangible User Interfaces (TUIs).

2. Computational Thinking in Education

In her seminal work [2], Wing introduced Computational Thinking (CT) as a set of thinking skills, habits, and approaches integral to solving complex problems using a computer, thus widely applicable in today's information society. It encompasses far more than just programming, representing a range of mental tools reflecting the fundamental principles and concepts of Computer Science, including abstracting and decomposing a problem, recognizing similar ones and being able to generalize their solutions. It shares many of its concepts, practices and perspectives with other subject areas taught in schools, such as science, mathematics, arts and engineering, making a strong case for its teaching in disciplines outside of Computer Science and right from kindergarten [3] as for its promotion to a new form of literacy [1].

*Corresponding author. Email: tommaso.turchi@brunel.ac.uk

[1]https://hourofcode.com

The echo of the discussion around the importance of teaching programming and CT in school [4] resounded already in the adoption of new curricula by many European nations, such as England, where coding is mandatory in elementary schools from year 1[2], Finland, planning to introduce CT in its curriculum from this year, and Estonia, having had coding as part of its curriculum since 2013[3]. Moreover, many other European [5] and extra European countries (e.g., Russia, South Africa, New Zealand, and Australia) either already have or plan to introduce Computer Science as part of their K-12 curriculum [6].

The idea of enabling pupils to participate in CT has been around for thirty years, and can be traced back to the work of Seymour Papert [7]; he first proposed the idea of children engaging in coding and developed the Logo programming environment to enable them to do so. In his paper of 1972 he predicted most of the creative movements around educational technologies we see today, by imagining what would happen if children could teach computers to express their ideas, build things and inventions [8].

Many other tools and programming environments followed and have been developed with the aim of promoting CT skills in K-12 education; the main principle guiding their development was the idea of "low floor, high ceiling" — i.e., they enable any beginner to cross the threshold to create working programs easily (low floor), but are also powerful enough to satisfy the needs of more advanced users (high ceiling). Besides, effective CT-supporting tools for children must not only (1) have low floor and high ceiling, but also (2) provide stepping stones with managed skills and challenges to get them from the "floor" to the "ceiling" (scaffold), (3) enable transfer between different application contexts, (4) support equity, and (5) be systemic and sustainable [9].

Visual Programming Environments (VPE) like Scratch[4], Alice[5], Kodu[6], Blockly[7] and App Inventor[8] closely follow these 5 principles on various degrees: they are relatively easy to use and allow novices to focus on designing and creating while avoiding the issues of the traditional murky and complicated programming syntax. They engage in CT using a three-stage "Use-Modify-Create" pattern [10], scaffolding increasingly deep interactions to foster the development of CT skills; students start using existing artifacts and gain skills and confidence through a series of iterative modifications and refinements, as well as fostering appropriation of what started as someone else's and became one's own.

Tools and programming environments however need to support — and, in turn, be supported — by curricular activities such as game design and robotics, typically serving as a trigger for the iterative exploration of CT while motivating and engaging school children. These activities — along with many similar ones — should support what have been proposed to be the four pedagogical phases of learning to think computationally [3]:

1. *unplugged* (off-screen) activities are used to inspire students and enhance subject knowledge, and can be implemented without the use of computers, making abstract concepts both tangible and visible [11] and improving upon their problem solving skills; they break down limited access barriers and focus on cognitive skills rather than implementation details. Computer Science (CS) Unplugged[9] is one of the main resource for unplugged activities, a collection of free learning activities that introduce students to CT through games and puzzles;

2. *making* activities include playing or making things and taking them apart to solve problems inspired by technology; they encourage students to cohesively combine multiple ideas into an organized process to produce something unexpected [12]. Prototyping and testing physical artifacts — also referred to as Tangible Programming (TP) — directly impacts a digital environment [13], thus tangibly representing CT;

3. *tinkering* [14] supports learning of CT concepts by exploring them in a creative way, allowing students to use materials and tools to represent CT. In agreement with Papert's constructionism model of learning [7], tinkering provides a rich context for developing and representing understanding through the experience and process of building something physical or digital;

4. *remixing* (or "hacking") involves critically looking at existing code, as well as practicing modifying it to suit new purposes; analyzing code, making connections and creating new applications from existing code require sophisticated reasoning and problem solving skills [3], all being essential elements for the development of CT.

[2]https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study
[3]http://www.sitra.fi/en/artikkelit/well-being/future-will-be-built-those-who-know-how-code
[4]https://scratch.mit.edu
[5]http://www.alice.org
[6]http://www.kodugamelab.com
[7]https://developers.google.com/blockly/
[8]http://appinventor.mit.edu

[9]http://csunplugged.org

Lastly, it is worth pointing out that the effectiveness of existing tools seems pretty unsettled with respect to the many facets of CT: for instance, a 2008 study [15] involving 80 urban youth aged 8-18 reported learning of several CT elements through the use of Scratch in an after-school setting; nonetheless, the tool does not provide a mean of encapsulating functionalities into procedures and functions, somehow failing to tap into the abstraction skills. There is undoubtedly a need for new tools that foster CT skills specifically targeted to K-12 education, following the principles just described and guided by the most recent research on how children approach problem solving [16, 17].

3. Fostering Computational Thinking skills with Tangible End-User Development

Fostering the development of Computational Thinking (CT) skills has recently become a focus of the End-User Development (EUD) research community [18], whose aim is to allow end users (i.e., any computer user) to adapt software systems to their particular needs at hand; it strives to enable them to exploit some of the computational capabilities enjoyed by professional programmers, thus to perform their tasks more efficiently and effectively (e.g., task automation). CT, therefore, seems the ideal skill set needed to help the EUD community to reach its aim of lowering programming barriers and fostering its spread.

At the same time, programming itself has proven to be an excellent way of developing CT skills [19] especially in K-12 education (as discussed in the previous section), thus existing EUD techniques might have a similar effect on end users, although much discussion is still ongoing about how much this comes about and results are still inconclusive.

Since its introduction, there have been many attempts of defining more precisely what CT actually means [20]. Even though there is not yet an agreement on it, the only consensus reached so far between the different proposed definitions pertains to the concepts of abstraction and decomposition:

- Abstraction helps modeling problems and systems by capturing only the essential properties common to a set of objects while hiding their differences, the latter being not relevant from the perspective we are currently interested in.
- Decomposition refers to thinking about a problem (or, more generally, an artifact — e.g., a system, a process, or an algorithm) in terms of its components; one can then understand, solve and evaluate them separately, making the overall problem easier to solve.

These two concepts are an integral part of CT, as well as frequently required during any programming task. Dealing with abstract concepts is often a challenge for inexperienced users, who usually need to be trained and practice this skill for quite a while before mastering it.

While developing his theories on learning and developing Logo, Papert widely referred to the work of Jean Piaget: his constructivist theory [21] tries to explain 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. Ultimately, they can solve problems in a logical fashion, but are typically not able to think abstractly or hypothetically. The following stage — i.e., the 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 operational to formal operational should occur by age 12, but a later study [22] found that most College freshmen in physics courses still have not made it, being incapable of grasping abstract concepts not firmly embedded in their concrete experience.

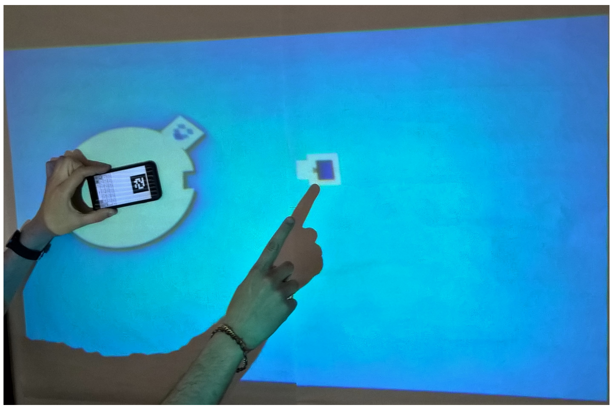
In the light of these considerations, we argue that exploiting our innate dexterity for objects' manipulation in the physical world could be an effective way of aiding concrete operational thinkers to grasp abstract concepts often involved by coding, thus fostering the development of CT skills. Moreover, we already pointed out how tangible objects can support computational learning in making activities [3].

Physical manipulation is an interaction paradigm currently employed in digital systems with the aim of providing users with an easy to use interface that can be used even by inexperienced people; this paradigm is called Tangible User Interfaces (TUIs) [23]. Employing a TUI in an End-User Development system — thus pairing it up with a technique with the aim of 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 they have to deal with.

3.1. TAngible Programmable Augmented Surface

In a recent study [24] we introduced TAngible Programmable Augmented Surface (TAPAS), a system that allows users to adapt a public display's features to their own needs, by using the movements of their smartphone to interact with it (figure 1); users can develop simple workflows by assembling different services together by means of a puzzle: each service is mapped to a puzzle piece and its shape dictates constraints on its required inputs and output. Even though components are not physically represented here but only digitally (i.e., projected over the surface), the system is controlled through a tangible object (i.e., the smartphone, but the system can work even with other types of objects), making it fun and easy to use [25]. The digital representation of the programming constructs involved makes it a rather flexible environment that can be easily repurposed to fit many heterogeneous contexts.

Figure 1. An example of a workflow assembled using TAPAS.



TAPAS was built on the premises of Visual Programming Environments (VPE) like Scratch, striving to properly support the development of CT with low floor and high ceiling, scaffolding, transfer between different contexts, equity, and being systemic and sustainable [9]. Even though TAPAS' actual delivery of many of these properties is yet to be evaluated, we have already carried out a preliminary study [26] with the aim of collecting initial user feedback on TAPAS' interaction modality and main idea.

We interviewed three groups — sized respectively 4, 5 and 6 — of second year Computer Science undergraduate students, each one involved in the developing of an Android application as part of their annual curriculum with the supervision of a teaching staff member. Each group has to meet at least once a week to discuss their progress and work collaboratively on their next development task. We set out to let them freely take TAPAS for a spin and then carried out a semi-structured interview about its core features.

Overall participants enjoyed playing with TAPAS and were able to understand how to operate it to develop simple workflows; their feedback pointed mostly on missing features related to the specific scenario — which could be added by developing additional puzzle blocks or allowing users to develop their own [27].

Nevertheless, further investigations are needed to see whether the proposed interaction modality helps developing users' CT skills in conjunction with its application domains. We are currently running a study with high school students to investigate such implications in a fully-educational domain.

Summarizing, we believe that exploiting the benefits of using a tangible interaction in conjunction with EUD techniques 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 CT skills.

Acknowledgement. We would like to acknowledge the Research Visibility Award funded by Brunel University London for partially supporting this work.

References

[1] Vee, A. (2013) Understanding Computer Programming as a Literacy. *Literacy in Composition Studies* .

[2] Wing, J.M. (2006) Computational thinking. *Communications of the ACM* **49**(3): 33.

[3] Namukasa, I.K., Kotsopoulos, D., Floyd, L., Weber, J., Kafai, Y., Khan, S., Yiu, C. *et al.* (2015) *From computational thinking to computational participation: Towards Achieving Excellence through Coding in elementary schools*. Tech. rep., University of Western Ontario.

[4] (2011) *Report of a Workshop on the Pedagogical Aspects of Computational Thinking* (Washington, D.C.: National Academies Press).

[5] Balanskat, A. and Engelhart, K. (2014) Computing our future: Computer programming and coding-priorities, school curricula and initiatives across europe.

[6] Grover, S. and Pea, R. (2013) Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher* **42**(1): 38–43.

[7] Papert, S. (1980) *Mindstorms: children, computers, and powerful ideas* (Basic Books, Inc.).

[8] Blikstein, P., Sipitakiat, A., Goldstein, J., Wilbert, J., Johnson, M., Vranakis, S., Pedersen, Z. *et al.* (2016) Project Bloks: designing a development platform for tangible programming for children .

[9] Repenning, A., Webb, D. and Ioannidou, A. (2010) Scalable game design and the development of a checklist for getting computational thinking into public schools. In *the 41st ACM technical symposium* (New York, New York, USA: ACM Press): 265–269.

[10] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W. and Erickson, J. (2011) Computational thinking for youth in practice. *Acm Inroads* .

[11] Curzon, P. (2013) Cs4fn and computational thinking unplugged. In *ACM International Conference Proceeding Series* (Queen Mary University of London, London, United Kingdom): 47–50.

[12] Wilkerson-Jerde, M.H. (2014) Construction, categorization, and consensus: student generated computational artifacts as a context for disciplinary reflection. *Educational Technology Research and . . .* .

[13] Strawhacker, A. and Bers, M.U. (2015) "i want my robot to look for food": Comparing kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education* **25**(3): 293–319. doi:10.1007/s10798-014-9287-7, URL `http://dx.doi.org/10.1007/s10798-014-9287-7`.

[14] Bers, M.U., Flannery, L., Kazakoff, E.R. and Sullivan, A. (2014) Computational thinking and tinkering:

Exploration of an early childhood robotics curriculum. *Computers & Education ()* **72**: 145–157.

[15] Maloney, J., Peppier, K., Kafai, Y.B., Resnick, M. and Rusk, N. (2008) Programming by choice: Urban youth learning programming with scratch. In *SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education* (MIT Media Laboratory, Cambridge, United States): 367–371.

[16] Chen, T.Y., Lewandowski, G., McCartney, R., Sanders, K. and Simon, B. (2007) Commonsense computing. *ACM SIGCSE Bulletin* **39**(1): 276.

[17] Pane, J.F., Ratanamahatana, C.A. and Myers, B.A. (2001) Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* **54**(2): 237–264.

[18] Kraemer, E., Ermel, C. and Fleming, S. (2015) Foreword VL/HCC 2015. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*.

[19] Orr, G. (2009) Computational thinking through programming and algorithmic art. *SIGGRAPH Talks 2009* : 1–1.

[20] Selby, C. (2013) *Computational thinking: the developing definition*. Other. URL http://eprints.soton.ac.uk/346937/.

[21] Inhelder, B. and Piaget, J. (1969) The psychology of the child.

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

[23] Ishii, H. and Ullmer, B. (1997) Tangible bits. In *the SIGCHI Conference* (New York, New York, USA: ACM Press): 234–241.

[24] Turchi, T., Malizia, A. and Dix, A. (2015) Fostering the adoption of Pervasive Displays in public spaces using tangible End-User Programming. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (IEEE): 37–45.

[25] Bellucci, A., Malizia, A., Díaz, P. and Aedo, I. (2010) Don't Touch Me: Multi-user annotations on a map in large display environments. In *Proceedings of the Workshop on Advanced Visual Interfaces AVI*.

[26] Malizia, A. and Turchi, T. (2015) Pervasive displays in the wild: Employing end user programming in adaption and re-purposing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[27] Hosio, S., Goncalves, J., Kukka, H., Chamberlain, A. and Malizia, A. (2014) What's in it for me: Exploring the real-world value proposition of pervasive displays. In *PerDis 2014 - Proceedings: 3rd ACM International Symposium on Pervasive Displays 2014*.