Characterizing the Nature of Programs for educational purposes

Violetta Lonati*
Università degli Studi di Milano
Italy
violetta.lonati@unimi.it

Andrej Brodnik*
University of Primorska / Ljubljana
Slovenia
andrej.brodnik@upr.si

Tim Bell
University of Canterbury
New Zealand
tim.bell@canterbury.ac.nz

Andrew Paul Csizmadia
Newman University, Birmingham
United Kingdom
a.p.csizmadia@newman.ac.uk

Liesbeth De Mol
CNRS, Université de Lille
France
liesbeth.de-mol@univ-lille.fr

Henry Hickman
University of Canterbury
New Zealand
henry.hickman@pg.canterbury.ac.nz

Therese Keane
Swinburne University of Technology
Australia
tkeane@swin.edu.au

Claudio Mirolo
University of Udine
Italy
claudio.mirolo@uniud.it

Mattia Monga
Università degli Studi di Milano
Italy
mattia.monga@unimi.it

Matti Tedre
University of Eastern Finland
Finland
matti.tedre@uef.fi

ACM Reference Format:
Violetta Lonati, Andrej Brodnik, Tim Bell, Andrew Paul Csizmadia, Liesbeth De Mol, Henry Hickman, Therese Keane, Claudio Mirolo, Mattia Monga, and Matti Tedre. 2022. Characterizing the Nature of Programs for educational purposes. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 2 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3502717.3532173

ABSTRACT

Programming plays a paramount role in many educational policies and initiatives. However, the current focus on coding skills poses a risk of giving pupils an over simplistic and impoverished idea of what programming means and involves. Their experiences would be much more significant if learning were aimed at understanding the richness of the nature of programs.

Programs are strange creatures that escape simple definitions: they are real – they affect our real lives; they are abstract – they process abstract entities; and they are concrete – they take up space in digital devices memory, and can be copied, transferred, corrupted. Thus, understanding the multifaceted nature of programs is crucial knowledge for all citizens of the digital era, and a fundamental component of such an understanding is getting a sense of how programs are created and work (i.e., the programming process).

To the best of our knowledge, there is no Nature of Programs framework (a set of statements describing the nature of programs) that teachers and policy makers can use to shape their practice and targets. Our goal is to develop such a framework, by collecting and organizing contributions from CER, CS experts, and educators.

*co-leader

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9200-6/22/07…$15.00
https://doi.org/10.1145/3502717.3532173

BACKGROUND AND RELATED WORK

The focus on programs. Computer programs are part of our daily life, we use them, we provide them with data, they support our decisions, they help us remember, they control machines, etc. Programs are made by people, but in most cases we are not their authors, so we have to decide if we can trust them. Programs enable computers and computer-controlled machines to behave in a large variety of ways. They bring the intrinsic power of computers to life. Programs have a variety of properties that all citizens must be aware of; due to the intangible nature of programs (NoP), most of these properties are unusual and peculiar, but fundamental for understanding the digital world. In other terms, understanding the NoP is a key component of the computing literacy: it is crucial to enable a creative and conscious use of computing devices, and should be one of the main outcomes of computing education — alongside with, *e.g.*, the development of problem solving and computational thinking skills. Moreover it should be part of any effort aimed at bringing digital competences to the general public. An attempt in this direction has been carried out by the WG proponents in the occasion of defining the *Programming* competence in DigComp 2.2 framework [16]; the outcome of this work is a preliminary list of knowledge statements and examples about the NoP as reported in [2].

However, the full understanding of NoP might not be a natural learning outcome of CS activities. For instance, using visual programming environments does not imply that students are able to recognize that the programs they write have the same nature as the "apps" they use on their mobile phones. Similarly, unplugged activities aimed at developing computational thinking skills might be perceived as disconnected from the use of digital devices and programs in everyday life [8, 12]. To overcome these limited perspectives, teachers need to be aware of what NoP is, and use this knowledge to inform their teaching practice.

Programs, programming, and computational thinking. The centrality of programming in CS is reflected in most computing education initiatives[1], which indeed often include some type of programming activity, mainly under the term 'coding'. One can even argue that, for many teachers, CS is just a synonym for coding [15].

Another fundamental component of computing education revolves around the idea of *Computational Thinking* (CT) [18]. Even if there is no its ultimate definition, this idea concerns the ability to address "problems in a way that enables us to use a computer and other tools to help solve them" [6]. CT encompasses a variety of creative cognitive processes and activities, like modeling real-life situations, representing information in digital form, organizing data, analyzing and generalizing computational solutions, assessing their social impact, etc. In other terms, CT goes far beyond coding and tries to represent and value the greater richness of computing. Since the above mentioned activities play a fundamental role also in the process of designing programs, our WG approach shares a similar scope as CT. A fundamental difference is that CT-based frameworks mostly focus on CT practical and cognitive *skills*, while here we chose to reflect about the underlying *knowledge* about the NoP, that CT activities should both promote and stem from.

Historical and philosophical perspective. As one can expect, the NoP has had a relevant role in the broader discussion about the nature of CS. Some very significant contributions on the issue appeared in the '80–90s [3, 11, 17], including the triggering question "Is CS a science?". CS and its impact on society has greatly evolved since, see e.g. [4, 5, 10] for more recent contributions on the nature of CS, and [13] for an articulate presentations of a historical perspective. Such a discussion, however, has been conducted by and for CS, philosophy, and history matter experts, and has rarely reached a wider audience. Notable exceptions are [1, 4, 14].

Focusing on programs, the ongoing project "PROGRAMme" starts with the premises that the seeminingly simple question "What is a computer program?" has no simple answer today[2]. It thus aims at developing "a coherent analysis and pluralistic understanding of 'computer program' and its implications to theory and practice", by taking a historical and philosophical approach. The project plans to consider the various characterizations of programs that derive from different viewpoints and pertain to different historical phases of the development of the discipline (either in academia or industry).

Considering "PROGRAMme" and the mentioned references on the nature of CS, we take a different approach. Instead of analyzing and contrasting the different points of view about the NoP, we want to identify the fundamental tenets that bring together different views. Specifically we target the educational arena with a goal to develop a NoP framework that teachers and educational policy makers can use in framing CS teaching practice and curricula.

[1]Cf. CS4ALL (https://www.csforall.org/) and Informatics for All Coalition (https://www.informaticsforall.org).
[2]See the project website https://programme.hypotheses.org

The Nature of Programs and the Nature of Science. The expression "Nature of Programs" draws inspiration from "Nature of Science" (NoS), an expression from the '70s that refers to the fundamental characteristics of science knowledge and scientific inquiry, as derived from how it is produced: a necessary knowledge to make informed decisions with respect to the ever-increasing scientifically-based personal and societal issues [9]. NoS is a significant component of scientific literacy and it is argued that NoS cannot be learned simply by studying science concepts or attending science labs, but it must be addressed explicitly with active reflective practice and discussions among students in their learning contexts [7]. This also implies that teachers should have a "shared accurate view of NoS" and agree that NoS needs to be taught and assessed explicitly [9].

CS education would benefit as well from a similar approach, in order to boost its contribution in creating a CS-literate society that is able to make informed decisions on CS-related issues. This working group proposes a first step in this direction, focusing on the Nature of Programs (including the way that they are created and built), due to the role that programs play in CS and all society.

REFERENCES

[1] Tim Bell, Paul Tymann, and Amiram Yehudai. 2018. The Big Ideas in Computer Science for K-12 Curricula. *Bull. EATCS* 124 (2018).
[2] Andrej Brodnik, Andrew Csizmadia, Gerald Futschek, Lidija Kralj, Violetta Lonati, Peter Micheuz, and Mattia Monga. 2021. Programming for All: Understanding the Nature of Programs. *CoRR* abs/2111.04887 (2021). arXiv:2111.04887
[3] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. 1989. Computing as a discipline. *Computer* 22, 2 (1989), 63–70.
[4] Peter J. Denning and Craig H. Martell. 2015. *Great Principles of Computing*. The MIT Press.
[5] Amnon H. Eden. 2007. Three Paradigms of Computer Science. *Minds Mach.* 17, 2 (jul 2007), 135–167. https://doi.org/10.1007/s11023-007-9060-8
[6] International Society for Technology in Education(ISTE) and Computer Science Teachers Association (CSTA). 2011. Operational Definition of Computational Thinking for K12 Education. https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf
[7] Norman G. Lederman. 2013. Nature of science: Past, present, and future. In *Handbook of research on science education*. Routledge, 845–894.
[8] Violetta Lonati, Dario Malchiodi, Mattia Monga, and Anna Morpurgo. 2015. Is coding the way to go?. In *8th ISSEP* (Ljubljana, Slovenia) *(LNCS, Vol. 9378)*, Andrej Brodnik and Jan Vahrenhold (Eds.). Springer International Publishing, Switzerland, 165–174. https://doi.org/10.1007/978-3-319-25396-1_15
[9] National Science Teachers Association. 2020. Nature of Science. https://www.nsta.org/nstas-official-positions/nature-science
[10] William J. Rapaport. 2005. Philosophy of Computer Science: An Introductory Course. *Teaching Philosophy* 4 (2005), 319–341. Issue 28.
[11] Mary Shaw. 1985. The Nature of Computer Science. In *The Carnegie-Mellon Curriculum for Undergraduate Computer Science*. Springer, 7–12.
[12] Rivka Taub, Michal Armoni, and Mordechai Ben-Ari. 2012. CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS. *ACM Trans. Comput. Educ.* 12, 2, Article 8 (April 2012), 29 pages.
[13] Matti Tedre. 2014. *The Science of Computing: Shaping a Discipline*. CRC Press.
[14] Matti Tedre. 2018. The Nature of Computing as a Discipline. In *Computer Science Education. Perspectives on Teaching and Learning in School*, Sue Sentance, Erik Barendsen, and Carsten Schulte (Eds.). Bloomsbury Publishing, Chapter 1.
[15] Mike Tissenbaum and Anne Ottenbreit-Leftwich. 2020. A Vision of K-12 Computer Science Education for 2030. *Commun. ACM* 63, 5 (April 2020), 42–44.
[16] Riina Vuorikari, Stefano Kluzer, and Yves Punie. 2022. *DigComp 2.2: The Digital Competence Framework for Citizens - With new examples of knowledge, skills and attitudes.* Joint Research Centre (European Commission), European Union.
[17] Peter Wegner. 1976. Research Paradigms in Computer Science. In *Proceedings of the 2nd International Conference on Software Engineering* (San Francisco, California, USA) *(ICSE '76)*. IEEE Computer Society Press, Washington, DC, USA, 322–330.
[18] Jeannette M Wing. 2006. Computational thinking. *CACM* 49, 3 (2006), 33–35.