Mind the Metaphor: Charting the Rhetoric about Introductory Programming in U.S. K-12 Schools

Quinn Burke[1]

Abstract

Computer programming is increasingly being touted by educators, economists, and policy makers as one of the definitive skills that all children need to learn on the K-12 level. Currently, the United Kingdom, France, South Korea, and Israel have mandated computer science to be taught on the K-12 levels, while Singapore and Italy are in the process of likewise developing national curricula. Through the White House, the United States recently committed an unprecedented amount of money—4 billion dollars—for the development of computer science coursework on the K-12 level. Despite these promising developments, there remains a lingering question as to where where coding can find its footing in an already full school day. The purpose of this exploratory paper is to review and discuss the varied ways computer programming is introduced to schools and families as a new form of learning. If the future of education will see programming in schools, where and how will it be introduced and implemented as a course on the K-12 level? The paper examines the rhetoric around coding within international academic journals and popular media articles over the past two decades. Findings point to two distinct ways in which introductory coding initiatives have been portrayed (and been perceived): as a new literacy and as a "grounded" math. Ultimately, the paper does not propose a single defining metaphor. Rather it argues that the metaphors one selects matter considerably in determining programming's future in entering (or not entering) schools, and that educators need to make a conscientious effort to consider multiple metaphors without choosing just one.

Introduction: Computer Programming as What Children Need to Learn

Computer programming—once considered to be solely the trade of "techies"—is increasingly being recognized by both economists and educators as one of the definitive skills that children need to learn and schools need to teach (Gardiner, 2014). Yet, among developed countries the United States falls far behind here in terms of its curricular offerings. As the Code.org program (2016) points out at their website, currently 22 states in the U.S. still do not allow computer science to count towards high school graduation, which is in stark contrast to countries like the U.K, France, South Korea, and Israel, all of which not only allow CS to count, but some of which have mandated their own respective national computer science curricula starting in the primary grades (Johnson, 2014). So how does computer science—specifically programming—find its entry point into the education system? This proposal argues that of all the challenges facing CS education on the K-12 level, the most immediate issue is one of presentation. What is computer science and how does it fit into schools' curricula? Which teachers may best be suited to teach CS? Is it akin to the general technology courses already offered at schools on selective days? These are important yet largely unanswered questions. This primary question of "fit" sits at the heart of the paper. It is this proposal's thesis that despite the best intentions of politicians and media pundits, if the rhetoric around CS in all schools is to become a reality, there first needs to be a greater focus on monitoring such rhetoric and better understanding exactly how computer programming is being presented to the wider public.

Analytical Framework: the Genrative Nature of Metaphor

Here enters the role of metaphor. Yes, metaphors, that which one may recall from middle school language arts coursework plays an instrumental role in determining how computer programming is presented to students and teachers. According to the seminal Metaphors We Live By (1980), metaphors represent nothing less than the tools by which we conceive life and function daily within it. The book, by linguist George Lakoff and philosopher Mark Johnson, represents the first substantial analysis of metaphors beyond their role as literary devices and as fundamental ways of understanding the world. Metaphors orient us (e.g., "future lies ahead") and they structure are perceptions (e.g., "time is money"). Indeed, among computer programmers, speaking in metaphors is a necessity on the job. Phrases such as "looping around a section" or "debugging a script" are common expressions among

[1] College of Charleston, United States of America

coders replete with metaphors that serve as crucial conceptual models for processes (Fishwick, 2002). Lakoff and Johnson's book however sees metaphor-making as an inherently generative process. In contrast to comparison theory (Suls & Willis, 1991), one of the key tenants of Metaphors We Live By is that metaphors do not uncover relationships that are objectively true, but rather create similarities (p. 153). "Getting the right metaphor is important, but so is knowing the limitations of our metaphors," explains renowned computer scientist Hal Abelson, "An imperfect metaphor can mislead as much as an apt metaphor can illuminate" (p.4). This returns to Lakoff and Johnson's (1980) framework, suggesting that since metaphors are less a matter of revelation than creation, one must be especially aware of those we create.

Data Sources

This paper represents an analysis of 67 peer-reviewed books and journal articles as well as news articles and editorials related to students' learning (or needing to learn) computer programming on the K-12 level. In terms of the former—books and articles—my focus was on how researchers and educators conceptualized programming with their educational outreach; in terms of the latter—news articles and editorials (circulation > 500,000)—I focused on how computer programming was being presented as a subject of study and whether it was being aligned (explicitly or implicitly) to any already existent academic subjects. Each article, whether academic or popular, is analyzed in terms of (a) the figurative language it employs, (b) the association(s) it make(s) with pre-existing school subjects as they relate to programming, and (c) the intended audience of the publication, whether purported or stated outright.

Minding the Metaphors

While the review is ongoing, of the 67 sources that have been analyzed thus far, two clear "camps" emerge in terms of the way introductory programming has been presented:

Code is "Grounded Mathematics"

First, is the *grounded math* metaphor which is the older of the two camps, having taken hold in the late 1970s and early 1980s with the introduction of Logo to K-12 schools. Developed by Seymour Papert (1972, 1980) of the MIT Media Lab, Logo represented the first widespread introduction of programming into schools and no small part of Papert's success came from his use of the "grounded" or "practical" math metaphor to explain code as a way to make math more tangible and real to students. "In this book, the *Mathland metaphor* will be used to question deeply engrained assumptions about human abilities," he writes, continuing on to explain that Logo acts as a land in which children can immerse themselves in practical math such as geometry and algebraic expression to generate content in such a world (pp. 6-7). Papert was a gifted writer and his metaphor use in the seminal *Mindstorms* (1980) extends beyond grounded math and into even more useful metaphors for explaining the function of math itself. But Papert's connecting code to mathematics is entirely sensible given the longstanding relationship between computer programing and mathematical proofs in which logical precision and sequence were of paramount importance (Chermside, 2012). Papert's metaphor of grounded math influenced the work of a number other leading computer scientists and educators at that time, who also employed the metaphor to explain code as mathematical proofs (Abelson & diSessa, 1980; Soloway et al., 1982; Wilensky, 1995).

However, with subsequent pushback on Logo as failing to lead to improved mathematical comprehension among students (Pea & Kurland, 1983), the math metaphor was largely abandoned as a means to promote introductory programming efforts. Select editorials (Burt, 2014; Weissman, 2014) continue to employ the grounded math metaphor as way to not only make programming more sensible to K-12 schools but likewise make mathematics education more concrete and practical to students—a "Trojan Horse" according to Gadanidis (2015) for making math more palpable. However, the overall demise of Logo in K-12 schools still makes the grounded math metaphor tenuous for any of those researchers and educators who remember Logo's relatively brief foray in classrooms in the late 1970s and early 1980s.

Code is Language/ Literacy

One of the more basic challenges with the math metaphor is that it does not necessarily "sell" computer programming to students because often the same students hesitant to try coding are the ones who likewise resist math. And unfortunately by middle school this divide often falls along racial and gender lines (Margolis & Fischer, 2002; Margolis, 2008). The more recent appearance within the

literature and media has been the metaphor of literacy, perhaps most famously promoted by theorist Douglas Rushkoff in his popular book Program or be Programmed (2010). And as a metaphor, literacy is fast gaining traction through various incarnations such as "procedural literacy" around programming games (Bogost, 2007), computational literacy around understanding computers (diSessa, 2000), and both computational thinking (Wing, 2006) and computational participation (Kafai & Burke, 2014). Of course while literacy is a relatively new way of representing computer programming, its roots are much deeper as, since its inception, CS has conceived of code as "language" and "writing code" as synonymous with programming a computer. Early manuals such as Kernighan and Plauger's (1974) The Elements of Programming Style (as the corollary to Strunk and White's literary book) and Knuth's (1968) multi-volume The Art of Computer Programming helped bolster this perception of code and written language as synonymous forms of personal expression. The metaphor is not only a natural extension of "code as language" but also a powerful tool, argues Vee (2013). The term "literacy", she points out, is a not-an-all-too subtle way of letting schools know that it is their responsibility to teach programming as a fundamental form of communication (p. 44). Under the metaphor of literacy, programming is no longer an erudite "skill set" but a fundamental way to communicate in the 21st century and one that schools need to address.

Of course, just as there are limits to the math metaphor so there are limits to the literacy metaphor. First, code is not the same as human language, which is a naturally occurring rather than artificial construct. As any programmer can attest, entering a semi-colon in the wrong place within a line of a code rather than within a sentence can have considerably more drastic results. A misplaced semi-colon within a sentence may muddle, even alter, meaning. But a misplaced semi-colon in a line of code may very well prevent a program from compiling altogether. This distinction between natural and artificial languages came to a head with the recent movement to get computer programming admitted into core curricula coursework under states' foreign language requirements (Adam & Mowers, 2013; Tyre, 2013). Rather than taking French or Latin, children, its proponents argued, should be learning computer coding languages—be it Java, C, Python—as the "real" languages of the future. Yet as immediate sensible as this suggestion may appear, Chris Stephenson, then head of the Computer Science Teacher Association, promptly dismissed this connection, pointing out that entering a programing class under the pretense that code functions like natural language ultimately clouds the overall picture of what code is and how it functions (2014).

Discussion/ Significance Going Forward

This research will be ongoing in terms of further categorizing and aligning the various metaphors used to introduce computer programming to schools. Returning to the initial query: Can we realistically expect coding to enter K-12 schools widely as part of core curricula? The challenges of recruiting and training a new body of teachers and carving out the requisite curricular time in an already full school day are daunting hurdles. But returning to this paper's central argument, until enthusiasts get a better handle on the various ways programming is being presented to the pubic, there is little chance for articulating a clear plan for wider implementation and utilizing pre-existing resources. It is not a matter of picking one metaphor but rather charting the various metaphors that are regularly (and often tacitly) employed.

References

[1]Abelson, H. & diSessa, A. (1980). Turtle geometry. Cambridge, MA: MIT Press

[2]Adams, A. & Mowers, H. (2013, October 30). Should coding be the new foreign language requirement? Edutopia. Available at http://www.edutopia.org/blog/coding-new-foreign-language-requirement-helen-mowers.

[3]Burke. (2012) The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. Journal of Media Literacy Education, 4(2), 121-135.

[4]Bogost, I. (2007). Persuasive games: The expressive power of videogames. Cambridge, MA: MIT Press.

[5]Burt, R. (2014, February 6). "Forget coding—let's change up how we teach math". Edublogger. Available at http://theedublogger.com/2014/02/06/forget-coding-lets-change-up-how-we-teach-math/.

[6]Chermside, M. (2012). Metaphorical programming. From Adventures in Programming. Available at http://mcherm.com/permalinks/1/metaphorical-programming.

[7]diSessa, A. (2000). Changing minds. Cambridge, MA: MIT Press.

[8]Gadanidis, G. (2015). Coding as a Trojan Horse for mathematics education reform. Journal of Computers in Mathematics and Science Teaching, 34(2), 155-173.

[9]Gardiner, B. (2014, March 23). "Adding coding to the curriculum". New York Times. Available at http://www.nytimes.com/2014/03/24/world/europe/adding-coding-to-the-curriculum.html?_r=0. .
[10]Hoyles, C. & Noss, R. (1987). Synthesizing mathematical conceptions and their formalization through the construction of a Logo-based mathematics curriculum. International Journal of Mathematical Education, 18(4), 581-585.
[11]Johnson, P. (2014, July 16). "France to offer programming in elementary school". IT-World. Available at http://www.itworld.com/application-management/427170/ france-offer-programming-elementary-school.
[12]Kafai, Y.B. & Burke. (2014). Connected code: Why children need to learn programming. Cambridge, MA: MIT Press.
[13]Kernighan, B.W. & Plauger, P.J. (1974). The elements of programming style, New York: McGraw-Hill.
[14]Knuth, D. (1968). The art of computer programming. Reading, MA: Addison-Wesley.
[15]Margolis, J. & Fisher, A. (2002). Unlocking the Clubhouse: Women in computing. Cambridge, MA: MIT Press.
[16]Margolis, J. (2008). Stuck in the shallow end. Education, race, and computing. Cambridge, MA: MIT Press.
[17]Papert, S. (1972). Teaching children to be mathematicians versus teaching about math. International Journal for Mathematical Education, Science, and Technology, 3, 249-262.
[18]Papert, S. (1980). Mindstorms: Children, programming, and powerful ideas. New York: Basic Books.
[19]Pea, R.D. & Kurland, D.M. (1983). Logo programming and the development of planning skills. Center for Children & Technology. NY: Bank Street College.
[20]Richtel, M. (2014, May 10). Reading, writing, arithmetic, and lately, coding. New York Times. Available at http://www.nytimes.com/2014/05/11/us/reading-writing-arithmetic-and-lately-coding.html?smid=tw-share&smv2&_r=4.
[21]Rideout, Victoria, Ulla G. Foehr, and Donald F. Roberts. (2010). Generation M2: Media in the lives of 8-18 year-olds. Kaiser Family Foundation. Available at http://kff.org/other/poll-finding/report-generation-m2-media-in-the-lives/.
[22]Rushkoff, D. (2010). Program or be programmed. NY: O/R Books.
[23]Soloway, E., Lockhead, J., & Clement, J. (1982). Does computer programming enhance problem solving ability? In Computer Literacy Issues and Directions. R Seidel, R. Anderson, & B. Hunter (Eds.). New York: Academic Press.
[24]Stephenson, C. (2014, February). "Why counting CS as a foreign language credit is a bad idea". CSTA Advocate. Available at http://blog.acm.org/archives/csta/ 2014/02/why_counting_cs.html
[25]Tyre, P. (2013, May 23). "Is coding the new second language?" Smithsonian.com. Available at http://www.smithsonianmag.com/innovation/is-coding-the-new-second-language-81708064/?all .
[26]Vee, A. (2013). Understanding computer programming as a literacy. Literacy in Composition Studies, 1(2), 42-65.
[27]Weissman, (2013, May 15). "Why high schools should teach computer programming like algebra". The Atlantic. Available at http://www.theatlantic.com/ national/archive/2013/05/why-high-schools-should-treat-computer-programming-like-algebra/275893/.
[28]Wilensky, U.(1995). Paradox, programming, and leading probability: A case study in a connected mathematics framework. The Journal of Mathematical Behavior, 14(2), 253-280).
[29]Wing, J.M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.