ACADEMIA | Letters

Teaching Coding as a Literacy: Issues, Challenges, and Limitations

Alvin Vista

The concept of regarding computer coding skill as a literacy skill has been around for some time. Computer scientists as early as the 1950s have emphasized that computer languages should be developed as to be as readable as a human language as much as possible and have linked programming with writing composition skills—that is, a changing perspective that programming involves more than computation and engineering. More recently, the slogan "coding as literacy" is gaining popularity [1]. This idea is based on the fact that computer code is fundamentally a language, a language for machines.

Computational linguists continue to argue whether or not all languages (natural or artificial) have common elements and follow fundamental rules [2]. Regardless of which camp is correct at the fundamental level, there are still important differences between natural and artificial languages at more practical levels. Artificial languages are heavily, if not totally, based on rules of logic and so share the same structure as mathematics. But whereas mathematics is primarily for model-building, language (both natural and artificial) is for communication or transferring information. Artificial languages are more structured at the moment, unlike natural languages where ambiguity and inconsistency are much more common features. Although with fast advances in natural language processing, higher level (or more abstracted) computer languages can evolve closer to natural languages even if lower-level languages (i.e., closer to the machine instruction set, with minimal abstraction) remain strictly structured.

From a learning perspective, the idea that computer code is just a language suggests that we should view learning how to read and write in code in the same way as any other language and therefore teach coding as we would teach literacy. This is a powerful idea.

Advocates of the view that teaching coding as fundamentally teaching literacy push for coding to be taught beyond the traditional confines of technology-oriented fields. Extending this view is the implication put forth by author Annette Vee that being literate in computer languages should be a skill that needs to be developed for all [3], just as we assert universal literacy in natural languages.

This is certainly a sensible advocacy. We can see that there is a growing trend of pushing for coding to become part of mainstream teaching and learning [4], especially in the early grades. It seems that coding is becoming sexy again.

The inseparability of literacy and language

Viewing coding as a literacy implies that it should be taught to as many learners as early as possible, just as we would develop literacy in the traditional sense. The direct and primary outcome that results in rolling out universal coding in schools is through exposing children to learn actual coding at a very early age. However, from the perspective of teaching coding-literacy, we must not forget the challenges and limitations we face in teaching literacy itself.

Teaching literacy is inherently linked with a language and cannot be separated from the issues and considerations that have to be made when teaching any particular language. Thus, when we talk about teaching literacy, we cannot ignore the question of "literacy in which language?" and the issue of utility.

Even among natural languages, not everyone sees the need to learn even particularly practical languages. English is commonly accepted as the lingua franca of the business world, and yet not everyone wants or needs to learn English. Even within the business community, English proficiency is far from a prerequisite to success.

Learning English is undeniably advantageous, similarly as being computer-literate. But this analogy is not perfect when applied to coding languages because the diversity of computer language means that there is no analogous lingua franca even in very narrow subfields. For example, there are more than a dozen mainstream coding languages just for web development and a user of JavaScript might not be fluent or even understand Ruby—much less languages from completely different fields (e.g., Wolfram for symbolic computation).

This inseparability of literacy and language has important implications for a wide range of policies that go beyond instructional reform. In particular, it has implications for policies related to curriculum reform if the aim is to implement "coding as literacy" in national education systems. It also has implications for system-wide assessment policies because there needs to be a uniform set of assessment frameworks if we are to develop common competency standards that align with curricula. Currently, there are standards and assessment frameworks on information literacy, but these are not the same and are much narrower compared to coding literacy. If coding literacy is to be implemented in the same way as traditional literacy, these policy implications will have to be carefully considered, and not just for curriculum and assessment but also for policies on teacher training, education financing, and even policies that relate to the workforce.

Are the secondary outcomes enough?

Advocating for universal coding has a general, although perhaps secondary, effect of teaching children how to think systematically and learn symbolic logic. Extending this advocacy as the children progress through school along this pathway, this secondary outcome also leads to children eventually developing computational thinking [5].

Some would argue that the indirect effects outweigh the costs such that, even when focusing on a particular language, doing so develops systematic and computational thinking. This is a valid argument and there is evidence [6] that developing literacy fundamentally changes the brain itself, such that even when the learned language is never utilized, the overall cognitive effect of that learning process is still positive.

If this is the case, there needs to be a balance between a useful language now and a language that maximizes the general learnings—a balance between utility and generalizability. This is because what might be useful now might be obsolete in the future. Languages that depend more on how current hardware work are more sensitive to hardware changes. Imperative languages, like most of the popular object-oriented languages, are designed to provide instructions on how to change the state of the system at the processor-level. These languages are therefore dependent on how the processors are designed. If mainstream processors evolve into something that's non-binary (e.g., quantum) or even non-digital (e.g., organic-based), then they will also require completely new imperative languages.

If the focus is on indirect effects such a systematic and computational thinking, then it would be more effective to focus on languages that emphasize the fundamental concepts of logic and computation. Because even if such languages are still designed for existing machines and remain susceptible to obsolescence, learning them has useful secondary effects. It can be argued that functional languages, which are closer to the structure of mathematics, might be more effective in incidentally developing systematic thinking. For example, Haskell would be among the best languages to teach if the focus is on developing systematic and computational thinking, even if it might not be a prioritized choice from a purely utilitarian perspective.

Focusing on secondary effects shifts the role of teaching coding to a means of developing these generalizable skills rather than as an end towards developing literacy in artificial languages. Because systematic and computational thinking can be framed as part of the so-called 21st century skills, teaching coding becomes part of the toolset for developing 21st century skills more broadly.

This approach also has policy implications, specifically related to implementation in the curricula—for example, grade-level scope, extent of domain coverage, and what subject-level integration strategies should be adopted. A recent report provides insight on how the Nordic countries (Denmark, Finland, Norway, and Sweden) are tackling these issues as they start to develop a cohesive policy strategy [7]. Developing policy insights such as these will be essential towards building a coherent set of policies on implementing a new learning domain such as computational thinking.

A cross-pollination of ideas as we move forward

Re-conceptualizing coding as a literacy demystifies it and helps remove preconceived notions that only a select few can (or need) to learn it. This can broaden the pedagogical scope of coding literacy and open opportunities for learning to those who might not conventionally be engaged in the traditional conceptualization of coding.

Going forward, the most important pedagogical consequence of re-conceptualizing coding is that we enlarge the resources that can be brought to bear in its teaching and learning. Literacy pedagogies have been around for much longer than computer science and there are well-established strategies based on the developmental processes of literacy in natural languages that can be applied to artificial ones.

Studies of effective reading and writing strategies can provide useful ideas that can be adapted for teaching coding. For example, focusing on emergent writing skills is important for any early language learners [8]. We can also use instructional strategies from literacy education that focus on vocabulary development, morphological knowledge, and semantic comprehension. Strategies designed for compositional writing are also important because working in a collaborative environment requires writing code that is readable by the team. In an increasingly global and linguistically diverse workforce, clear writing both of the code as well as the supporting documentation is even more relevant.

We can also adopt solutions from language teaching on the issue of balancing primary and secondary outcomes. We can approach the challenge of balancing between utility and generalizability in the same way as how first (L1) and second languages (L2) are taught and prioritized in schools.

There are numerous factors that affect the transferability of skills between L1 and L2. Research findings show that there are differential impacts of these factors on transferability, depending on the language pair. This means that even if there is a common underlying proficiency between languages such that learning one makes learning another easier, the order of which language to learn first is not the same for all pairs. For example, learning English as L2 might be easier regardless of L1 if the learner is in an English-speaking country, but not the other way around. This has implications for the choice of L1 and L2 analogs to balance the utility and generalizability in teaching coding. Finally, research that tackles issues of convergence and divergence in language evolution that arise from their usage (for example, native vs business English [9]) can inform how we teach coding as a literacy at systems level.

All the cross-pollination of research findings and broader discussion of challenges will be especially beneficial as we aim to develop coding literacy among the very young. Coding as literacy is an approach that offers opportunities to teach artificial languages side by side as they learn natural languages. It will be exciting to see a new generation of children who are literate in languages in which the distinction between natural and artificial is no longer relevant.

References

[1] Vee, A. (2013). Understanding Computer Programming as a Literacy. *Literacy in Composition Studies, 1*(2), 42–64. `https://doi.org/10.21623/1.1.2.4`

[2] Evans, N., & Levinson, S. C. (2009). The myth of language universals: Language diversity and its importance for cognitive science. *Behavioral and Brain Sciences, 32*(5), 429–448. `https://doi.org/10.1017/s0140525x0999094x`

[3] Vee, A. (2017). Coding Literacy. *The MIT Press*. `https://doi.org/10.7551/mitpress/10655.001.0001`

[4] Lynch, M. (2018, January 29). Coding as a Literacy for the 21st Century. *Education Futures: Emerging Trends in K-12*. `http://blogs.edweek.org/edweek/education_futures/2018/01/coding_as_a_literacy_for_the_21st_century.html`

[5] Wing, J.M. (2006) Computational Thinking. *Communications of the ACM*, 49, 33-35. `https://doi.org/10.1145/1118178.1118215`

[6] Carreiras, M., Seghier, M., Baquero, S. et al. An anatomical signature for literacy. *Nature* 461, 983–986 (2009). `https://doi.org/10.1038/nature08461`

[7] Bocconi, S., Chioccariello, A. and Earp, J. (2018). The Nordic approach to introducing Computational Thinking and programming in compulsory education. Report prepared for the Nordic@BETT2018 Steering Group. `https://doi.org/10.17471/54007`

[8] Pinto, Giuliana & Bigozzi, Lucia & Accorti Gamannossi, Beatrice & Vezzani, Claudio. (2012). Emergent Literacy and Early Writing Skills. *The Journal of genetic psychology* 173. 330-54. `https://doi.org/10.1080/00221325.2011.609848`

[9] Seidlhofer, B. (2004). Research perspectives on teaching English as a lingua franca. *Annual Review of Applied Linguistics* 24, 209-239. `https://doi.org/10.1017/s0267190504000145`