

Education

From Computational Thinking to Computational Participation in K–12 Education

Seeking to reframe computational thinking as computational participation.

COMPUTATIONAL THINKING HAS become a battle cry for coding in K–12 education. It is echoed in statewide efforts to develop standards, in changes to teacher certification and graduation requirements, and in new curriculum designs.¹ The annual Hour of Code has introduced millions of kids to coding inspired by Apple cofounder Steve Jobs who said, “everyone should learn how to program a computer because it teaches you how to think.” Computational thinking has garnered much attention but people seldom recognize that the goal is to bring programming *back* into the classroom.

In the 1980s many schools featured Basic, Logo, or Pascal programming computer labs. Students typically received weekly introductory programming instruction.⁶ These exercises were often of limited complexity, disconnected from classroom work, and lacking in relevance. They did not deliver on promises. By the mid-1990s most schools had turned away from programming. Pre-assembled multimedia packages burned onto glossy CD-ROMs took over. Toiling over syntax typos and debugging problems were no longer classroom activities.

Computer science is making a comeback in schools. We should not repeat earlier mistakes, but leverage what we have learned.⁵ Why are students interested in programming?



Students in a Makey Makey workshop conducted by volunteers from Robogals Wellesley.

Under what circumstances do they do it, and how?² Computational thinking and programming are social, creative practices. They offer a context for making applications of significance for others, communities in which design sharing and collaboration with others are paramount. Computational *thinking* should be reframed as computational *participation*.

Computational Participation

This idea expands on Jeannette Wing’s original definition of computational

thinking.⁷ *Computational participation* involves solving problems, designing systems, and understanding human behavior in the context of computing. It allows for participation in digital activities. Many kids use code outside of school to create and share. Youth-generated websites have appeared to make and share programmable media online. These sites include video games, interactive art projects, and digital stories. They are inherently do-it-yourself (DIY), encouraging youth programming as an effective way to create and share online,

and connect with each other, unlike learned disciplines such as algebra or chemistry. Through individual endeavor or mixed with group feedback and collaboration the DIY ethos opens up three new pathways for engaging youth.

From building code to creating shareable applications. Programming that prizes coding accuracy and efficiency as signifiers of success is boring. To learn programming for the sake of programming goes nowhere for children unless they can put those skills to use in a meaningful way. Today children program to create applications like video games or interactive stories as part of a larger learning community.³ They are attracted to the possibility of creating something real and tangible that can be shared with others. Programming is not an abstract discipline, but a way to “make” and “be” in the digital world.

From tools to communities. Coding was once a solitary, tool-based activity. Now it is becoming a shared social practice. Participation spurred by open software environments and mutual enthusiasm shifts attention from programming tools to designing and supporting communities of learners. The past decade has brought many admirable introductory programming languages to make coding more intuitive and personal. Developers and educators realize that tools alone are not enough. Audiences are needed, and a critical mass of like-minded creators. Scratch, Alice, and similar tools have online communities of millions of young users. Children can work and share programs on a single website. This tacitly highlights the community of practice that has become a key for learning to code.

From “from scratch” creation to “remixing.” These new, networked communities focus on *remixing*. Students once created programs from scratch to demonstrate competency. Now they pursue seamless integration via remixing as the new social norm, in the spirit of the open source movement. Sharing one’s code encourages others to sample creations, adjust them, and add to them. Such openness heightens potential for innovation across the board. Young users embrace sampling and sharing more freely, challenging the traditional top-down paradigm.

These three shifts signal a social turn in K–12 computing. They move from a

How do we facilitate broader and deeper participation in the design of the programming activities, tools, and practices?

predominantly individualistic view to greater focus on underlying social and cultural dimensions of programming. We should rethink what and how students learn to become full participants in networked communities.

Broadening and Deepening Computational Participation


It is not possible to address all of the difficulties of implementing computational participation by placing students in groups, having them program applications, and encouraging them to remix code. Computational participation will present new challenges in bringing programming back into schools. How do we facilitate broader and deeper participation in the design of the programming activities, tools, and practices?

Computational thinking is a social practice. We must broaden access to communities of programming.⁴ Children are not “digital natives” who naturally migrate online. Establishing membership in the programming community is not easy. Groups with powerful learning cultures are often exclusive cultures. Students need strategies to cope with the vulnerability of sharing one’s work for others to comment on and remix.

In addition, students need a more expansive menu of computing activities, tools, and materials. Designing authentic applications is an important step in the right direction, but games, stories, and robotics are not the only applications for achieving this goal. We need different materials to expand students’ perspectives and perceptions of computing.

Broadening computational participation gets students into the clubhouse. The next challenge is to help them develop fluency that permits them to engage deeply, making their participation meaningful and enriching. These levels of computational participation are still rare. To learn to code students must learn the technicalities of programming language and common algorithms, *and* the social practices of programming communities.

Conclusion

Computational participation provides new direction for programming in K–12 education. It moves us beyond tools and code to community and context. It equips designers, educators, and researchers to broaden and deepen computational thinking on a larger scale than previously. Users of digital technologies for functional, political, and personal purposes need a basic understanding of computing. Students must understand interfaces, technologies, and systems that they encounter daily. This will empower them and provides them with the tools to examine and question design decisions they encounter. Computing for communicating and interacting with others builds relationships. Education activist Paulo Freire once said that “reading the word is reading the world.” He was right. Today, reading code is about reading the world. It is needed to understand, change, and remake the digital world in which we live. 

References

1. Grover, S. and Pea, R. Computational thinking in K–12: A review of the state of the field. *Educational Researcher* 42, 2 (2013), 59–69.
2. Kafai, Y.B. and Burke, Q. *Connected Code: Why Children Need to Learn Programming*. MIT Press, Cambridge, 2014.
3. Kafai, Y.B. and Burke, Q. Constructionist gaming: Understanding the benefits of making games for learning. *Educational Psychologist* 50, 4 (2015), 313–334.
4. Margolis, J. Estrella, E., Goode, G., Holme, J. J. and Nao, K. *Stuck in the Shallow End: Race, Education, and Computing*. MIT Press, Cambridge, 2008.
5. Palumbo, D.B. Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research* 60, 1 (1990), 65–89.
6. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.
7. Wing, J.M. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33–35.

Yasmin B. Kafai (kafai@gse.upenn.edu) is a Professor and Chair of the Teaching, Learning, and Leadership Division in the Graduate School of Education at the University of Pennsylvania.

Copyright held by author.