Education

Scale or Fail

Moving beyond self-selected computer science education in Switzerland.

K-12 COMPUTER SCIENCE Education (CSed) is an international challenge with different countries engaging in diverse strategies to reach systemic impact by broadening participation among students, teachers and the general population. For instance, the CS4All⁹ initiative in the U.S. and the Computing at School⁴ movement in the U.K. have scaled up CSed remarkably. While large successes with these kinds of initiatives have resulted in significant impact, it remains unclear how early impact³ becomes truly systemic. The main challenge preventing K-12 CSed to advance from teachers who are technology enthusiasts to pragmatists is perhaps best characterized by Crossing the Chasm,⁵ a notion anchored in the diffusion of innovation literature. This chasm appears to exist for CSed. It suggests it is difficult to move beyond early adopters (Figure 1, red and orange stages) of a new idea, such as K-12 CSed, to the early majority (Figure 1, green stage). Switzerland, a highly affluent, but in terms of K-12 CSed somewhat conservative country, is radically shifting its strategy to cross this chasm by introducing mandatory pre-service teacher computer science education starting at the elementary school level.

Three fundamental CSed stages, captured in Figure 1, are characterized by permutations of *self-selected/all* and *students/teachers* combinations. It took approximately 20 years to transition through these stages. Each stage is described here from a more general CSed perspective as well as my personal perspective.

Figure 1. Switzerland is crossing the computer science education chasm with mandatory pre-service teacher education.

Stage I: The “Friday Afternoon Computer Club” Stage (Self-Selected Students/Self-Selected Teachers): Stage I focused mostly on the “right” tools. In the 1990s the overall negative perception of programming by children, best described as “hard and boring,”⁷ suggested *cognitive* as well as *affective* challenges. Being constantly just one semicolon away from total disaster with traditional programming languages such as C or Pascal, is an example of syntax being a cognitive challenge. But even with the presence of early educational programming languages, such as Logo and BASIC, programming in schools was typically marginalized into after-school contexts such as Friday

afternoon clubs. These clubs, in turn, attracted only the usual suspects, that is, self-selected kids (mostly male) instructed by self-selected teachers (also mostly male). To make computing more relevant to schools it was crucial to focus less on tools supporting programming per se but to create tools to *forge explicative ideas through computing*, or as Papert started to call it, *computational thinking*.⁶

I developed *computational thinking tools* to allow students to get past the cognitive and affective challenges⁷ at the University of Colorado. Computational thinking tools have the ultimate goal to transform people’s perception of programming from “hard and bor-

ing” to “accessible and exciting.” K–12 CSed does not have the intent of producing a fleet of programmers but on producing computational thinkers. Students should be enabled by computational thinking tools to participate in activities including programming, such as creating STEM simulations or games, in the regular K–12 course context and not just at the Friday afternoon computer clubs. With Agent-Sheets we made programming more accessible by moving beyond the challenges of syntax through drag-and-drop programming.⁷ With AgentCubes we made programming more exciting^a to kids by providing them with browser-based tools to create 3D shapes, which they can 3D print, compose into interactive 3D worlds, and share with their friends.

Stage II: The “Professional Development Movement” Stage (All Students/Self-Selected Teachers). Once computational thinking tools sufficiently addressed the cognitive and affective challenges dimensions of CSed, it was time to shift the research focus from tools to curricula and teacher professional development. Researchers started not only to design, and conduct teacher professional development but also to systematically evaluate its efficacy. The teachers attending were mostly self-selected, but in many cases the classes they would teach were aimed at all students and no longer limited to after-school contexts. Stage II started to reach a much broader audience than Stage I.

I initiated Scalable Game Design as a strategy to teach computational thinking through game design and STEM simulation creation activities.⁸ Scalable Game Design is rooted in a didactic model called the Zones of Proximal Flow, which helps students to design their own games and STEM simulations based on the understanding of common computational thinking patterns.¹ Teacher professional development gradually scaled from local face-to-face summer institutes at the University of Colorado, to online and blended professional development, reaching schools in all 50 U.S. states.

K–12 computer science education does not have the intent of producing a fleet of programmers but on producing computational thinkers.

Private funding supported international Scalable Game Design sites, for example, Scalable Game Design Mexico funded by Google.

Stage III: The “Mandatory Pre-Service Teacher Education” Stage (All Students/All Teachers). Compelling curricula can spread through networks of excited CS teachers surprisingly quickly,² but how will they cross the CSed chasm (Figure 1) and persuade more conservative audiences? If society truly believes computational thinking is an essential 21st-century skill, just like mathematical thinking, then it will have to shift to transformative mandatory practices reaching all pre-service teachers.

In 2013, I began leading a pilot project in Switzerland to explore systemic impact on K–12 CSed through mandatory pre-service teacher education. This project engages all pre-service elementary school teachers at one of Switzerland’s largest Schools of Education, PH FHNW, in CSed, through two mandatory CS courses. Other Swiss Schools of Education, e.g., the PHSZ, are offering similar mandatory CS courses. The new Swiss Lehrplan 21, a national curriculum reaching the 21 German-speaking states out of all 26 states in Switzerland, enabled this kind of transformational practice because it mandates K–12 CSed. The transformative nature hinges on the possibility for students to fail becoming an elementary teacher based on failing a mandatory CS course. This unparalleled kind of causality resulted in government intervention. Four years of negotiations, where the societal ben-

efits of CS had to be explained to the Swiss Conference of Cantonal Ministers of Education, finally resulted in the accreditation of mandatory CS courses for elementary school teachers.

Scalable Game Design Switzerland Courses

Conceptually, the Scalable Game Design Switzerland course is rooted in the Scalable Game Design framework developed at the University of Colorado⁸ but it had to go through radical transformation to deal with the different needs of a non-self-selected audience. Less than 1% of these pre-service teachers had programming experience. The expectations of teachers (75% women, 25% men) varied widely with many anticipating learning how to use Word or PowerPoint. Instead, just like the original course, this new course focused on computational thinking conveyed through game design and STEM simulation building. To make the course relevant to pre-service teachers, Scalable Game Design Switzerland had to be carefully aligned with the learning goals defined as competences in the Swiss Lehrplan 21.

On September 17, 2017, in four states, seven instructors began to teach 26 CS courses with approximately 25 pre-service teachers each. All 600+ pre-service teachers must take two courses: Introduction to Computer Science and Computer Science Didactics. This totals four credit hours, resulting in over 25,000 contact hours. The Introduction to Computer Science course, which finished in December 2017, was based on three core concepts. Each concept is described here and includes a brief description of our preliminary assessment:

1. **Learning and Motivation Strategy: Scalable Game Design.** The strategy is to teach computational thinking through a series of computational thinking patterns⁸ that gradually—hence the notion of scalability—expose students to increasingly sophisticated game design and STEM simulation building challenges, following the Zones of Proximal Flow strategy.¹

The “scalable” aspect of the course worked well resulting in all pre-service teachers (students of the school of education) developing the necessary skills to program at least a basic game. Before the course, the acceptance of game design as learning strategy by teachers

a AgentCubes online is a 3D computational thinking tool with more than one million student projects created in 2017 in over 180 countries.

Figure 2. AgentCubes games designed and programmed by pre-service teachers.

was one of our biggest concerns. Course satisfaction overall was high with many even mentioning in their course evaluation that they liked programming games. Figure 2 shows sample games and simulations produced by teachers such as 1980s-era arcade-style 2D games (Donkey Kong), puzzle games (Tetris), educational simulation games (honey bee pollination), 3D indoor games, and 3D outdoor games. To design these games students had to engage in an iterative computational thinking process.⁷

2. Tools to support computational thinkers in schools: Computational thinking tools. We used our AgentCubes computational thinking tool because it had established accessibility even with younger kids, independent of gender and race,⁸ and it enabled kids to create interesting artifacts such as 3D games and STEM simulations. To cover the wide range of needs in elementary schools, we also included CS Unplugged activities, for classrooms lacking computers, and Processing, as brief exposure to more traditional textual programming.

There was a huge shift of perception regarding pre-service teachers' ability to program. Before the course essentially nobody had done any kind of programming nor did they think programming was particularly relevant to K–12 education. At the end of the first course all 600+ pre-service teachers

had created and programmed multiple games and participated in multi week game programming group projects. Everybody had gone beyond the minimal basics of programming, for instance by programming collaborate ghosts in a Pac-Man like game. Also, learning was not reduced to CS in isolation but included uses of computation in other subject areas such math, music, art, and social sciences.

3. Course structure: The seven big ideas of computer science. The mapping of the Scalable Game Design strategy onto the computer science part of the Lehrplan 21 was straightforward. The three main Swiss Lehrplan 21 CS topics (data, algorithms, and systems) were identified as a subset of the seven big ideas found in the AP Computer Science Principles framework: creativity, abstraction, data, algorithms, programming, the Internet and global impact. Each Computer Science Principle idea was mapped onto a two-week block in the 14-week course.

In spite of its density, most students really enjoyed this course structure. Students took particular pleasure in the Computer Science Principles ideas that were either evidently relevant to their lives such as the Internet, or conveyed through engaging activities such as a CS unplugged role play of people acting as bubbles in

bubble sort to understand the notion of an algorithm.

Scale or Fail?

The preliminary evaluation of the first part of the course (Introduction to Computer Science) suggests a shift in audience from *self-selected in-service teachers* to *all pre-service teachers* may not be as difficult as anticipated. However, to actually assess if we can teach these pre-service teachers to teach computational thinking to their students will not only require us to conduct the second part of the course (Computer Science Didactics) but also much more time to observe practical impact. We have an approach that has been successful in the past, and we have a broad, transformative experiment that could scale to an entire country. We have confidence that this approach will allow us to cross the CSed chasm.

References

1. Basawapatna, A., Repenning, A., Koh, K.H., and Nickerson, H. The zones of proximal flow: Guiding students through a space of computational thinking skills and challenges. In *Proceedings of the International Computing Education Research (ICER 2013)*, San Diego, CA, USA, 2013.
2. Bradshaw, P. and Woollard, J. Computing at school: An emergent community of practice for a re-emergent subject. In *Proceedings of the International Conference on ICT in Education*, Rhodes, Greece, 2012.
3. Brown, N.C.C., Sentance, S., Crick, T., and Humphreys, S. Restart: The resurgence of computer science in U.K. schools. *Trans. Comput. Educ.* 14 (2014), 1–22.
4. Crick, T. and Sentance, S. Computing at school: Stimulating computing education in the U.K. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli, Finland)*, 2011, 122–123.
5. Moore, G.A. *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*. HarperBusiness, New York, 1999.
6. Papert, S. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning* 1 (1996), 95–123.
7. Repenning, A. Moving beyond syntax: Lessons from 20 years of blocks programming in AgentSheets. *Journal of Visual Languages and Sentient Systems* 3, (2017), 68–89.
8. Repenning, A. et al. Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *Transactions on Computing Education (TOCE)* 15 (2015), 1–31.
9. Vogel, S., Santo, R., and Ching, D. Visions of computer science education: Unpacking arguments for and projected impacts of CS4All initiatives. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Seattle, WA, USA.

Alexander Repenning (alexander.repenning@fhnw.ch) is Professor and Chair of Computer Science Education at FHNW, School of Education, Windisch, Switzerland, and Professor of Computer Science at the University of Colorado at Boulder, Boulder, CO, USA.

This research was supported by the U.S. National Science Foundation, the Swiss National Science Foundation, and the Hasler Foundation.

Copyright held by author.