Comparing Block-based, Text-based, and Hybrid Blocks/Text Programming Environments in Introductory Computer Science Classes

David Weintrop
Northwestern University

Computation is changing our world. From how we communicate and how we make decisions, to how we relax and how we shop - few aspects of our lives have been left unaffected by the long reach of computation and the technologies that it enables. Smartphones, tablets, and laptops have become the lenses through which we see, organize, and interpret the world. As such, for young learners growing up in this technological landscape, being able to recognize the capabilities and limitations of these technologies, and most critically, to be able to contribute in this technological culture is essential. Programming is the skill that enables this participation. Programming, and the critical thinking and problem solving skills that accompany it, constitute a new 21st century literacy that will need to live alongside reading, writing, and mathematics as essential competencies to empower today's students to fully engage with our technological world. These skills have far reaching benefits as they underpin and enable new forms of creative expression, support learning in diverse computational contexts across a wide range of disciplines, and provide the foundation for future careers in our increasing computationally driven economy. The bureau of labor statics estimates that 135,000 new computing jobs are created every year in the technology sector. Similar growth of computing jobs is projected in other fields; by 2020, one in every two jobs in the STEM disciplines will be in computing (ACM Education Policy Committee, 2014).

Despite this momentous shift happening in our world and the far-reaching benefits that accompany learning to program, very little programming education can be seen in today's classrooms. Computer science, the field that is driving this computational revolution, is rarely present in K-12 education. Only an estimated 10% of schools offer programming or computer science courses (Code.org, 2014), and in schools where computer science is present, courses are often taught in ways disconnected from the computational lives of today's students and leave the learner without the feeling of empowerment that can and should accompany learning these skills. Further, the students who have the opportunity to pursue programming do not reflect the racial and gender distribution of the larger population. In 2013, 14.6% of bachelor's degrees in computer science and related fields were granted to female students, with 4.5% of the graduates being African American, and 6.5% being Hispanic (Zweben & Bizot, 2014). This disturbing trend is mirrored at the high school level, where only 18.6% of students who took the 2013 AP Computer science exam were female, while 8.2% of test takers were Hispanic, and only 3.7% were African American. Research into the cause of these low numbers has identified numerous causes, including limited access to courses, a lack of support for students who express interest in the field, and cultural issues that make underrepresented populations feel unwelcome (Margolis, 2008; Margolis & Fisher, 2003).

Numerous national efforts are underway to address the lack of computer science learning opportunities for both underrepresent minorities and the student body at large, that utilize innovative materials, engaging pedagogy, and new tools and environments for students to learn the concepts (Astrachan & Briggs, 2012; Goode et al., 2012). Central to these initiatives is the use of new, more convivial forms of programming that emphasize personal expression, align with current youth culture, and draw on prior student knowledge and values. These new programming environments designed to make the concepts more approachable, intuitive, and engaging for learners. Questions on the effectiveness of these new tools with respect to how they affect learners' perceptions of the field, the feelings of confidence and identification as programmers they engender, and understanding how these new programming environments affect students' comprehension of computer science concepts are of critical importance to this effort and are the focus of this dissertation study.

This dissertation will answer three sets of interrelated research questions all of which address different facets of the guiding question: How best can we educate the next generation of computationally literate citizens? The first set of questions investigates the relationship between the representations students use while learning to program and the resulting attitudinal and conceptual outcomes. This is important as new environments for teaching programming are emerging and becoming increasingly used in formal educational settings, but we lack a clear understanding of the relationship between these new tools and the resulting conceptual gains, attitudinal outcomes, and programming practices they promote. Research towards this end has identified that representational tools greatly affect the learning process and outcomes (diSessa, 2000; Green & Petre, 1996; Sherin, 2001; Wilensky & Papert, 2010), but little work has been done on the current generation of programming environments with respect to these questions.

The second set of research questions looks at the suitability of these new introductory programming approaches for preparing learners for future computer science learning opportunities. Research is emerging that suggests that many of these introductory tools, while successful in changing attitudes and engaging learners, do not adequately prepare them to transition to more conventional programming languages, thus imposing an artificial ceiling on how far learners can progress with these tools (Cliburn, 2008; Garlick & Cankaya, 2010; Parsons & Haden, 2007; Powers et al., 2007). This finding is consequential as it calls into question the utility of such introductory tools in the first place. The work done to date has largely provided descriptive accounts of learners failing to transfer knowledge and practices from introductory environments to more sophisticated, powerful tools. This dissertation will contribute detailed accounts of students transitioning from introductory to professional programming environments, and provide mechanistic, theoretically sound cognitive explanations of if, how, and why gains made in introductory environments do or do not transfer to more sophisticated programming tools.

The final set of research questions surround the evaluation of a new hybrid introductory programming environment that will be designed and implemented as part of this dissertation. The new environment will blend the strengths of various existing programming tools in an effort to create a tool that provides the low-threshold to entry and high level of engagement of existing introductory approaches, with the high-ceiling and powerful expressivity of more fully featured programming tools. Based of findings from the first two sets of questions, the goal is that this new programming tool can serve as the central environment upon which new introductory computer science curricula can be built.

This dissertation is built around a 3-condition, quasi-experimental study comparing three introductory programming tools – two environments are exemplars of common approaches currently used in introductory programming contexts, and the third will be a new environment developed as part of the study. The study will take place over 20 weeks in three introductory programming classes at two Chicago public high schools with diverse student populations. Beginning on the first day of school, students will spend five weeks working through a custom designed curriculum using one of the three introductory programming environments. At the conclusion of the fifth week of school, all three classes will transition to the Java programming language and follow the same curriculum for the remainder of the year. With this study design we can directly compare the effectiveness of the three different introductory environments, as well as, answer questions about their suitability for preparing students for future learning as we follow the students through their transition to the Java programming language. The study uses a mixed-methods approach and will include qualitative, quantitative, and computational data collection and analysis techniques. During the 20-week study, we will observe classrooms on a weekly basis, conduct student interviews, collect and analyze student-authored programs, and administer pre/mid/post content assessments and attitudinal survey to answer the stated research questions.

We are at a critical juncture in the history of computer science education in this country. The ability to program is a central skill all students should develop, but it is currently absent from the coursework of today's students. To address this gap, educators, school administrators, and state and national legislators are all taking action to bring computer science into the classroom. The practices, tools, and curricula that are being developed today, will become the standards used for years to come. Therefore, it is critical that we are confident that the curricula and environments that we advocate for today are effective at teaching the core concepts, engaging learners from diverse backgrounds, and successful in preparing students for the computational endeavors they will face in the future. The findings from this dissertation will advance our understanding of how best to introduce students to these core 21st century skills and contribute new tools that will prepare students to be successful in the computational futures that await them.

References

Astrachan, O., & Briggs, A. (2012). The CS principles project. *ACM Inroads*, *3*(2), 38–42.

Cliburn, D. C. (2008). Student opinions of Alice in CS1. In *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual* (p. T3B–1). IEEE.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Education Policy Committee, A. (2014). *Rebooting the Pathway to Success Preparing Students for Computing Workforce Needs in the United States*. Association for Computing Machinery.

Garlick, R., & Cankaya, E. C. (2010). Using Alice in CS1: A quantitative experiment. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 165–168). ACM.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: the exploring computer science program. *ACM Inroads*, *3*(2), 47–53.

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A "cognitive dimensions" framework. *Journal of Visual Languages and Computing*, *7*(2), 131–174.

Margolis, J. (2008). *Stuck in the shallow end: Education, race, and computing*. The MIT Press.

Margolis, J., & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. The MIT Press.

Parsons, D., & Haden, P. (2007). Programming osmosis: Knowledge transfer from imperative to visual programming environments. In S. Mann & N. Bridgeman (Eds.), *Procedings of The Twentieth Annual NACCQ Conference* (pp. 209–215). Hamilton, New Zealand.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. *ACM SIGCSE Bulletin*, *39*(1), 213–217.

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning*, *6*(1), 1–61.

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulating knowledge disciplines through new representational forms. In J. Clayson & I. Kallas (Eds.), *Proceedings of the Constructionism 2010 conference*. Paris, France.

Zweben, S., & Bizot, B. (2014). 2013 Taulbee Survey. *COMPUTING*, *26*(5).