White Paper: Nonvisual Access to Computation and Programming

Clayton Lewis
University of Colorado, Boulder
clayton.lewis@colorado.edu
October 25, 2013

*Research questions:* How can access to computation and programming be extended to people who cannot see well? How can new representations of data, programs and their operation be developed that make programming easier to understand for people generally, including people who cannot see well, or cannot use common pointing devices?

*Background:* Common efforts to make programming easier to understand (e.g. Scratch, Alice, AgentSheets, LabView) move away from textual representations of programs toward "visual" representations. These languages and environments are difficult or impossible to use for people who can't see well, or for whom common pointing devices don't work because of motor impairments. While some efforts have been made to help students with disabilities use these tools (Adams, 2009; Touretzky et al., 2013), the tools themselves remain “inaccessible”, that is, difficult for blind learners to use, and it is proving difficult to add accessibility supports to the existing tools (Ludi et al., 2009).

To improve this situation we need to explore how the potential virtues of "visual" representations can be delivered for people with visual impairments. After all, if these program representations have the pedagogical advantages their proponents hope for (and their wide acceptance suggests that they do) students who cannot see well should not be denied these benefits.

Creating nonvisual visual programming tools is not the contradiction in terms that it might seem to be. Prominent blind computer scientist T V Raman of Google has pointed out (personal communication; see Lewis, 2013) that the visual system can be thought of as a way to answer questions against a spatial database (the spatial world); if users are provided alternative ways to ask these questions and receive the answers, they do not need vision (for related ideas see Raman and Gries, 1997, and Raman, 2006). This insight urges that we should separate the *conceptual structures* offered by "visual" programming representations from the *presentation* of these structures, which need not be visual. For example, Lewis (2013) describes a proof of concept system in which a declarative representation of a dataflow program can be accessed either via a conventional visual depiction, or by a completely nonvisual representation that uses keyboard commands with speech output to create, edit, and execute programs. These dataflow programs are similar in concept to those in LabView, one of the languages used with Lego Mindstorms robotic kits. Similarly, it should be possible to create alternative, nonvisual presentations of Scratch programs, or programs in AgentSheets. In this way we can bring the conceptual benefits of these tools to students who cannot see well. A research agenda is needed to pursue these opportunities, exploring and exploiting the benefits of Raman's insight for programming education.
This research should be combined with an increased investment in enhancing existing text-based languages and tools to make them more useful for students who cannot see well, as for example in the work of Stefik and colleagues (Stefik and Gellenbeck, 2009; Stefik et al. 2011). As Stefik et al. (2009) point out, the traditional approach to accessibility has been the use of screen reader tools, that read textual material aloud, but this approach works poorly for program code. As they argue, consistently with Raman’s insight, we should not limit blind students to using representations crafted for sighted users, but should explore a range of representations explicitly intended for nonvisual access.

An allied line of research explores nonvisual access to computational products like information visualizations. While very useful work is being done in developing cross modal access to visual content (for example by using acoustic or tactile representations of spatial arrangements commonly accessed visually; see for example Petit et al., 2008) there is a need for research that presses Raman's insight in this domain, as well. We need to be studying how we can move beyond providing alternative access to visual forms, to providing access to the content that underlies the visual forms. This opens the way to many new ways of thinking about data and its meaning, as for example in Walker (2013). Since applications that produce graphical output are commonly used to make introductory computing more engaging (as in Guzdial, 2003), we need to develop these approaches for use in pedagogical settings in CS, as well as elsewhere.

The critically important underlying theme in all this work is the separation of content and presentation. By moving beyond familiar presentations of programs and data, spurred by the need to create new presentations that work for a wider range of users, we enter new spaces for design, with enormous potential for productive innovation.

**Impact of this research:** A primary benefit of this research is, of course, greater inclusion of students with disabilities in CS education. The social and economic impact of this is large, both in terms of an expanded technical workforce, and in terms of greater economic participation by people with disabilities, for whom technical careers offer many potential advantages.

The benefits extend further, however. We have ample evidence that the common representations of computational processes available to us pose challenges for many typical learners, and so research that explores and develops new representations has great value. It is a commonplace, but nevertheless actually true, that designing against severe and unusual constraints is often profoundly creative, and produces unexpected insights. Research on nonvisual programming should be conducted with a view to securing benefits for typical learners, as well as those with disabilities, through the development of novel representations of programs and computational processes, and new ways to interact with these representations.

Pushing the work beyond "computer science education", narrowly conceived, to address wider public literacy and competence in computation and programming, offers further benefits. Today, people with disabilities are often dependent on technically capable
people without disabilities to identify and meet their needs for new applications for their phones, tablets, and computers. This is an inefficient, lossy process, with many potential opportunities not identified, and much effort diverted into products that don't actually meet the needs of their intended users. Developing programming tools that can be understood and used by people with disabilities themselves will greatly improve the situation.

References


