Understanding the Connection between CS and Other Domains
Colleen M. Lewis
Assistant Professor of computer science
Harvey Mudd College
lewis@cs.hmc.edu

Guiding Question:
How do competencies within computer science build upon and relate to competencies within other domains?

This guiding question provides the opportunity to:

- Reject the idea that computer science ability is innate, which is a barrier to diversifying computing,
- Connect computer science education with education research, and
- Improve the teaching of computer science at all age levels.

Overview:
Computer science education research has documented the pattern that many students are not successful learning to program (e.g., McCracken et al., 2001). However, it remains an open question what non-programming experiences could prepare students for success learning to program (Simon et al., 2006). I believe it is necessary to investigate the question of what experiences students bring to the computer science classroom, how they can contribute to success, and how computer science pedagogy can take advantage of them. The hypothesis that students have untapped resources upon which we can transform undergraduate computer science education runs counter to what may be a common assumption among computer scientists of the existence of an innate aptitude that determines students’ success learning to program (Robins, 2010; Lewis, 2007). This guiding question, provides opportunities to improve the teaching and learning of computer science and to build mutually beneficial connections to educational research. Additionally, this line of work could provide explanations that counteract the assumption that computer science ability is innate.

Rejecting the idea that computer science ability is innate
Developing a better understanding of how students learn computer science may be the only way to counteract the problematic assumption of an innate ability for computer science.

While it is possible that students could have a genetic predisposition to program computers, this is currently an untested assumption, which can have real consequences and can play into self-fulfilling prophecies (Dweck & Legget, 1988; Steele, 1997). Even if we assume that many students lack the intellectual resources to become as successful as Alan Turing, it is reasonable to hope to connect all motivated students with an environment in which they can become competent at programming. In contrast, Simon et al. (2006) summarize an ongoing research direction that I believe may be an outgrowth of a dearth of explanations of why some students are less successful.
“The literature abounds in assertions of the existence of an aptitude for programming, and of attempts to find a suitable predictor for that aptitude so as to avoid wasting time and effort educating students who are unlikely ever to become good programmers.” (Simon et al. 2006)

The assumption of an innate aptitude is often implicit and is made explicit in more subtle ways. For example Lister et al. (2004) describe why differences in innate talent at various institutions constitute a complication in analyzing the study’s multi-institutional data.

“Clearly, some institutions attract students with a greater innate talent for programming.” (Lister et al., 2004)

In multi-institutional studies it is arguably relevant to discuss differences in the student populations. However in this statement the authors indicate an otherwise unstated assumption that there exists an “innate talent for programming”.

In my related research, I have documented students’ beliefs about whether or not computer science ability is innate and how the environment of an introductory programming class shapes these beliefs (Lewis, Yasuhara, Anderson, 2011). There were cases in which participants endorsed the existence of an innate ability that demonstrated how this belief can discourage persistence and exclude students that are underrepresented within computer science. The student quoted below attributed the idea that computer science ability is innate to her introductory computer science professor. She said:

“Even my [UA-CS2 professor] told us that some people are just born that way, with that mental outlook that is compatible with CS... They feel it’s so easy for them... Yeah, and he told the rest of the people that some of you will try but some of you won’t get it, and it’s just that your mental outlook isn’t made that way. It’s something you’re born with. You can’t help it” (p. 6, Lewis, Yasuhara, & Anderson, 2011)

Another Participant in this study said that she thought female students might be less innately abled at computer science and said that few women in the field might be evidence of this lack of an innate ability (Lewis, Yasuhara, & Anderson, 2011).

Based upon research from Dweck and her colleagues (see Dweck & Leggett, 1988 for a review) and Steele and his colleagues (Steele, 1997; Carr & Steele, 2009) there are negative consequences for students when their success or lack of success is framed as indicative of innate aptitude. If students come to believe that there exists an innate aptitude for computer science they may adopt a mindset that can stifle their academic growth (Dweck & Leggett, 1988; Simon et al., 2008).

I perceive no larger barrier to diversity within computer science than the assumption that there exists an innate ability for programming. This idea, either publicly or privately, could be used to justify the continued underrepresentation of people of color, women, and people with disabilities.
Connecting computer science education with education research

The proposed line of research could identify connections across educational research within other domains and the current late introduction of students to computer science provides a unique opportunity for educational research.

The proposed research follows a long line of research investigating prior knowledge from other domains (diSessa, 1993; diSessa & Sherin, 1998; Wittman, 2001; diSessa & Wagner, 2005; Wagner, 2006; Parnafes, 2007; Levrini & diSessa 2008; Hammer, 2000; Russ & Sherin, 2008) and developing transformative pedagogy (diSessa & Minstrell, 1998). This prior work provides methodological examples of how to identify students’ knowledge resources and beliefs that play a role in learning. More generally these researchers engage in the enterprise of studying conceptual change and attempt to understand the dynamic process of thinking and learning in the domain of physics. An emphasis on identifying connections with other intellectual domains may strengthen the connections to educational research within these domains. For example, in my dissertation (Lewis, 2012, NSF #: 1044106) I applied learning theories many of which had not previously been applied to the domain of computer science.

Few students have the opportunity to learn computer science before attending college. Unlike other intellectual domains, such as mathematics or history, many students’ first introduction to computer science is in college. Certainly some students have access before college, but the inequality of access perpetuates existing barriers to students underrepresented within computer science (Margolis et al., 2008). However, with this missed opportunity of early learning comes a unique opportunity for educational research. As an educator and an educational researcher I often work with students who are first learning computer science in college and therefore I have the opportunity to observe students engaged in learning within a domain for which they are both ignorant and potentially well prepared by their other academic experiences. However, some students with what we believe to be adequate preparation and motivation are not successful. This current situation provides an interesting context for educational research.

Improving the teaching of computer science at all age levels

This line of work has the potential to transform computer science education through identifying and building upon students’ strengths to ultimately support the success of more students.

It is an open question what non-programming experiences may support success learning to program (Simon et al., 2006). Educational research from a wide variety of fields argues that students’ prior knowledge must be taken into account (Ben-Ari, 2004; Clancy, 2004; diSessa, 1993; Fleury, 1993; Soloway, Bonar, & Ehrlich, 1983; Vosniadou & Brewer, 1992). There is also strong support for the assumption that without understanding the interplay between non-programming knowledge and the learning of programming, pre-programming and programming instruction at best will be impoverished and at worst will fail (Soloway, Bonar, & Ehrlich, 1983; Fleury, 1993; Ben-Ari, 2001; diSessa, 1986; Vosniadou & Brewer, 1992; diSessa, 1993; diSessa & Wagner, 2005).
Pennington (1987) found that the most successful programmers were those who frequently made connections between the program text and the non-programming or real-world goals. Based upon this finding, I hypothesize that students will be less successful if they fail to connect their programming knowledge to their prior non-programming knowledge.

In a related project (Lewis, Yasuhara, & Anderson, 2011), we found that students frequently describe computer science as unconnected to their previous ways of thinking. For example, one student said, “It’s like a different way of thinking. Like it’s really confusing. You have to get used to it.” Another similar sentiment, “I feel like you shouldn’t do it unless you like—unless you’re like more attuned to that kind of thinking. If you don’t think that way, it’s just going to be really difficult for you.” Students appear to believe that their existing ways of thinking are not relevant to learning to program and that to be successful in computer science they have to adopt a completely new way of thinking. This model of adopting a new way of thinking rather than adapting your current thinking may be a significant barrier to students making productive use of prior knowledge.

**Previous Research**

My dissertation (Lewis, 2012, NSF #: 1044106) took a first step toward this set of research questions.

To investigate what out-of-domain knowledge supports students’ success, I conducted a detailed analysis of students’ reasoning on computer programming questions that were identified by previous research as highly correlated with success on the AP CS exam (Reges, 2008). The participants were college students enrolled in one of three introductory programming courses at the University of California, Berkeley. As such, these students had been successful in their previous academic pursuits and could be expected to have a variety of out-of-domain knowledge, some of which may be relevant to reasoning about computer programming problems.

I found that participants appeared to build upon several kinds of out-of-domain knowledge. For example, many students used algebraic substitution techniques when tracing the state of recursive functions. Students appeared to use metaphors and their intuitive knowledge of both iteration and physics to understand infinite loops and base cases. On the level of an individual students’ reasoning, a case study analysis illustrated the ways in which a participant integrated her linguistic knowledge of “and” into her reasoning about the computer science command “and.”

**Conclusion:**

This line of work is built upon the constructivist assumption that prior knowledge can serve as a significant support for learning computer programming. This may be able to drastically improve the teaching and learning of computer science. Additional connections to education research within other domains provides mutual benefit. Perhaps most important, this line of research can work to dispel the assumption of an innate ability for computer programming.
References


reading and tracing skills in novice programmers. Working group reports from ITiCSE on Innovation and technology in computer science education, 119-150.


