Why Don't Some CS0 Students Succeed? How Important Are Background, Experience, Culture, Aptitude, Habits and Attitude?

Daniel D. Garcia (moderator)
University of California, Berkeley
777 Soda Hall
Berkeley, CA 94720
+1 (510) 517-4041
ddgarcia@cs.berkeley.edu

Colleen Lewis
Harvey Mudd College
Department of Computer Science
Claremont, CA 91711
+1 (909) 607-0443
lewis@cs.hmc.edu

Stuart Reges
University of Washington
Box 352350
Seattle, WA 98195
+1 (206) 685-9138
reges@cs.washington.edu

Nathan Ensmenger
Indiana University
School of Informatics & Computing
Bloomington, IN 47408
+1 (812) 855-0705
nensmeng@indiana.edu

Keywords
Computer science education, hidden prerequisites, litmus test

1. SUMMARY
There are always some students who succeed and some students who don't. Our four panelists are committed to the success of all students, but have different explanations for students' lack of success. This panel discussion will highlight both their shared beliefs and disagreements between veteran CS educators Stuart Reges and Dan Garcia, CS education researcher Colleen Lewis, and Professor of History and Philosophy of Science Nathan Ensmenger. We hope this lively discussion will bring together divergent and complementary positions and expertise, as well as invite significant audience participation.

2. DANIEL D. GARCIA
After ten years of teaching our non-majors introductory computing classes, to hard-working students who have never programmed before, I have noticed several common features of students who succeed: **logical thinking** (e.g., debugging skill, or noticing that `XOR(a,b)` is really just `a!=b`), **problem solving** (e.g., knowing when a solution is a dead-end, how to back up to the last decision branch, where that branch is, and what the other possibilities are from that branch), **lateral "out of the box" thinking** (e.g., able to generate creative, non-linear solutions), **persistence in the face of challenge** (i.e., "grit"), **a love of "tinkering"** (i.e., pleasure in casually playing with an artifact), **comfort in a world of unknowns balanced with a drive to make sense of it** (e.g, getting a new electronic toy with lots of buttons and wanting to try each one out), and the ability to **abstract** (removing irrelevant details, generalizing something by noticing common patterns, etc.).

I believe all students who put in the time and effort can succeed in introductory computing classes (and beyond), and when some don't, it's *our* fault. We should make sure the climate doesn't have bias and is fully conducive to their learning. We may also be operating under a false assumption that all students have all the features listed above. Those who did succeed may have honed these skills in another context and transferred their knowledge over! We should make these prerequisites explicit, and/or provide activities in our classes to help develop them.

3. STUART REGES
My 27 years of teaching programming to novices has left me with some deep intuitions about how people learn to program. I share Don Knuth's belief that there is a mode of thinking that is particular to computer science (CS) and that some students have a greater aptitude than others. As Knuth has written, "I conclude that roughly 2% of all people 'think algorithmically,' in the sense that they can reason rapidly about algorithmic processes" [5]. My own intuition about this is that there are students who think this way naturally, but we can build up this ability in a broad range of students with exercises that allow them to practice this thought process. I have presented some questions that I hypothesize measure what Knuth calls "algorithmic reasoning" [9], although another study failed to replicate the result [8], so there is still no conclusive evidence to support this intuition that many of us have.

Whether or not CS aptitude exists, most of us who teach introductory CS classes observe wide variation in how students take to programming. Many students who expect to enjoy it because they enjoyed using video games or other software are surprised to find that they don't enjoy the process of programming. Other students who had no clue that they would enjoy programming are surprised to find that they thoroughly

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA
ACM 978-1-4503-3685-7/16/03.
http://dx.doi.org/10.1145/2839509.2844667

enjoy the mental challenge presented by it. Often it is men who are disappointed and women who are pleasantly surprised, although surprises happen in the other direction as well.

Even though I believe that some students more easily pick up programming than others, I also believe that we can design introductory courses in which every student can succeed. Instructors should:

- present coding topics incrementally with lots of examples
- provide a strong safety net for students who need help
- point out common programming patterns and include exam questions that reward students who learn these patterns
- include non-programming, "mechanical" questions on exams
- emphasize that effort is at least as important as aptitude in predicting success

Of course, you don't want to tailor your course too much for the students who struggle because you want to make sure that there is enough intellectual content to attract the students who end up loving it. This is a delicate balancing act that I feel I have struggled with throughout my career.

4. COLLEEN LEWIS

A common occurrence in the United States is that students arrive at college without programming experience. At many institutions, these students have a rich set of skills and deep knowledge in domains other than CS. Certainly, educating these students should be easier than educating younger, less skillful, and less knowledgeable students. Unfortunately, as a CS education community, we have few techniques for building upon students' out-of-domain knowledge and skills. Even worse, we have no clear understanding of how, when, and why various out-of-domain knowledge and skills are productive in CS.

My hypothesis of what makes students successful is that they, without being instructed, apply the right out-of-domain knowledge and skills. Other students may have the right knowledge and skills, but do not apply them. My research [7] attempts to identify productive out-of-domain knowledge and skills for reasoning about computer programs. If we better understand what is required to master computer programming, we can make these requirements explicit and support them in developing these competencies.

The hypothesis that there exists an innate ability for CS is flawed and deeply problematic. In decades of research, there have been no factors that reliably predict an individual's success learning to program [10]. Why would there exist an innate ability, uncorrelated with other abilities, for a task (CS) that did not exist one hundred years ago? The potential for discouragement and discrimination in assuming there exists an innate ability for CS seems obvious. As evidence of this potential for discouragement, students' belief in an innate ability for CS appears to be consequential for students' decision to major in CS [6]. In addition, it has been shown that students who believe intelligence is innate avoid taking intellectual risks [3], which is the exact opposite of the behavior we want from our students.

We are far too young as a field to assume that students who are not successful cannot be successful. As a community we should start identifying relevant out-of-domain knowledge and skills and developing teaching techniques to build upon them.

5. NATHAN ENSMENGER

From the establishment of the Curriculum '68 [1] guidelines to the present, there has existed a long-running debate within the academic CS community about what skills, abilities, training (and even personality type) are required to be a successful computer scientist. I am especially interested in the relationship between the formal curriculum in CS education and the informal subculture that is often associated with the computing professions. I argue that the distinctive cultural norms and practices of certain computing communities, which draw on or appeals to tacit knowledge, skills, and other cultural affinities independent of the actual intellectual content of the CS curriculum, send subtle message to potential CS students about who does (and, more importantly, does not) belong. Educators who are sensitive to the influence and effects of this subculture can use it to encourage a broader and more diverse range of students. This is particularly true in regard to gender diversity, an issue of concern to many CS departments.

Much of my historical research focuses on the role of women in computing, and I find that including this history in the curriculum both helps with the recruitment and retention of women, but also encourages all students to appreciate the value of a broad and inclusive perspective on their discipline [4]. There is a also growing body of research that suggests that not only can historical case studies be useful in teaching core concepts in CS, but that they can help students better appreciate the value – to themselves, and to society – of a CS education.

6. REFERENCES

[1] ACM Curriculum Committee on Computer Science. 1968. Curriculum 68: Recommendations for Academic Programs in CS. Comm. ACM 11, 3 (Mar. 1968), 151-197.

[2] Briggs, A. and Snyder, L. 2012. Computer science principles and the CS 10K initiative.ACM Inroads 3, 2 (June 2012), 29-31. doi.acm.org/10.1145/2189835.2189847

[3] Dweck, C. (2007). Mindset: The new psychology of success. New York, NY: Random House, Inc.

[4] Ensmenger, Nathan. "Making Programming Masculine." In Gender Codes: Why Women Are Leaving Computing, edited by Thomas Misa, Wiley, 2010.

[5] Knuth, D. 2004. Selected Papers on Computer Science. CSLI.

[6] Lewis, C. M., Yasuhara, K., & Anderson, R. E. (2011). Deciding to Major in Computer Science: A Grounded Theory of Students' Self-Assessment of Ability. Proceedings of the International Computer Science Education Research Workshop. Providence, RI. 3-10.

[7] Lewis, C. M. (2012). Applications of Out-of-Domain Knowledge in Students' Reasoning about Computer Program State. (Doctoral dissertation). Retrieved from ProQuest Dissertations and Theses. (Accession Order No. 12710).

[8] Lewis, C. M., Khayrallah, H., & Tsai, A. (2013). Mining data from the AP CS A exam: patterns, non-patterns, and replication failure. Proceedings of the International Computer Science Education Research Workshop. San Diego, CA, USA. 115-122.

[9] Reges, S. The mystery of "b := (b = false)." Proceedings of the 39th SIGCSE technical symposium on Computer Science education.

[10] Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. CS Education, 20(1), 37-71.