article:1929

Use and Assessment of a Rigorous Approach to CS1

When we initiated this research, an introductory computer science course for CS majors ("CS1") was typically the first computer programming experience for undergraduate students (although this observation is less true today, due to increased exposure to technology in and before high-school). The critical role of CS1 at the entry to the curriculum led the CS education community to invest great energy in the design of the CS1 course. This energy is most readily seen in CC2001[1], Chapters 6 and 7: no fewer than six different approaches to the introductory course were suggested.

Just after the release of CC2001, the reported dot-com bust and perceived loss of professional opportunities in computing correlated with a dropoff in enrollments in CS1 throughout the US. Undergraduate computing programs looked for ways to respond by making CS1 accessible and attractive to their students. Allen Tucker's SIGCSE 2001 keynote address, "From Rigor to Rigor Mortis"[2], was a call to the community to avoid sacrificing rigor for access. This keynote provided the focus of our work: to make our rigorous approach to CS1 accessible to a wide variety of students.

We had originally envisioned our course as appropriate for students with significant experience in mathematics and a high degree of comfort with proof techniques. Our course did not shy away from difficult problems, for it here where the true power of computation can be seen. Furthermore, we wanted students to experience the importance of deep and varied thinking about problem solving. To this end, our CS1 course exposes students to two programming paradigms, starting with the pure functional approach and leading into the imperative approach. It illustrates a number of software design techniques, such as top-down design, abstracting and generalizing from examples, recursive design, design of loops from their invariants[3], etc.. Throughout the course, we couple each discussion of a programming skill or language feature with a discussion of appropriate reasoning tools (e.g., pre- and post-conditions for pure functional code; loop invariants for imperative code; an informal discussion of static single assignment to relate the two).

We saw the post-dot-com-meltdown enrollment decline as an opportunity to convey the existing concepts to new student populations without over-crowding our labs. We worked to make our challenging lab assignments as engaging as possible, and to create opportunities for students to succeed, while retaining the fundamental character and chellenges of our course. Our approach was multi-faceted:

* We developed new lectures about writing proofs and developing loop invariants.
* We reduced the number of proofs required of the students, while leaving the degree of difficulty the same. (An attempt to teach students to come up with loop invariants without requiring that they write loop proofs was abandoned by faculty after one try.)
* We added to the appeal of our lab projects by replacing some assigmnents with more fun examples of the same concept (such as "help the mouse escape from the (acyclic) maze" for the design of loop-based functions).

* We added visual interest to our labs through increased use of graphics (for the mouse in the maze, numerical simulation of heat flow, and computational geometry problems).
* We provided a number of new supports for students who needed extra help, including an evening faculty-led open lab hour, increased training of teaching assistants, and printed course notes covering our approach to the harder material[4].

More recently, we have also switched from C++ to Python, which seems to produce less student frustration in many (but not all) aspects of programming. (This switch and our MacOS/linux transition demonstrate the "portability" of our approach.)

Against this background, we conducted a study of the effectiveness of loop invariants as an approach to support student understanding of the development of loop-based algorithms. While our sample size (7) was too small to claim statistical significance, we did find that the students who started by writing the logical statements performed best, despite taking the smallest amount of time for testing, of the three groups we studied.

[1] 2001. Computing curricula 2001. J. Educ. Resour. Comput. 1, 3es (Sep. 2001), 1. DOI= http://doi.acm.org/10.1145/384274.384275.
[2] Tucker, A. "From Rigor to Rigor Mortis: Avoiding the Slippery Slope". Keynote Address, SIGCSE 2001, Charlotte, North Carolina, available at http://www.bowdoin.edu/~allen/sigcse2001/.
[3] Gries, D. "The Science of Programming". Springer-Verlag 1981.
[4] Wonnacott, D. "From Vision to Execution: The Creative Process in Computer Science". http://www.lulu.com/content/1094615.

Author 1: John P. Dougherty jd@cs.haverford.edu

Author 2: David G. Wonnacott davew@cs.haverford.edu

Article Link: http://portal.acm.org/citation.cfm?id=1047431&coll=portal&dl=ACM&CFID=9122936&CFTOKEN=32438184

: Back to 2009 Spring/Summer Issue Vol. 4, No. 3

: Back to List of Issues

: Back to Table of Contents