

# SUNY Geneseo's Three-Fold Introduction to Computer Science

Doug Baldwin

SUNY Geneseo

Computer science is a thinking discipline — to be a capable computing professional, you must be able to think clearly and effectively about problems, algorithms, programs, etc. Such thinking involves more than writing programs. It also involves understanding the role of problems and algorithms alongside programs; it involves understanding the mathematical concepts that support specification and analysis of problems, algorithms, and programs; it involves understanding how to empirically test theoretical beliefs against the real world. In an effort to establish our students as thinkers, and to expose them early to the full set of mental tools available for thinking about computing, SUNY Geneseo teaches an introductory computer science sequence based on the methods of inquiry that characterize computer science [Baldwin 1994].

Geneseo's introductory sequence emphasizes three methods, namely

**Design** of algorithms and programs, i.e., the planning and creation of algorithms and their translation into programs.

**Theoretical Analysis** of problems and algorithms, i.e., the use of mathematical tools to specify problems and to determine whether an algorithm correctly and efficiently solves them.

**Empirical Analysis**, i.e., the use of experiments and the scientific method to verify theoretical predictions.

These three methods correspond closely to the three “paradigms” described by Denning et al. in [Denning 1989]; while many curricular responses to that report introduce students to computer science's breadth of topics, Geneseo is unique in introducing students to the field's breadth of methods of inquiry.

The first course in our sequence introduces students to all three methods. This course has a prerequisite of “any college-level computing course”, meaning that students enter with some amount of computing maturity, although not necessarily with programming experience. The prerequisite also means that most CS majors take this course in the second semester of their first year. The course begins with short introductions to objects and messages, to sequence and conditional control structures, and to pre- and postconditions as specifications of what an algorithm must do. The bulk of the course is an intensive introduction to recursion, including its use to define and design algorithms and data structures, and the mathematical analysis of recursive algorithms (specifically, induction for reasoning about correctness, and recurrence relations for deriving execution time formulae). These

analyses are supported by empirical tests of execution time in the form of experiments that quantitatively compare measured execution times to derived formulae, and with empirical tests of correctness in the form of systematic testing and debugging strategies. The course takes an object-oriented view of algorithms, but uses class libraries to allow students to be mainly users of objects rather than definers of classes. Students do, however, define subclasses to extend the abilities of library classes — for example, one of the libraries provides simple list and binary tree classes, which students subclass to create searchable lists and trees (using recursive list and tree search algorithms).

The second course consolidates students' understanding of the methods of inquiry while looking "inside" many of the concepts introduced in the first course. For example, the second course begins to look more thoroughly at the design of classes and standard data structures, and at some of the underlying implementation ideas (e.g., pointers). This course is also the first place where we "officially" admit that iteration exists as a control structure, building on what students already know about analysis of recursion to analyze iteration (e.g., induction and loop invariants to reason about correctness, finite series to analyze execution time).

Geneseo has taught this sequence since the 1992-93 academic year. The sequence successfully teaches students the rudiments of all three methods of inquiry, and is now a well-established part of our major. Over the years, however, there has been considerable variation between instructors in the relative emphasis placed on programming versus non-programming topics, particularly in the first course. This means that students enter the second course, and the rest of the CS major, with widely varying abilities, depending on who taught their introductory courses. To solve this problem, we are changing the prerequisite for the first course to a specific introductory object oriented programming course. This change means that students will enter the sequence with a stronger and more uniform programming background, reducing the pressure on instructors to teach programming at the expense of other methods (and in fact removing some of the initial programming material from the first course). The change also means that the sequence will become CS2 and a follow-on course, whereas now it is best described as CS 1 1/2 and CS 2 1/2.

## References

- [Baldwin 1994] D. Baldwin, G. Scragg, and H. Koomen. "A Three-Fold Introduction to Computer Science", Twenty-Fifth SIGCSE Technical Symposium on Computer Science Education, Mar. 1994. pp. 290–294.
- [Denning 1989]. P. Denning et al. "Computing as a Discipline", *Communications of the ACM*, Jan. 1989 (32:1). pp. 9–23.