Incorporating Writing Into The CS Curriculum

Lisa Kaczmarczyk (Moderator)
University of Texas at Austin
Austin, TX, USA
lisak@cs.utexas.edu

Dian Rae Lopez
University of Minnesota, Morris Campus
Morris, MN, USA
lopezdr@mrs.umn.edu

Gerald Kruse
Juniata College
Huntingdon, PA, USA
kruse@juniata.edu

Deepak Kumar
Bryn Mawr College
Bryn Mawr, PA, USA
dkumar@cs.brynmawr.edu

SUMMARY
There are many different views about how to incorporate writing into the computer science curriculum. These views lead to inevitable questions about process and goals. For example, should writing be taught as a separate course or integrated into traditional content classes? How much latitude should students be given in selecting writing component courses? Should writing focus on developing research skills, workplace skills or personal growth? Who should perform assessment: individual faculty, small teams or the whole department? How can a CS educator assess writing ability – where does one even start? Our panel members will answer these questions differently, sharing both successes and challenges. We represent a large public research university, a small public liberal arts university, a private women’s liberal arts college, and a small private co-educational liberal arts college.

Categories and Subject Descriptors
K.3[Computers and Education]:Computer & Information Science Education – Computer Science Education, Curriculum.

General Terms
Documentation, Human Factors

Keywords
Writing, Pedagogy, Communication

1. LISA KACZMARCZYK
Many of the CS students that I have taught do not initially believe communications skills are important to their future careers. In the Technical Writing class that I currently teach at the University of Texas at Austin, I have turned this problem into an opportunity[3]. Whether students plan a career in industry or academia, the analysis and production of formal writing represents a unique opportunity to provide concrete evidence that a CS student should care about their writing. My class is an upper division elective, one of several writing component courses available to CS students.

In order to make the case for writing skills as explicit as possible, the course is divided into “academic” and “industrial” parts, each lasting half the semester. In the academic part of the course, undergraduates get a glimpse of the world of research and publication, and a gentle introduction to journal articles. Students read peer reviewed articles and analyze their writing for strengths and weaknesses. By having to study the writing of professional computer scientists, students learn about how structure and rhetorical technique can enhance or detract from an argument. In the process, they discover that technical merit is not the only criterion determining how others perceive their work.

This insight is linked directly to expectations for their own writing. Each student presents their analyses as formal essays, which are assessed primarily on how well they are written and how logically they are argued. Assessment uses “Minimal Marking”[2], a system which not only reduces hours spent grading, but requires students to interact deeply with their writing in order to improve it. Every student has the opportunity (strongly encouraged and supported) to rewrite and improve their papers, and hence their grade. Much of in-class time is spent dissecting and discussing pieces of student papers and journal articles.

In the “industrial” part of the course, students do a project with a partner. The team designs a software user's guide, which I make clear, is primarily a vehicle for learning about “assessment in the real world” and acquiring communication survival skills. Much of the industrial part of the class is devoted to learning to identify personal assumptions about computer science and technology that may interfere with professional success. As part of designing their own manual, students evaluate existing documentation of all types to identify overt and covert assumptions of audience and context. They tailor technical information for various audiences, such as peers, co-workers, bosses, and clients. Students make regular class presentations, which become more formal as the semester progresses. These presentations often involve describing their current successes and challenges with the project so that classmates can provide insight and share ideas.

2. GERALD KRUSE
I teach Analysis and Algorithms as a writing course (where 25% of the grade depends on written work). The rich content of the course, which includes studying algorithms, their complexity, and
supporting data structures, lends itself well to wonderfully descriptive narratives. Typically I assign the students two or more writing projects per semester. In these projects they explore open-ended questions, where they are required to choose, implement, use, and compare various algorithms. For each project, they turn in a first draft, which I evaluate and return with comments, before their final draft is due. I also meet with each student to discuss their writing style and other subjective content. While it might sound daunting, meeting with the students actually reduces my overall time commitment, since it helps to improve the quality of their final draft and makes it easier to grade.

When I assess student work I strive for an objective and repeatable assessment. This can be a difficult goal given the subjective nature of most writing assignments, so I attempt to separate the assessment of the objective components of their work from the subjective components. When I share my grading rubric beforehand, the students will focus on the objective content of their work, making sure what they say is true and grammatically correct. In my experience it is easier for students to improve a mechanically correct, but dull first draft, than a very readable first draft with serious errors in their process. Since this is an advanced course and they take it after completing several other CS and writing courses, the students are more willing to write in their own voice, rather than writing what they think I want to hear.

3. DIAN RAE LOPEZ

Presentation, both oral and written, is an integral part of the computer science curriculum at the University of Minnesota, Morris Campus (UMM). In fact, it is required that all students at UMM meet a “writing requirement” in their major field of study. To meet this requirement, two courses are taught, Seminar I and Seminar II (one credit each), which emphasize writing. Both courses are required in the major.

In both courses, students write papers on a topic of their choice. They first learn how to search the literature, then how to read a scientific paper, then how to synthesize the information and write about it professionally. Zobel’s Writing for Computer Science – The Art of Effective Communication [4] is a very helpful resource. Their drafts are “marked up” with lots of questions and ideas for improvement. An assessment instrument is used in both classes by students and faculty to assess the presentation skills of each student. Both courses meet once a week with the professor giving assignments and exercises to help them succeed in their writing endeavors.

The first course, Seminar I, is taken during the sophomore year. Student groups do research in the field of computer ethics and present their results both as papers and as presentations. The last few weeks of the semester are set aside for students to present their work in a professional manner to the class.

The second course, Seminar II, is taken during the senior year. Each student chooses a research topic and a computer science professor familiar with that field to help them prepare their research paper. The topic chosen must have recent published research results. The student struggles with reading these papers (the professor can help tremendously here – one on one meetings are held once a week in addition to class time). The student then writes a professional paper using the ACM Guidelines for journal paper submission. Each student is assessed individually on how much they have improved during the two courses and all faculty in the department must be convinced that the student is prepared to use the skills learned in future endeavors. Although students often write in other courses, these two courses concentrate on this aspect of their education.

4. DEEPAK KUMAR

We consider Writing (and reading) an essential part of a computer scientist’s training. The practice of programming (or writing programs) has much to gain from the practice of writing (essays) and vice versa. Both activities require structure analysis and iterative refinement. Further, in our curriculum, writing is at the core of ALL course work [1]. Besides programming/lab projects and exams, grades in writing exercises form a good percentage of a student's overall course grade. Writing exercises vary from simple reflective essays on every programming assignment, to the writing of technical papers as the end result of a programming assignment where the actual code is included in the appendix.

All of our courses include, in addition to covering the core content, “humanizing” readings that help introduce the culture, relevance, and excitement in computer science. For example, consider the course, Principles of Programming Languages. This is considered a core course in computer science. It is largely concerned with the principles underlying the design of programming languages. Traditionally flooded with technical content, the course has been redesigned to include case studies of practitioners’ lives. There are several biographies and essays available that describe the lives of computer scientists who could be classified as “programming language experts”. Some of them have been designers of prominent programming languages, while some have led commercial ventures based on programming language products. Thus, a “humanizing element” is included in the course without necessarily sacrificing the technical matter. Students are required to write reflective essays on these readings which are then discussed during class. Such activities help broaden the context of computer science itself and help eliminate the perception of the field as one which is attractive only to “coding geeks”.

5. REFERENCES