IT'S ALL WRITING: EXPERIENCE USING REWRITING TO LEARN IN INTRODUCTORY COMPUTER SCIENCE[*]

Dr. Brian C. Ladd
Mathematics, Computer Science and Statistics Department
St. Lawrence University
Canton, NY 13617
(315)229-5471
blad@it.stlawu.edu

ABSTRACT

Writing computer programs is writing to communicate with a human and a machine audience. This fact is often lost on students in introductory computer science courses. At St. Lawrence University, an effort has been made to use techniques developed as part of the writing across the curriculum movement such as portfolio evaluation and multiple drafts of written work to improve student outcomes in introductory computer science courses. Motivation and evaluation of the St. Lawrence experience is reported.

INTRODUCTION

As the Pragmatic Programmers remind everyone involved in software development, "It's all writing."[Hun99] To date, computer programs and their documentation are written communications between humans and other humans and machines. Recognizing this permits instructors to appropriate teaching methods from the writing across the curriculum (WAC) movement in an effort to improve student outcomes in introductory computer science courses.

Advocates of WAC offer several techniques such as portfolio evaluation and multiple drafts of work that can be fruitfully applied in introductory computer programming courses. Portfolio evaluation has students examine their work and reflect on their learning style so that

[*] Copyright © 2003 by the Consortium for Computing in Small Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing in Small Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

they take ownership of their learning process. Multiple drafts of a single assignment gives students a chance to revise their work; they begin to see that editing is an integral part of producing quality documents or software. Reflecting on learning and guided revision can inform the practice required for students to learn problem solving and how to express solutions in a given programming language.

This paper reports our experiences applying a multi-draft model of programming in CS1 and CS2 courses. The next section gives a brief overview of writing across the curriculum and how St. Lawrence University uses it, followed by a survey of some related efforts in using writing tools in programming courses. The next two sections report and evaluate our experience using writing across the curriculum tools in computer science.

EDUCATIONAL CONTEXT

Writing across the curriculum (WAC) is a movement that has, for a quarter century, studied how students learn using writing; WAC advocates have then pushed for "students [to] use written language to develop and communicate knowledge in every discipline and across disciplines."[You99] At St. Lawrence University, the goals of WAC have been manifest in the First Year Program (FYP), a residential educational program with a strong writing component. All first-year students receive instruction in how to use writing as they learn and in improving writing through rewriting. This foundation is built upon in the many writing intensive courses offered across the curriculum. For almost two decades, the FYP has used all of the writing assignments described below and recruited instructors from across all departments, providing training in WAC. The idea of applying rewrites and portfolios to introductory computer courses came while reading the instructor's training materials for the FYP and hearing about their successes.

WAC and the associated writing center movement have developed many useful techniques: one-minute essays, journals, multiple drafts for formal writing, and the use of portfolios. These assignments reflect the observation that writing serves at least two different purposes: writing to learn and writing to communicate[You99]. Writing to learn is less formal and designed to promote active learning through student conversation with themselves. Writing to communicate is the writing students to do communicate the knowledge they have gained.

The assignments cataloged above emphasize different types of writing. One-minute essays are directed, one minute summaries in writing of what the student knows or thinks they know; a journal is an on-going personal conversation about learning. These are both examples of writing to learn and encourage student reflection on learning as a process as well as on the material being learned; reflection on learning is has been a focus of other introductory computer science innovations[Fek00].

Multiple drafts of a formal writing assignment are done to improve the final product through guided revision. They also demonstrate that rewriting is the primary tool for producing good writing. Teaching the "real-world" method of writing improves students' understanding of the process. It is designed to improve their mastery of writing to communicate. Portfolios are a collection of examples of the student's writing, selected by the student in consultation with the instructor, along with reflection on what the writings represent; programming portfolios have been used in the computer science classroom [Est01].

St. Lawrence University is a small liberal-arts university with twenty years of experience with a combined computer science/mathematics degree program and a four year old computer science major. The introductory CS1/CS2 sequence is divided into three courses with the second course focusing on a semester long project with multiple phases. There will be more on the introductory sequence below.

This section has mentioned several WAC techniques but is by design incomplete. Interested readers are urged to review the WAC literature or find a writing center near them. A survey of related efforts to apply some of these ideas to computer science courses is the next section and the following sections describe experiences using WAC in introductory computer science courses.

RELATED EFFORTS

Learning problem solving and computer programming is a complex, involved process. It requires practice and assessment that encourages active engagement with the material. It also requires broader assessment criteria than a single final exam. Computer science instructors, departments, and institutions have addressed these needs with explicitly reflective writing and programming portfolios.

In [Fek00] the authors report success in using reflective writing assignments in large introductory classes. Using WAC names, they assign and assess journals to promote student reflection on what they do at each stage of the course and what they have learned. They also have students assess programs/documentation using the same criteria they are graded on. The only negative reported is student opinion on the amount of work the journal systems is. This work is related to the current work in goal but not in specifics; St. Lawrence University courses are much smaller with fewer teaching resources so the same tools cannot be applied.

Programming portfolios are similar to artistic portfolios, a collection of the programmer's best work. While the traditional programming portfolio is static, a notebook filled with program listings and snapshots of program execution, recent efforts have focused on creating dynamic programming portfolios. These dynamic portfolios are easy for students to update and easy for instructors and others to review because they are typically implemented on the World Wide Web.

One of the best known implementations of electronic portfolios though not specifically in computer science is the RoseE-Portfolio system[Rog99] at the Rose-Hulman Institute of Technology. Students are happy with the ease of updating their portfolio and the ability to link it into their resumes. Faculty are happy with the ease of access to the contents and have been using the portfolios to enhance student outcomes across the curriculum.

The best known application of electronic portfolios to computer science is the Interactive Programming Portfolio[Est00][Est01] (IPP). The IPP is also successful because of the ease of insertion of projects and ease of review. Student work in their growing portfolio is open for review from across the Web. The IPP provides links to evaluation rubrics to guide reviewers' comments into a constructive vein. As reported, the IPP is effective for assessment and for spurring educational development in students who use it.

Both of these portfolios permit students to submit, reorder, and replace work. Neither is focused on documenting multiple drafts of a single project. The next section reports experience with rewriting to improve student outcomes and student satisfaction; the following section evaluates that experience.

THE USE OF REWRITING AND PROGRAMMING PORTFOLIOS

This section describes the motivation for using rewrites and portfolios in introductory courses and then the techniques we use. Rewriting programs is important for students because it is important to programmers in the "real world". It is also highlights the level of confidence the student has with various concepts and gets them to reflect on what they have learned. Portfolios serve a similar purpose of self-examination.

Integrating revision into the introductory computer programming course is important to combat compartmentalization. Without integrated revision, students compartmentalize program revision into something done in a software engineering course, believing that programs spring fully formed from the brows of programmers; their misguided perfectionism leads to disappointment when they cannot produce programs in this way. Without integrated revision, students compartmentalize each programming assignment as an isolated event unrelated to any other program; revision attempts to inspire reflection on the process. Without integrated revision, students compartmentalize each part of a single program, requirements, design, documentation, and code, as disjoint assignments; revision integrates feedback across all of these parts so that students recognize them as a related whole.

Revision in computer programming is the norm rather than the exception. Many software engineering textbooks remind developers that 40-60% of the expense of a software product is spent in the maintenance phase [Som00][Gla02]; the maintenance phase is when requirements change and the software is revised. Revision in computer science education is, however, rare. This problem is exacerbated by sample programs given to the students in class or in their texts. These programs are the final product of a lot of work either by the instructor or by the textbook author and her editor. Students never see or experience those efforts.

Integrating revision into programming assignments also combats student belief that all course programs are "toy" or busy work. The programs are still simplified for time and pedagogy but the student cannot finish the program and forget all about it. They will get a copy of the program back with constructive comments (and a draft grade). They will be expected to address the comments in their revision. Further, combined with the use of a course portfolio, students are expected to address these comments in an additional, consciously reflective, essay. Tying the reflection directly to the program in the form of rewrite comments and the reflective essay, students reflect on the kind of assignments they are doing and the kinds of solutions they have learned to produce. This is just the kind of active learning we want to inspire.

This pattern of writing combined with rewriting combined with a course portfolio is one students have seen before in their FYP. By consciously mimicking the FYP model we emphasize that it's all writing. Students see that every document they produce is part of the whole of the solution. Style and substance are both addressed in the draft comments and both must be addressed in rewrites and the portfolio; integration of all facets of the student's writing in the assessment points the student toward integrating it in their learning.

The curriculum committee had much of this in mind when spreading the standard CS1/CS2 sequence across three semesters and designed the second course around a substantial project with multiple phases. Students must turn in solutions to each phase of the project but they cannot just put them on the shelf when they are done. Each phase builds on one or more that came before so students are continuously challenged to improve their earlier solutions so that the whole project is improved. The second course is thus focused on software reuse and similar software engineering topics. Portfolios and rewrites are used extensively in the first and third semesters of the sequence.

Students in the first course, Introduction to Computer Programming, write a sequence of separate programs. Before rewriting was introduced, there were about twelve per semester and they were due every week. The programs were turned in, graded, handed back, and the topic covered was often put to rest until the exam. Now there are fewer projects but each has not one but two due dates. The first draft is due, is graded and handed back, and then the student typically has another week to rewrite the program. Due to the sheer volume of topics to be covered, first drafts of some programs are sometimes assigned between the first and second draft of another program. Students are required to turn in their marked up first draft with their second draft along with a page on how they addressed the evaluation comments.

Students maintain a hard-copy portfolio containing all drafts of all projects. At the end of the semester they select several assignments that illustrate trends they see in their work and write a 300 word essay on what those assignments show about what they learned. They are required to turn in all their projects but only the ones they choose are considered part of their portfolio.

In the third course, Data Structures, this same model is followed but the size of the individual programs has grown. The amount of programming that the students must do with the different data structures they need to see led to a severe time crunch when rewriting was introduced. We still use portfolios with required reflective essays but the student and faculty workload required to rewrite every project has meant that we only rewrite one or two per semester. Some ideas for addressing this will be mentioned in the future work section below.

Convincing students that it is all writing must extend beyond source code so half way through the first course students must include design documents talking about the decisions they made and code they reused. A portion of a lecture describes the intended audience for a design document and gives them a standard format to follow. These documents are expanded through the second and third courses to include additional levels of detail. Treating the design document as a necessary part of any solution, a part that must be rewritten with the rest, integrates the writing across the entire development process.

One administrative note: it is much easier to enforce a rigid lateness policies when students know they will be rewriting their work than when something is completely done at the deadline. Students come to recognize that writing a program is an ongoing process and that the first set of design decisions is seldom optimal. They know that they are expected to revise and therefore are more comfortable handing in a draft that mostly works. This mindset has been fostered in their FYP as well. As described in the next section, this can be a blessing and a curse.

The next section looks at how well this approach has worked and the following section discusses how to address the shortcomings and build upon the successes.

EVALUATION

Students start the FYP seeing multiple drafts as some sort of busywork. With directed revision they learn to use rewriting as a time to reflect on how well the original writing worked and try to improve it. That same pattern has emerged in the introduction to computer programming though the benefits of rewriting computer programs are not as clear to the students.

Students typically start the course with little experience writing computer programs. This makes introductory programming different than college-level writing. This lack of experience makes students happy to have the opportunity to rewrite their programs to improve their grades. At the beginning of the semester, then, the majority of the "reflection" amounts to cataloging the errors that were found in their first draft.

Later, with some guidance in class and in comments on second drafts, students begin to look for patterns in the types of mistakes they make. This is actually very useful to the instructor because the students sometimes see patterns that are missed in grading each program individually. This permits the problem to be addressed either one-on-one with the student or in class if there are widespread misunderstandings. Students find an increased satisfaction with the grading and then, often, with the material itself.

Teaching upper-level courses, we sense an increase in retention of higher-level concepts such as parameters, functions, and objects from the introductory courses. There is no corresponding improvement in retention of syntax and other low-level details. The improvement, if there is one, is so far only anecdotal.

Some students take their job of rewriting more seriously than others so some find it to be busy work or just a way to protect their grade (work hard on rewriting programs with poor first drafts and let the good first drafts stand on their own for second drafts as well).

This benefit is not without cost. It takes a great deal of faculty time to grade each assignment twice. Weaker students find the task of addressing many major shortcomings in a first draft daunting and often feel that the "red marks" are directed at them personally. Some of the best students are not well served because they do have previous experience and their first drafts are often very, very good.

The problems with the best and the worst students are problems found in other applications of the WAC model[You99]. They are difficult to address but the discussion below of more peer evaluation should help address them, at least for the best students. That faculty investment is increased in this plan is systemic and if that investment leads to more active learning and better student outcomes, it is worth it. That is not to say that more cannot be done to streamline the process; the next section addresses where we hope to go with this.

FUTURE WORK

There are three areas to address as we move forward with this teaching model: formal evaluation of student success, addition of Web-based portfolio management, and adoption of additional WAC techniques, particularly informal or writing to learn assignments. The previous section reports how students and faculty feel about the use of rewriting assignments and course portfolios. The student opinions are based on required course evaluations that directly ask about student satisfaction. A more formal understanding of educational outcomes would justify the extra effort faculty are investing. Of particular interest is how this model prepares students for the core courses of the major such as programming languages.

Given IPP and RosE-Portfolio, it would seem to make sense to move to an interactive programming portfolio. This might limit the extra work needed by faculty and it would certainly make portfolio evaluation easier since students will not have to turn in their sole copy. This requires some work to get the on-line evaluation tools in place so it is not clear how soon it can be undertaken.

WAC recognizes writing to learn and writing to communicate. Some advocates have begun to discuss the range of writing as a continuum rather than two distinct points with writings for peers (and much electronic writing) as somewhere between the two poles. In any case it seems likely that students would benefit from computer programming equivalents of more informal writing assignments such as the one-minute essay (the one-minute design: take one minute to write down your design for the current programming problem [thanks to Dr. Collen Knickerbocker]) and letters about programs, one to another. WAC advocates also push for peer assessment, an area to exploit in computer science. It might relieve faculty workload and it will improve student to student communication leading to yet another channel of writing.

CONCLUSION

Our experience with rewriting and portfolio grading in introductory computer science courses has been positive. Faculty work harder but so, it seems, do students. They learn that revision is a normal part of programming and they become more comfortable with debugging. As predicted by writing across the curriculum advocates, multiple drafts on formal writing projects help students understand what they are saying and why they are saying it. Students learn to think about what they do, that there is no such thing as perfect code, and to plan for change, three more useful tips from the Pragmatic Programmers to coders, old and new[Hun99].

REFERENCES

[Est00] Estell, J. "Programming Portfolios on the Web: An Interactive Approach". *Journal of Computing in Small Colleges*, 16(1) pp 55-67. CCSC. 2000.

[Est01] Estell, J. "IPP: A Web-based Interactive Programming Portfolio". SIGCSE 2001, pp149-153. ACM. 2001

[Fek00] Fekete, A, *et. al.* "Supporting Reflection in Introductory Computer Science". SIGCSE 2000, pp144-148. ACM. 2000.

[Gla02] Glass, R. *Facts and Fallacies of Software Engineering*. Addison-Wesley. 2002.

[Hun99] Hunt, A. and Thomals, D. *The Pragmatic Programmer*. Addison-Wesley. 1999.

[Rog99] Rogers, G. and Williams, J. "Building a Better Portfolio". *ASEE Prism*, 8(5). pp30-32. ASEE. 1999.

[Som00] Sommerville, I. *Software Engineering*, 6E. Addison-Wesley. 2000

[You99] Young, A. *Teaching Writing Across the Curriculum*, 3E. Prentice Hall. 1999.