Rewrite Cycles in CS Courses: Experience Reports

Linda Null
Penn State Harrisburg
Middletown, PA 17057
null@trex.hbg.psu.edu

Mike Ciaraldi
Worcester Polytechnic Institute
Worcester, MA 01609-2280
ciaraldi@wpi.edu

Liz Adams
James Madison University
Harrisonburg, VA 22807
adamses@jmu.edu

Ursula Wolz
The College of New Jersey
Ewing, NJ 08628-0718
wolz@tcnj.edu

Max Hailperin (Moderator)
Gustavus Adolphus College
St. Peter, MN 56082
max@gustavus.edu

1 Summary

The generally accepted wisdom among teachers of English composition is that a "rewrite cycle" should be used as a teaching strategy. Rather than expecting students to extrapolate from the grading comments on paper N what they should do differently on paper $N + 1$, it has become conventional to ask for a rewrite of paper N itself. Of course, there are many variations on this theme. For example, peer review may supplement or replace some of the rounds of grading.

The panelists will explore some of the ways they have applied this pedagogic strategy in computer science courses. Most obviously, when we have our students write papers, we have them do rewrites. However, some of us don't stop there, but rather also apply the same idea to the writing of programs or mathematical analyses. This encourages our students to do a high-quality job, and to feel that they have truly mastered a topic.

Clearly, there are tradeoffs and difficulties, principally involving time. The panelists will also discuss this aspect, indicating how they have coped with the pitfalls, and indicating what has worked well, and what not so well. Ultimately, however, all the panelists are optimistic about the value of rewrite cycles.

After the panelists share their experiences, there will be some time for discussion with the audience. Based on the interest this topic provoked on the sigcse.members mailing list, and in ensuing private email, we look forward to an active audience. We will also make materials from the session available at http://www.gustavus.edu/∼max/rewrite/

2 Linda Null

The Pennsylvania State University recently required all programs to include a minimum of one writing intensive class in each major. Writing-intensive courses require multiple writing assignments. Most importantly, opportunities for students to receive written feedback from the instructor and to apply the instructor's feedback to their writing must be built into the course.

As someone with an undergraduate English degree, I have been instrumental in modifying some of our computer science courses for this new designation. I will discuss how we have incorporated the required writing components into various classes. The challenge lies in using writing to help students explore and understand the concepts of the particular course. In addition, these writing courses should provide students with the necessary writing skills to function in today's society. Writing in computer science must be more than simply the mechanics of grammar and spelling. Style, reasoning ability, and organization skills must also be addressed.

Writing need not be limited to traditional term papers; other activities are typically far more interesting. Possibilities for projects with rewrites include: technical article summaries, software specification documents, short evaluations of computer science seminars and colloquia, written analyses of student presentations or simulated student interviews, proposals for funding potential projects, lab reports, progress reports or work logs, summaries of previous class presentations, summaries of class debates, software documentation, case studies for software projects, and critiques on particular software packages. With most of these assignments, students are allowed to evaluate the work of their peers in addition to revising their own writing after instructor comments.

3 Mike Ciaraldi

I have used an extended version of the rewrite cycle in a software engineering course. The students revise a requirements document three times, incorporating feedback from both grading and peer review, in the following sequence:

1. I give out the general outline of a project. Each student must submit a requirements document. I grade these and give them back.

2. Students form groups. Each group turns in one revised requirements document, incorporating the best ideas from the members and informed by my comments. I grade these and give them back.

3. Each group turns in a revised requirements document and a specification for a software package meeting these requirements. I grade these and give them back.

4. Each student gets a blinded copy of another group's requirements and spec. The student (acting as client) comments on it and says whether the developers should go ahead with the project. I grade these critiques and give the groups blinded copies of them.

5. The groups produce an updated requirements document and spec for the whole project, and a detailed design of one subsystem. I grade these and pass them back.

6. The groups implement the subsystem. The last day of class, each group presents its implementation to the class. I grade the implementation and presentation. Next time, I will probably do as I have done in some other courses, and ask the audience to grade the presentations too; the group grade would be based partly on the audience evaluation.

4 Liz Adams

Rewrite cycles are a valuable teaching technique even when the students are writing in a programming language, rather than in English.

I've used it in two different ways, most recently in my second semester Java course where the students wrote a program to play Caribbean Stud Poker against the dealer. Each student was asked to place the class files for his/her "draft program" on a disk. I collected the disks and placed them in the disk drives of other students. The students did not know whose disk they received. They were asked to run the programs and to make written suggestions for how the program could be improved. The suggestions and the disks were returned to the program authors who then had the opportunity to make corrections before submitting the program to me for a grade. After I graded the programs, students were again given the opportunity to correct the defects I identified during the grading process. Thus the full process was write, be peer reviewed, rewrite, be graded, rewrite again, and be graded again. In addition, since I had two separate sections working on the program, I also ran initial drafts from the other section in class and projected them for class discussion.

The other way I incorporated a rewrite cycle (in a data structures class) was to have students take the program they'd gotten the lowest grade on during the semester and rewrite it and submit it as the final program.

5 Ursula Wolz

The analogy between teaching programming and teaching English composition can be extended beyond just rewriting. Good writing is a six phase cycle (where you can loop through the middle four steps indefinitely). Adapted to programming, this cycle becomes:

Free write: Given a problem spec, use a design strategy to flesh it out. Write words, not code. Talk about the design with collaborators.

Rewrite: After the ideas are on paper, begin to organize them into objects. Minimize writing of code. Use diagrams for interrelationships.

Edit: Read through the design looking for inconsistencies, flaws, or blatant misunderstandings. Find an editor (e.g. a collaborator) to read through it.

Polish: Write code, modifying for clarity. Reorganize objects if necessary.

Critique: Run it. Make notes on what works, what doesn't, what could be improved. Go back to the rewrite stage.

Finish: The previous phases naturally generate documentation. Go back and make sure that the documentation does indeed reflect the code.

Grading is problematic with this approach, because the stages are a process intended to proactively develop habits. Only toward the end of the semester do students really rewrite well. Furthermore, evaluating the stages would be an overwhelming grading task. But without accountability, students will ignore the stages. To keep them accountable, I refuse to help with problems unless I see evidence of English (natural language) notes. In our lab environment where they do much of their work under my supervision, such informal assessment goes far to encourage them to developing a writing cycle.

6 Max Hailperin

I will describe several semesters of experiences with "mastery homework," in which each homework problem is graded on a binary scale (mastered vs. not-yet-mastered) and can be handed in as many times as it takes to reach mastery, any time over a broad period (sometimes even the full semester). The homework portion of the course grade is simply the percentage of problems eventually mastered. I generally turn each problem submission around in less than a day. Grades of "not-yet mastered" are invitations to come talk. I've used this not only for problems that call for writing prose or code, but also on problems where the primary work product is setting up some equations and deriving a numerical answer. A colleague has also used it for proofs.