IMPLEMENTATION OF PROGRAMMING STANDARDS IN A COMPUTER SCIENCE DEPARTMENT

Lionel E. Deimel, Jr.
Mark Pozefsky

Computer Science Department
North Carolina State University
Raleigh, North Carolina 27650

1. Introduction

As programming has evolved from a black art practiced by eccentric geniuses to a common occupation of ordinary persons, the importance of program structure and programming style has been increasingly recognized. Although there is surely no "theory of programming" yet, there is enough concensus about what techniques have and have not proven effective that this knowledge is being codified and widely disseminated.

The teaching of programming in the universities has undergone enormous changes during this evolution. Structured programming is almost universally accepted and taught in academe. Largely because of the structured programming movement, the effectiveness of teaching in programming courses has improved, particularly for the mediocre student. Evidently universities are not doing a sufficiently good job, however, because there are still complaints that computer science students in upper level courses cannot program. Many students ask "if the program worked, why didn't I get an A?". Students are confused over what is required of them, and plead that "Professor X said last term to do it this way."

Agreement about general programming methodology is relatively easy to reach within a single computer science department. However, individual notions concerning the details of that methodology are likely to differ significantly. Agreement to teach "structured programming," for example, leaves many issues undecided, and different teachers may handle these issues in different ways. Such diversity leads to confusion among students, who must resolve the different points of view. Too many instructors expect students to learn their particular documentation and programming style through osmosis. Instructors will have honest differences: What are acceptable forms of indentation? What should appear in the comments at the top of a procedure? The exact resolution of such issues is less important for the student than that they be resolved.

2. Programming Standards

The problems we have cited stem for a lack of explicit agreement within the faculty concerning the finer details of programming methodology and an effective method of communicating these details to students. These problems can be alleviated only by establishing a written set of departmental programming standards. Furthermore, the standards must be accepted by all instructors, so that they not confuse the students but instead give them a sense of uniformity and continuity within the curriculum.

What sort of standards are appropriate for this purpose? We feel the standards should be broad, so that they are applicable to a variety of languages and circumstances, although certain language-specific rules will also have to be established. The standards should not be so restrictive as to discourage a student from developing his own style or to unduly constrain instructors. Such standards might, for example, encourage the use of a particular indentation rule but should also suggest acceptable variations of this rule. It is important the student explore different styles, but we should create an environment in which he does not do so randomly.

The standards should establish rules for deciding on the appropriate control constructs to be used (when to use an iterative DO as opposed to a DO...WHILE, for example). Proper use of data types should be addressed (say, the use of fixed-point rather than floating-point counters). The standards should demand that a program contain comprehensible procedure, section, line, and declaration comments, and that it utilize mnemonic label and variable names. User orientation and program generality can be encouraged by requiring (in general) echoing of data, input and parameter validation, readable output, parameterized algorithms, and so forth.

3. Enforcement

Codifying a set of rules without enforcing it is probably even worse than having no rules at all. Scanning only a program's output and checking that it is "structured" without exhaustively examining the code and documentation is like reading an English theme for content while ignoring organization, spelling, and punctuation.

A line-by-line program examination obviously takes significantly more time and effort than is typically expended on program grading. We believe this more extensive quality control can be adequately provided by intelligent, well-trained student grading assistants who can evaluate an algorithm's design, the program documentation, and even its coding.

Obviously such strict quality control has its costs. First, the department must be able to agree upon a written set of documentation and programming guidelines. Individuals may have to relinquish some personal rules or biases for the sake of consistency. Second, from preliminary observations, approximately 30 minutes is needed to examine the average 100-statement program. If the program is found lacking, the grader must determine the nature of the difficulty before deciding how many points to deduct.

The standards should not specify point values associated with infractions; the instructor can do so in light of his objectives for a particular assignment or the course as a whole. Establishing such a table of specific point values for each assignment allows a more objective evaluation of programs than the subjective "gut feeling" approach.

Finally, the graders probably need to be the cream of the crop of available student assistants--they must be able to evaluate problem refinements/designs, to find "bugs" and assess their seriousness, to determine if error cases are handled properly, and to locate redundant code, all of these without running the programs they are evaluations.

4. Training Procedures

What is the best way to educate a group of undergraduate or graduate students to be able to grade programs according to the standards? This semester the North Carolina State University Computer Science Department has initiated a clinic for explaining the standards and giving practice in applying them to student-written programs. The clinic consists of at least three sessions the perspective student grader must attend. Faculty attendance has also been encouraged.

Session 1 discusses the philosophy behind this new program grading method and how it can provide the student with valuable feedback and reinforcement. Session 2 discusses the documentation standards and how they contribute to better programs. To make the standards more concrete, student programs, both good and bad, are examined, and acceptable and unacceptable practices are pinpointed. We discuss why standardization is crucial, what a program should look like and why, why documentation is important, and why line-by-line evaluation is necessary. Session 3 involves a similar examination of programs, this time focusing on problem refinements and coding techniques. While probing the details of the program coding, we take a second look at the documentation. The last session gives practice in grading sample programs with instructor feedback, followed by a final "examination" to rank the group of graders.

Most of the first two sessions are language-independent: program development, documentation, and coding techniques are discussed in general terms applicable to most languages. Several ideas or techniques peculiar to PL/I are demonstrated, but since PL/I is the "departmental language," all the graders are already familiar with the language. Session 3 obviously needs to be much more language-specific, but still, many of the lessons will carry over into other languages. All graders will be "certified" to grade PL/I programs. We are providing additional "session 3's" for other languages (FORTRAN, assembler), so graders can also be certified in those languages.

5. Benefits

Considering all the costs associated with this scheme, how do we expect to benefit? Applying the standards to all sections and courses in the curriculum should provide more consistent, predictable grading, which will decrease students' uncertainty about what a particular professor actually expects. Students will have to understand the programming language better because their programs will be so minutely scrutinized. Misconceptions and bad habits established early in the student's career remain handicaps for a long time, particularly since programming skill is cumulative in nature. Furthermore, although computer sicence and programming are not synonymous, programming competence is a necessary tool for learning computer science.

In later courses such as data structures or numerical methods, students will be able to concentrate on course concepts rather than grappling with an untamed tool. Neither office hours nor lecture time will have to be devoted to explaining the intricacies of the programming language supposedly being employed only as a learning tool.

We also believe that tighter grading standards will tend to "weed out" marginal computer science students before they reach the high level courses where instructors are more reluctant to fail them on their lack of programming ability alone.

As more students become indoctrinated in the standards by our regular courses, we would hope to shorten the grading clinic. Documentation guidelines and coding rules will have been explained and practiced in previous coursework. Training will still be needed in the mechanics of applying the standards in grading, however.

Whether the benefits sought are realized will be greatly dependent upon faculty commitment and co-operation. They will also be affected by the quality of the graders. We feel the NCSU plan has good prospects for success and believe it can serve as a model for future programs designed to make the teaching of programming a more uniform and algorithmic endeavor.