A Tool For Program Grading: The Jacksonville University Scale

R. Wayne Hamm
Kenneth D. Henderson, Jr.
Marilyn L. Repsher
Kathleen M. Timmer

Jacksonville University, Florida

Burgeoning enrollments in computing present problems--nicer problems than those encountered by our colleagues in disciplines now in less demand--but problems nonetheless. Large enrollments cause particular difficulty in those courses which require substantial programming assignments since programming is not a spectator sport. Good programming techniques are seldom learned by reading programs or by watching others program. Rather, such skills are learned by doing.

Professors confronted with large numbers of programs to grade tend to defend themselves in several ways. They may employ a cadre of graders or teaching assistants. They may decrease the number of programming assignments. Or they may be forced to grade so hastily that they seize one or two simplistic criteria often unrelated to their course objectives. Unfortunately, the results are evaluation inconsistencies, a loss of student confidence in grading fairness, and a diminished level of student competence in programming.

Although the computing curricula have moved away from the teaching of programming languages as important in themselves, skill in programming remains both a marketable commodity and a doorway to a thorough understanding of concepts of computer science and of information processing. Good programming courses demand expert teachers and excellent pedagogy. Too often students encounter classes in which plagiarism is rampant, style is ignored, and grading between classes or among graders is inconsistent. This latter problem is exacerbated in many departments by heavy use of adjunct faculty, many of whom do not or cannot confer frequently with regular faculty. Indeed, even the teacher who personally evaluates all students' programs finds difficulty in grading fairly and consistently.

These considerations prompted the computer information systems faculty at Jacksonville University to meet and discuss ways of improving grading consistency within and between courses and of ensuring attention to efficiency and style as well as correctness of output while increasing the speed and ease of evaluation. The product of these deliberations is an instrument we will refer to as the Jacksonville University (J.U.) Scale.

Our solution to the problem evolved as follows: one of the writers of this paper came into the computing field as a result of his interest in artificial languages. A professor of linguistics in the English department, he is also a specialist in the teaching of composition. As he familiarized himself with the planning tools of computer programming, he began to see an exciting potential for pedagogical cross-fertilization between his new and old fields. Concepts such as top-down planning, pseudo-code, and modularization gave him powerful new analogies to present to his composition students. But of greatest interest to us was an idea that passed in the other direction. One day, after hearing several of us complain repeatedly of the burden of program-evaluation, wishing for a better way, he said, "But there is a better way! Try the Diederich Scale."

He explained that Paul Diederich, a specialist in information processing since World War II and a leading expert of the Educational Testing Service (E.T.S.), had developed the most widely respected set of techniques for evaluation of student writing available. The technique is explained in the classic Measuring Growth in English (1). What English professionals respect about

Diederich is the breadth and quality of the research that led to the development of the scale. The rigor of this research has not been paralleled in the development of any competing grading scheme. His procedure was to give three hundred pieces of student writing to sixty professional readers--thirty in academic areas, including English, and thirty in non-academic professions, such as business and law. The readers were asked to sort all the papers into nine piles of graduated merit and--most importantly--to note on each paper some indication of their reasons for a particular classification.

The results of this procedure were many and suggestive, but for our purposes the most striking feature was Diederich's factor analysis of the comments. Using a large number of papers and a wide range of readers, his group of specialists determined statistically what English teachers had previously only guessed at--sometimes shrewdly and accurately, often disastrously off the mark--the nature and weight of the factors that cultured readers use formally or informally when evaluating writing. He found that readers look first at the quality of the ideas and then at the organization. Both of these factors have twice the weight of any other factor, the others being flavor, wording, usage, spelling, punctuation, and manuscript attractiveness and legibility. Diederich observes that "the five factors we found in this particular study are a matter of knowledge, not opinion. We know that these five qualities in student writing influenced the judgments of this particular set of readers, and I use the word know deliberately. These results are far more convincing than any theoretical, armchair analysis of how students ought to write" (1.9).

Diederich arranged these factors in a useful weighted scale, as follows:

Ideas	2	4	6	8	10
Organization	2	4	6	8	10
Flavor	1	2	3	4	5
Wording	1	2	3	4	5
Usage	1	2	3	4	5
Spelling	1	2	3	4	5
Punctuation	1	2	3	4	5
MS Quality	1	2	3	4	5

He instructed his readers at E.T.S--and has encouraged English teachers everywhere --to use these factors and this scale in their evaluation of writing.

The advantage of Diederich's method is threefold. First, it is the result of careful research and of the widest controlled field testing ever given a scoring method. Teachers can have confidence in the method. Second, it demonstrably improves a teacher's consistency, cutting down on the opportunity for idiosyncratic or eccentric grading. Teachers can feel more comfortable about the grades they assign. Third, it drastically reduces the burden of evaluation. The teacher can simply read a paper once, score it by circling appropriate numbers on the scale, and move on (1.3). The teacher need not identify and mark errors on the paper. In fact, it is best if the scores are not totaled as the teacher reads through a set of papers. That way he or she will not know how many high or low scores are emerging. This knowledge can be a subtle but invalid pressure to tighten or loosen one's evaluation. Totalling can be done mechanically or by an aide at a later time.

The similarities between writing a program and writing an English paper would validate the development and use of a similar scale for grading student computer programs. Like an essay, a program is the solution to a communications problem--a double one, in this case: communication with the computer, and communication with another person. As with an English paper, one starts with an outline or flowchart--at any rate, some logical plan--and then implements the logical plan, well or less well, in code. Like writing, programs have qualities of style and individuality. Two programs which generate the same output can yet have differing qualities. Like an English student, the programmer has more than one way to code the logic. And like English teachers, teachers of computing

need the guidance of consistent factors in order to avoid personal prejudice, the quirks produced by the vagaries of one's own background, and to ensure cross-teacher and cross-course consistency. Like English students, computing students have the right to expect that the grade given by one professor will bear some resemblance to the grade given by another on the same project. Above all, computing teachers are like English teachers in their anguish over paper-load management. In fact, the situation may be worse for the computing teacher: programs are more boring than essays; and typically the computing class is larger than the English class.

Despite the absence of discipline-wide research to determine criteria, we reached departmental agreement about the factors we actually were using in practice. We grouped these into a scale that would be used like Diederich's. At J.U. we deal with courses in which programs are written in BASIC, COBOL, FORTRAN, PASCAL, or APL. Programs may represent anything from a student's first experience with computing through a semester project for a senior seminar. Pending wider research to validate criteria, we settled on the following list of seven factors as generally applicable:

Execution of the program
Correctness of the output
Design of the output
Design of the logic
Design of test data
Internal documentation
External documentation

To facilitate the use of this scale, we wrote a program to generate forms to use in grading projects. The numeric portion of the form evolved to the following:

COMPUTING PROJECT 1 (see appendix)
P - POOR; A - ADEQUATE; G - GOOD

P		A		G	
0	7	13	20		
0	7	13	20		
0	4	8	12	16	20
0	4	8	12	16	20

The version of the program we are currently using allows the instructor to receive complete or minimal instructions, specify the weight for each of the seven factors and up to thirteen additional ones, specify the total value of the project, and an identification for the project. After the user enters the necessary information, the computer generates a form.

When we began to use the new method, we were pleased to discover that our program-grading time was markedly decreased. We felt more secure about grading since each paper had been evaluated against the same criteria. We were apprehensive about student acceptance of the new method. To our surprise, we found that students were pleased to have criteria that they could aim to meet; they liked the attempt to improve fairness by ensuring equal grading across the department; they understood that execution of a program is only one factor.

The usefulness of any method of evaluation, however, depends less upon its users' initial response to it than upon its validity, reliability, and effectiveness as a teaching aid. The J.U. Scale lacks (for now) the kind of validation that went into Diederich's famous one. This is a deficiency that we hope partially to overcome with future research which may well result in modification of the scale. The scale itself probably will not bring immediate reliability tc our grading. Three of us, using the scale, graded all papers turned in for a given project. We had hoped to demonstrate that use of the scale would show that the reliability of grading would be marked. Forty-four percent of the scores, however, differed by more than ten percent among the three graders. This is not surprising since Diederich points out that a single person cannot improve reliability of scoring. Reliability can be enhanced only by group work. Indeed, an early indication that this may apply to the grading of computing projects was found in the fact that, after observing the pattern of deviations, one grader was able, in grading subsequent projects, to bring his

scores closer to the group norm. Any further attempt to improve reliability, however, will necessitate the use of fixed criteria such as the J.U. Scale. Thus our scale is a valuable first step toward improving reliability.

We believe that the scale is an effective teaching aid in three ways:

The discipline can benefit from the use of this method of evaluation by gaining a hierarchy of values against which all programming projects can be evaluated. Such a set of principles needs to apply to all programming languages and yet should allow for the differences between languages. Our scale offers those advantages by judging all projects by a specific set of criteria while still allowing for additional criteria as required by each language.

Individual instructors benefit from our method by relying upon predetermined criteria for all projects rather than by determining a different, often highly idiosyncratic, set for each project. The burden is removed from the individual instructors, who often have to defend personal evaluation methods. Now instructors consider the same criteria equally important. We acknowledge that this is, at this time, the weak point in our argument. Not until research like Diederich's is accomplished can we say with certainty, "These are the criteria that the academic and commercial computing world looks for in a program." But even departmental--or, we might hope, regional--acceptance of our criteria would dramatically increase consistency in evaluation. At the same time, some room has been left in our scale for individual differences. Pending more study, our open weighting system allows for personal decisions. The key is that a pre-specified set of criteria must be considered and weighed by an instructor.

Every advance in fairness is an advantage for the student, the third and most important user group. Students greatly benefit from knowing in advance the criteria by which their work will be judged. Students, we have found, will ask educationally valuable questions such as, "Well, what do I have to do to get a 'Good' on my external documentation next time?" The ensuing discussion is likely to be useful to a group wider than the instructor and that single student. Students also profit because the whole project is weighed against individual parts. The grade comes not from the biases of the instructor but from the student's achievement of criteria set by the department and, we hope, the wider discipline. For example, no project that executed, had correct output, good logic, and useful documentation could fail just because some "output-oriented" instructor didn't like the design of the output. This instructor would be forced to see output as one criterion among others and thus would have to assign proper weight to the other criteria

In summary, we think that use of the J.U. Scale is advantageous to all three interests involved. The key advantages are consistency across the curriculum, consistency between instructors, and consistency across individual projects. As we have stressed, further research is needed to establish and weigh criteria more scientifically. But until this work is done and its results are available, we believe that use of the J.U. Scale will make the discipline's evaluation more consistent, will make the work of the instructor easier and faster, and will improve the students' understanding of the evaluation of their projects.

References

1. Paul B. Diederich, Measuring Growth in English (Urbana, Ill.:NCTE, 1974).

Appendix

Sample Run of Scoring Program

```
:RUN SCORE


DO YOU WANT COMPLETE INSTRUCTIONS?YES

THE PROGRAM PROVIDES FOR THE EVALUATING OF UP TO 20 ASPECTS OF A PROGRAM.
  7 ARE SPECIFIED AND 13 MAY BE SPECIFIED BY THE INSTRUCTOR.

FOR EACH ASPECT, THERE IS A CODE AND A WEIGHT.
  IF THE CODE IS 1, THE MAXIMUM NUMBER OF POINTS
    FOR THAT ASPECT IS ASSOCIATED WITH ADEQUATE.
  IF THE CODE IS 2, THE MAXIMUM NUMBER OF POINTS
    IS ASSOCIATED WITH GOOD.
  IF THE CODE IS 0 OR NOTHING IS ENTERED, THE ASPECT IS NOT TO BE SCORED.

YES OR NO QUESTIONS MAY BE ANSWERED "Y" FOR YES.
RESPONSES NOT STARTING WITH "Y" ARE INTERPRETED AS "NO".


ENTER THE PROJECT IDENTIFICATION
APPLICATION DEV. PROJECT 4


FOR EACH ASPECT ENTER THE CODE AND WEIGHT, SEPARATED BY A SPACE.
   A RETURN ELIMINATES GRADING ON THE ASPECT
    EXECUTION OF THE PROGRAM   1 20

    CORRECTNESS OF THE OUTPUT  1 20

    DESIGN OF THE OUTPUT

    DESIGN OF THE LOGIC        2 20

    DESIGN OF TEST DATA

    INTERNAL DOCUMENTATION     1 20

    EXTERNAL DOCUMENTATION     2 10

HOW MANY ASPECTS DO YOU WISH TO SPECIFY?
YOU ARE ALLOWED A MAXIMUM OF 13   1

ENTER THE ASPECT TO BE GRADED
USE OF A TABLE

ENTER THE CODE AND WEIGHT FOR THIS ASPECT   1 10

THE TOTAL OF THE POINTS ENTERED IS     100
THE POINTS CAN BE SCALED IF DESIRED.
WHAT WOULD YOU LIKE THE TOTAL TO BE? 50
```