Use of Peer Ratings in Evaluating Computer Program Quality

Nancy Anderson
Ben Shneiderman
University of Maryland
College Park, Maryland 20742

INTRODUCTION

Peer review techniques can be useful tools for supplementing programmer education, improving cooperation and communication within programming teams, and providing programmer self-evaluation. This paper will explore the benefits of peer review for practicing professional programmers and describe a framework for administration of an annual or semi-annual peer review process.

Professions which require complex cognitive skills where objective evaluation of success is difficult have frequently used ratings or a form of peer review for improving productivity, communication among individuals, and performance self-evaluation. For example, physicians have peer review boards to discuss successful or unsuccessful treatment policies. Clinical psychologists make case presentations to assess their therapeutic plans, learn from their colleagues, and communicate new therapy strategies. The academic research community uses peer review for refereeing papers submitted for publication. A paper is often refereed by two or more selected professionals active in similar research. Criticisms and suggestions for improvement are returned to the author while the editor decides whether the reviews indicate sufficient grounds for publication. The author is given specific guidance for improvement and the referees benefit by being made aware of new research at an early stage. Furthermore, compelling referees to give detailed critiques requires them to struggle through the arguments of the paper and hopefully learn something.

Peer ratings used in private industry and areas of the federal government are often used to educate management and executives about themselves (Educational Testing Service, 1962) and to predict success or failure to complete training programs (Doll and Longo, 1962; as well as Hollander, 1957). Frequently, the majority of individuals who participate in peer rating programs have reported that it was a constructive and nonthreatening experience (Roadman, 1964). Thus, experiences reported in the literature generally support the educational usefulness of peer ratings.

For programmers, the peer review process would provide an opportunity to get beyond the annoyance of debugging today's code and consider more fundamental issues. The kind of peer review that we recommend focuses on programming, not systems analysis and design. We are interested in providing feedback to programmers on the quality of

their product--the programs they wrote. This is necessary because we have poor metrics of program quality. While Gilb (1976), Boehm (1976), Halstead (1977), and others have proposed extensive objective and automatable measures of program code, all of these techniques fail to capture a measure of the "quality" of a program. The execution time, memory space, number of module linkages, depth of nesting, or number of operators or operands are important properties of programs, but these alone may not tell us whether a good algorithm has been selected, whether the code will be easy to debug or modify, whether the output is in a natural and comprehensible form, or whether the modular decomposition was reasonable.

In recent work, Shneiderman (1977) used memorization and recall tasks as metrics of program quality and comprehension. This approach remains appealing, but it does not provide comparative feedback to programmers or necessarily improve communication among programmers.

A peer review process applied to programs has the potential to provide feedback to programmers, to offer an educational experience for both the reviewer and the programmer, and to improve communication among teammates. In general, communication among individuals increases when working together in a team. Reported satisfaction with a rating technique increases when the interactions are similar to typical tasks the individuals are required to perform.

METHODOLOGY

We propose that a practicing programmer in an organization be designated as the administrator of the peer review. By having a peer as the administrator, anxiety or mistrust will be reduced. It must be stressed that the peer review process we propose is not to be used for promotion or salary increases, but is directed at:

1) programmer education
2) improving cooperation and communication in a programming team
3) self-evaluation.

High level management approval must be obtained since the administrator will have to contribute at least one full day and programmer participants will have to contribute a half day each. Managerial support of peer review will ensure the highest level of effort from the participants, but managerial involvement should be prohibited.

Selecting the Participants

The administrator selects six to twenty participants, but the range could be extended. Fewer participants may destroy the anonymity and lead to personal discomfort, anxiety, and ego-threatening disagreements. Too many participants may reduce interest, bring together people of overwhelming diversity, and cause administrative complexities.

The participants should have similar experience, levels of competence, and interest in common problems. If the diversity is too great, the participants will not be able to intelligently review their colleagues' programs. Clearly, FORTRAN scientific programmers, COBOL business programmers, and assembly language systems programmers should not be in the same group. Sackman (1972) and others have shown at least a 28:1 ratio in performance in programming tasks for individuals with the same job description--there is no need to seek out diversity by including individuals with vastly different backgrounds.

The participants should be told about the peer review and its goals and should be given the opportunity to sign up as participants. Coercive measures might alienate potential participants and suggest management pressure or devious goals. The peer review should be advertised as an opportunity for self-improvement. The administrator may or may not be a participant.

Generating the Materials

At least two weeks before the established peer review day, programmers should be asked to select two samples of their programs. The "best" should be an example of what they consider to be their finest work; the "second" should be a program which they themselves perceive as being definitely poorer in quality. We will be interested in whether participants can distinguish between programs of differing quality. It will be reassuring and stimulating for future work if people discover that others are able to distinguish quality from second best. Two copies of these two programs should be turned in at least one week before the peer review day. The programs should not have notes indicating authorship.

An informal survey of a number of large programming sites suggests that there are a wide variety of program development styles. Some site managers felt it would be reasonable to ask programmers to provide a 50 to 200 line PL/I or FORTRAN module of code which was authored by a single programmer. Other site managers felt that a 600 line COBOL module was typical of the work of an individual programmer. At other sites, managers felt it would be difficult for programmers to select modules that they had authored independently: there was a great deal of teamwork or a large fraction of the work was maintenance of programs. At these sites, programmers would be asked to provide modules they had worked on sufficiently to call their own, even if they had not been the original author. A final general category of sites were those that maintained a single, large, typically transaction-oriented on-line system. Even at these sites, managers felt that programmers could choose a module that they felt represented their work, even if others had participated in development or maintenance. It may be a strong assumption, but it seems that full-time professional programmers do have some piece of code which they feel is their own.

Again, every effort should be made to minimize the diversity of the submissions. Programs should all be of about the same size, using the same language, and of similar complexity. As much as possible

the problem domains should be similar. Statistical programmers should not be mixed with compiler writers, even if they are all using the same programming language. The participants should be similar enough so that they are capable of evaluating their colleagues'work. If the differences are great then the participants may not value the feedback they receive and the ratings may not be as reliable as when the raters are in a good position to make judgments of performance (Borman, 1974). Frequently, individuals who do not value the feedback may not use it to learn material which is applicable to their work.

Along with the programs each participant should provide a sheet with five objective questions of the fill-in-the-blank type, ranging from simple to very difficult. The questions should be clear and unambiguous. Typical questions could be of the form:

How many times is a specific statement executed for a given input?
How many times is a specific sub-program invoked?
What is the value of a variable at a specific line?
What is the output for a given input?
Give a one-sentence description of the function of the program.
What is the result of a specific minor alteration?
Trace execution by line numbers for a given input (the expected trace should not exceed 15 lines).
What inputs would be required to cause execution of a specified line?
What inputs would cause a specified variable to attain a specified value?

The questions should be randomly ordered in difficulty during presentation. The results will provide participants with an indication of whether their perceived difficulty matches the actual difficulty.

When the two copies of the two programs and the questions have been collected from each participant the administrator should prepare a distribution schedule. Each participant will review four programs for 30 minutes each. The distribution schedule ensures that each participant sees two "best" and two "second" programs, but never more than one program from any specific participant. Order effects should be minimized by the distribution schedule.

Running the Peer Review

Rooms should be reserved for a three-hour period on the peer review day and sufficient desk space provided for studying programs. It is not necessary to have all participants in the same room; in fact, small groups of 3 to 5 per room are preferred. An informal atmosphere should be maintained, but participants are to work by themselves and not to comment aloud. A relaxed learning and study environment should be encouraged and interruptions are not allowed.

Participants will be asked to spend 30 minutes studying each program, filling out the evaluation form (see sample in Appendix 1), and answering the questions. Timing of the four 30 minute sessions should be done accurately and there should be a 10 minute break at the halfway point. Participants must work on only one program during a 30 minute session and can not go back or forward to other programs even if they have competed their work early. The evaluations and question answering are done anonymously. At the end of the four 30 munute evaluation sessions the participants are to fill out the summary evaluation sheet (see Appendix 2). The main purpose here is for the raters to rank the four programs in quality.

When all the evaluations are completed, the question forms should be returned to the programmers for grading. The administrator keeps the evaluation forms. Twenty points are assigned to each of the five objective questions and partial credit may be given. The graded questions are then returned to the administrator.

Evaluation

The results of the evaluation and the questions should be promptly keyed and analyzed so that final results can be returned to the programmer participants as quickly as possible. The original programs, evaluation sheets, question forms, and printout of results should be given to the participants for them to keep. Copies should not be made and the data file should be destroyed. The peer review process is for the educational benefit of the programmers, not management.

If programmers are told to expect an annual or semi-annual peer review then they may be more motivated to produce quality programs for potential submission. The goal of peer review is to improve the overall quality of code production. This is facilitated by compelling people to read other programmers' codes, critiques of their own codes, and responses to the questions which enable programmers to asses how well others can comprehend their own programs. The peer review process also fosters better interaction since programmers are required to read several pieces of active code from individuals in the programming team. This may expose new programming techniques or foster a better understanding of developments by others in progress.

Each participant will receive a one-page printout for each program including subjective ratings and averages across the entire group of participants. Participants can see how well they did compared to their colleagues and can compare the evaluations of their "best" and "second" programs. Reliability of the "highest quality" and "lowest quality" rankings from the summary evaluation form as well as scores on the objective questions will be presented. Each person will receive an analysis of how well their ratings of other programs compared with other raters of the same programs.

Some of these rating measures may be compared with the automatable measures of programs to provide additional insight into what the quality judgments are based upon.

VARIANTS OF THE PROGRAMMER PEER REVIEW

Any of the parameters of the peer review process that have been described can be altered to suit needs and desires. Some fundamental differences are proposed in this section.

An oral evaluation peer review could be made about each program. This would leave room for discussion and questions. Although the benefits of this approach may be substantial, there is real danger of increased anxiety and ego-destroying confrontations. Open discussions should probably be held only among groups of programmers who have already established trust and respect. An experienced group leader, such as a psychologist or psychiatric social worker, might be included to mediate debates and promote productive ego-less (Weinberg, 1971) discussion. These open discussions might be similar to the structured walk-through technique (IBM, 1973). Walk-throughs are generally applied to design discussions; it is not clear that the complexities of reviewing programming language code are amenable to the same techniques.

A promotion related peer review or rating might be conducted by management. Participation might be required and participants would be informed of the goals of the evaluation. Other criteria such as a person's ability to work with colleagues, willingness to adapt to new problems, dedication, motivation, etcetera, would have to be included if the results were to be good predictors of success.

A team evaluation peer review might be established to compare a group of programmers to industry norms. The program samples would include a set of benchmark programs which had been thoroughly evaluated and tested for reliability. Then a comparison of how well programs from the team did against the benchmarks could be obtained. Alternatively, batches of program samples could be exchanged between two programming shops. This would allow a comparison across the industry and would enable programmers to see techniques used outside of their local environment.

SUMMARY

The sequence of steps in the programmer peer review process can be summarized as:

1) Administrator announces peer review and invites participation--homogeneity of subjects is encouraged.
2) Two programs plus questions collected from participants--homogeneity of programs is encouraged.
3) Subjective evaluations of programs and question answering takes place anonymously in an informal atmosphere.
4) Questions are graded by originators.
5) Computer-generated evaluation of scores is performed and all materials are returned to participants. No records are kept.

The programmer peer review process described in this paper is designed to improve programming skills, build confidence, encourage cooperation, and boost morale among a homogeneous group of programmers. Anonymous evaluations enable participants to make honest evaluations without threat of retribution. As much as possible this process has been designed to benefit the programmer participants, minimize negative side effects, and avoid management interference. A field trial is planned during the coming year.

APPENDIX 1

Sample Evaluation Form Program Number _____
Please make any written comments you wish after each question.

	YES						NO
Were reasonable variable names used?	1	2	3	4	5	6	7
Were sufficient and useful comments provided?	1	2	3	4	5	6	7
Were spaces and blank lines used properly to produce a program with a pleasing format?	1	2	3	4	5	6	7
Was the low level logic of the program comprehensible?	1	2	3	4	5	6	7
Was the high level design (for example, top-down or modular) apparent and reasonable?	1	2	3	4	5	6	7
Was the algorithm a good choice?	1	2	3	4	5	6	7
Was this program easy to comprehend overall?	1	2	3	4	5	6	7
Would it be easy for you to modify this program?	1	2	3	4	5	6	7
Is this program compiler and machine independent?	1	2	3	4	5	6	7
Would you be proud to have written this program?	1	2	3	4	5	6	7
Have you ever seen this program before?	1	2	3	4	5	6	7
Could you have written this program better?	1	2	3	4	5	6	7

How would you improve this program?
(you may indicate answers directly on the program.)

General comments about this program:

APPENDIX 2

Sample Summary Evaluation Sheet

Which program was of the highest quality? _____

Which program was of the lowest quality? _____

Which program was second highest in quality? _____

General comments about the peer review process:

REFERENCES

Boehm, B. W., Brown, J. R., & Lipow, M. Quantitative evaluation of software quality. Proceedings of the 2nd International Conference on Software Engineering, San Francisco, 1976.

Borman, Walter C. The rating of individuals in organizations: An alternate approach. Organizational Behavior and Human Performance, 1974, 12, 105-124.

Doll, Richard E., & Longo, Alexander A. Improving the predictive effectiveness of peer ratings. Personnel Psychology, 1962, 15, 215-220.

Educational Testing Service. The conference on executive study. Identifying management talent. Princeton: Educational Testing Service, 1962.

Gilb, T. Software metrics. Cambridge, Mass.: Winthrop Publishers, Inc., 1976.

Halstead, M. Elements of software science. New York: American-Elsevier, Inc., 1977.

Hollander, E. P. The reliability of peer nominations under various conditions of administration. Journal of Applied Psychology, 1957, 41, 85-90.

IBM. Structured walk-throughs: A project management tool. IBM Data Processing Division, August, 1973.

Roadman, Harry E. The industrial use of peer ratings. Journal of Applied Psychology, 1964, 48, 211-214.

Sackman, H. Man-computer problem solving. Princeton: Auerbach, 1970.

Shneiderman, B. Measuring computer program quality and comprehension. International Journal of Man-Machine Studies, 1977, 9, in press.

Weinberg, G. The psychology of computer programming. New York: Van Nostrand-Reinhold Publishers, 1971.

Zwany, Abram and Arie Y. Lewin. Peer Nominations: A Model, Literature, Critiques and a Paradigm for Research. Personnel Psychology, 1976, 29, 423-447