d) Comparative studies of types of computer facilities such as on-line, off-line, and hybrid installations.
e) Comparative studies of performance effectiveness of broad classes of program languages with respect to representative programming problems, for machine-oriented, procedure-oriented, and problem-oriented languages.
f) Systematic collection, analysis and evaluation of the empirical characteristics, correlates and variation associated with individual performance differences for programmers, including analysis of team effectiveness and team differences.
g) Development of cost effectiveness models for computing facilities, incorporating man and machine elements, with far greater emphasis on empirically validated measures of effectiveness, and less emphasis on abstract and over-simplified models than has been the case in the past.
h) Detailed case histories on the genesis and course of programmer problem-solving, the frequency and nature of human and machine errors in the problem-solving process, the role of realtime machine feedback and reinforcement in programmer behavior, and the delineation of critical programmer decision points in the life-cycle of the design, development, and installation of computer programs.
i) And finally, integration of the above findings into the broader arena of man–computer communication for the general user.

More powerful applied research on programmer performance, including experimental comparisons of on-line and off-line programming, will require the development in depth of basic concepts and procedures for the field as a whole, as roughly indicated by the above broad areas of applied research.

ACKNOWLEDGMENT

We are indebted to R. L. McCornack for consultation and statistical services at SDC, to R. M. Berger at the University of Southern California for the administration of the Basic Programming Knowledge Test, and to many colleagues at SDC who critically reviewed the results or who participated as subjects.

A Critique of "An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions"

BUTLER W. LAMPSON

Abstract—The preceding paper by Grant and Sackman, "An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions" is discussed critically. Primary emphasis is on this paper's failure to consider the meaning of the numbers obtained. An understanding of the nature of an on-line system is necessary for proper interpretation of the observed results for debugging time, and the results for computer time are critically dependent on the idiosyncracies of the system on which the work was done. Lack of attention to these matters cannot be compensated for by any amount of statistical analysis. Furthermore, many of the conclusions drawn and suggestions made are too vague to be useful.

INTRODUCTION

IT IS GENERALLY accepted in the computing field that experimental data on the performance of programmers in various environments are both needed and sadly lacking. It is also generally recognized that such information is difficult to acquire because of the cost of conducting experiments with statistically significant numbers of subjects. What is perhaps less widely appreciated, however, is the fact that the raw data which emerge from programmers' time sheets and computer accounting records must be interpreted with the greatest caution, not only because of the wide variations in programmers' ability and the large number of factors which influence their performance, but also, and more important, because of the enormous differences among computer systems, differences which often depend on rather subtle technical considerations. It is simply not true that data taken in a particular environment, with a particular machine, language, and operating system, have any direct relationship to the results that may be expected in a different environment, even if the adjectives "time-sharing" or "on-line" can be applied to both. To take a very down-to-earth example, a difference of 50 percent

Manuscript received September 19, 1966.
The author is with the Department of Electrical Engineering, University of California, Berkeley, Calif.

in turnaround time in a batch processing system can drastically change the performance of programmers (and their morale), yet differences of this order between two installations using the same equipment are not at all uncommon. The differences among time-sharing systems in the present state of the art are far more significant.

It is the purpose of this paper to point out a number of factors which cast the gravest doubt on the general applicability of the results presented in the paper by Grant and Sackman [1] which is under consideration. These remarks should not be regarded as a condemnation of the experiment which they conducted; any attempt to obtain quantitative data on the performance of programmers or computer systems is laudable in our present state of ignorance about these matters, even if it serves primarily to reveal some of the methodological pitfalls of such investigations. The major point of the succeeding sections is that in these experiments a serious attempt is needed to understand the significance of the property being measured; efforts to determine the relationship between this property and the numbers actually obtained by observation are, although important, necessarily secondary. Attention will be concentrated primarily on the comparison of on-line and off-line conditions. Before we plunge into these matters, however, a few general comments may be in order.

For one unfamiliar with the niceties of statistical analysis it is difficult to view with any feeling other than awe the elaborate edifice which the authors have erected to protect their data from the cutting winds of statistical insignificance. It does seem, however, that there is a basic weakness in an experiment which, finding that one group of programmers takes 60 percent longer than another to code an Algebra problem, is unable to conclude that this difference is not the result of chance. Even with regard to differences which are significant according to the statistical tests being used, such as a ratio of about 2 to 1 for programmer time in favor of on-line debugging, the analysis offers no guarantee that this figure is correct. In other words, it is not possible to regard the results of this experiment as anything but qualitative indications of relative performance. The large standard deviations and small sample size make the actual numbers quite unreliable. The cure for this would seem to lie in the construction of better experiments rather than in a reliance on elaborate analysis to salvage something from bad ones.

On-Line Systems

We turn now to a consideration of the nature of on-line systems. A time-sharing console is a device with enormous potentialities. If they are to be realized, however, carefully designed programs are needed to provide the console user with the services he needs. If these programs are slow, incomplete or cumbersome to use, much of the value of the system is lost. A good deal of nonquantitative evidence has been accumulated in the last few years as to the importance of good interactive software. The JOSS system at RAND [6] and OPS at MAC [7] are only the best-known examples.

The relevance of these facts to the experiment being considered is twofold. First of all, the JTS language [8] and compiler in which almost all of the experimental programs were written is not an ideal on-line tool. To write and run a program under this system requires the following steps.

1) Type the program into the TSS editor [9], a program for creating and modifying text.
2) Call in the JTS compiler to convert the program into the binary form in which it can be run. The entire program must be compiled at the same time, and any change in it requires a complete recompilation. The time required for a program like the Algebraic Interpreter is one or two minutes of computer time and 10 or 15 minutes of console time. If the computer detects any errors, go back to step 1 to fix them and try again.
3) Load the binary into core, where it can be run, examined, and altered *in machine language* using the TSS DBUG [10] commands. The program can be referenced only with octal absolute addresses; the data can be referenced symbolically by the name given it in the source language program. Program changes must be made in machine language, and with octal addresses. Any change to the source language requires editing and recompilation (see step 2 above).

This may be called a separated system. It is to be compared with integrated on-line source-language debugging, which is implemented or planned for most commercial time-sharing systems [2]–[5]. The user types his program into the compiler, which offers all the features for modifying it that the editor provides, but also checks each statement for correctness and generates an error comment if it is in error. When enough has been typed in to make a workable section, he runs it. If errors are made, they are corrected on the spot. The programmer works in source-language at all times and does not have to learn two different forms of representing his program. The time required to change one source line is about 10 seconds, rather than 10 minutes. The price of this power in machine time is considered in the next section.

But the Q-32 implementation even of the separated procedure leaves much to be desired. Because logically distinct parts of the program cannot be compiled separately, both programmer and machine time are wasted in repeated recompilation of debugged sections of code. The TSS editor is altogether lacking in the convenience and power that users of commercial systems will demand. And the debugging system, requiring the user to deal with his program (although not his data) in octal addresses instead of symbolic ones, is cumbersome and inconvenient. The atmosphere of the system is summarized by a remark in the DBUG manual: "Any user who wants more detailed information concerning an error can type SERROR/ and present his teletype copy to TSS personnel." The possi-

bility that the computer might be able to provide informative messages is apparently not considered.

It is not, of course, the fault of the experiment that these deficiencies exist; the reason for discussing them is to clarify the differences between the "on-line" debugging studied by Grant and Sackman and the facilities which are available in true on-line systems, and hence the significance of the numbers arrived at in this study. It is interesting that the Q-32 system reduces debugging time by a factor of about two from the time required in an "off-line" system, but this fact cannot be regarded as more than a suggestion of the performance to be expected from more sophisticated systems.

Computer Time

The results obtained for the expenditure of computer time suffer from much the same problems as the ones for programmer time: the Q-32 system has characteristics entirely different from those of later time-sharing systems now commercially available. This section is devoted to a discussion of some of these differences, although one which, for obvious reasons, has been greatly oversimplified.

The factors contributing to the use of machine time in a time-sharing system are not widely known. Confusion in this area stems from a failure to understand the implications of frequent interaction between the user and the computer and from the complexities of the scheduling and accounting algorithms which can be used. The first point is more important, and consideration of a misleading statement in the paper may serve to clarify it.

Discussing the choice of programming language for the experiment, Grant states that off-line coding in a language like JTS places less demand on a time-sharing system than on-line coding in an interactive language. Although this is probably the case for the Q-32 system, it is by no means true in general. In a properly designed system such as the SDS 940 or GE 645, the cost of having a terminal attached and active is very low as long as the user is not doing anything: perhaps 50 words of core, a couple of thousand on a drum, and one teletype input channel, equipment with a total rental of $50/month in one of the commercial systems now on the market. The teletype itself is worth anywhere from $15 to $60/month. Altogether, an hourly cost of fifty cents is probably about right.

Of course, the cost rises if the user initiates a computation. In a system which is capable of overlapping input/output (including the operation of swapping the user between core memory and the drum on which he resides when not active) with computation, however, the amount of CPU time used by simple but common operations such as editing is rather small, and a considerable number of such operations can be accommodated without greatly affecting the performance of the system. These remarks may appear to be inconsistent with Grant and Sackman's statement that a good deal of machine time was used by the on-line editor. The explanation is that the Q-32 system does not overlap swapping with computation, so that the cost of a small interaction of the kind which is constantly occurring in the editor is artificially inflated.

The cost of allowing the user to enter his program interactively at the console is rather small. On the other hand, the amount of computer time saved by an interactive system can be considerable, because the cost of correcting an error in the source language is much less than in a system like JTS. In the former case only one statement need be recompiled, while in the latter the entire program must be processed. As Grant points out, a large part of the machine time expended by the on-line subjects was used in recompilations. It therefore appears likely that on-line machine time would be significantly reduced by an interactive language; this fact is independent of the advantages of interaction for the programmer, which have already been discussed.

Consideration of computer time may properly be concluded by some mention of its cost relative to programmer time, since the entire issue of on-line versus off-line is primarily a question of economics. Time on systems which are very roughly comparable to the Q-32 TSS can be purchased commercially for about $350/hour, and this figure may be expected to decline significantly in the near future. A reasonable minimum for the cost of an hour of programmer time is $10. On the basis of these figures and the totals in Table I of Grant and Sackman's paper, it is easy to see that programmer debugging time cost more than twice as much as the machine time used for the same purpose. A convincing case can, therefore, be made for the limited application of the maxim: "Don't think, compute." A fairer, though less pithy, statement of it might be: "Let the machine do the dirty work; it's a lot cheaper."

Turnaround Time

The authors present in Section 6.3 a discussion of the effect of changing the off-line turnaround time. It is difficult to know what to say about this discussion: its "exploratory" nature blunts the edge of criticism. There does not, however, seem to be any reason for taking seriously the graph of debugging time against turnaround time presented in their Fig. 2, since the assumption of linearity on which it is based has no foundation in empirical data and a weak one in intuition about programmer behavior.

Furthermore, it is not clear what the difference is between short response time and the "interactive features" of the Q-32 system. As the last two sections have indicated, this system does in fact often look to the user like a low-turnaround off-line system because of the design of much of its software.

In the absence of any attempt to determine the actual effects of varying the turnaround time, the entire section seems to be unnecessary, especially in view of the fact that it ends with the statement that it has suggested "some hypotheses, derived from the observed data, that

might be explored in such studies." A careful qualitative consideration of the factors likely to bear on the effects of changing turnaround time would have been more valuable. The question of the significant differences between 5-minute turnaround and on-line operation is certainly an open one, but Grant and Sackman have not contributed to its resolution.

Conclusion

Perusal of the paper leaves a strong impression that the authors are not in close touch with reality. They have ignored a number of vital factors bearing on the interpretation of their results, some of which have been discussed above. Furthermore, some of the conclusions they draw and many of their suggestions for further work are questionable.

Much is made, for example, of the differences between Algebra and Maze problems. In fact, however, these problems are very similar when viewed against the background of the problems for which many programs are written. There are, for instance, small numerical problems for which a language like JOSS is suited; large numerical problems like matrix inversions; major commercial data processing jobs requiring months of effort; and large system programs. Furthermore, it is observed that good programmers are able to turn their attention from one class of problems to another with relative ease. This fact throws doubt on the "differentiation hypothesis," which is not really supported by the data. The failure of experience to correlate with performance offers no evidence that specialized experience is any more valuable than general. Intuition would suggest that innate ability is the most important factor in performance after a certain amount of familiarity with computers has been obtained, and that *knowledge* of techniques and methods of organization, rather than experience in their use, is next in importance. The idea that programming should be split up into a lot of little compartmented areas is a most unfortunate one.

Another remarkable conclusion is that "it is apparent from the spread of the data that very substantial savings can be effected by successfully detecting and reassigning low performers to other positions." Every supervisor of programmers knows this perfectly well. Unfortunately, its practical application is not so easy because of the extreme shortage of high performers. Programming differs little from other creative work in this regard.

Finally, the broad areas for further research proposed at the end of the paper can be summarized very briefly as a proposal to "study computers and their users." That this is a laudable goal will be disputed by few readers.

References

[1] E. E. Grant and H. Sackman, "An exploratory investigation of programmer performance under on-line and off-line conditions," *IEEE Trans. on Human Factors in Electronics*, this issue, page 33.

[2] W. W. Lichtenberger and M. W. Pirtle, "A facility for experimentation in man–machine interaction," *Proc. 1965 Fall AFIPS Joint Computer Conf.*, pt. 1, vol. 27, pp. 589–598.

[3] B. W. Lampson, W. W. Lichtenberger, and M. W. Pirtle, "A user machine in a time-sharing system," *Proc. IEEE*, vol. 54, pp. 1766–1774, December 1966.

[4] W. T. Comfort, "A computing system design for user service," *Proc. 1965 Fall AFIPS Joint Computer Conf.*, pt. 1, vol. 27, pp. 619–625.

[5] F. J. Corbato and V. A. Vyssotsky, "Introduction and overview of the Multics system," *Proc. 1965 Fall AFIPS Joint Computer Conf.*, pt. 1, vol. 27, pp. 185–196.

[6] J. C. Shaw, "JOSS: A designer's view of an experimental on-line computing system," *Proc. 1965 Fall AFIPS Joint Computer Conf.*, pt. 1, vol. 27, pp. 455–464.

[7] M. Greenberger and M. Jones, "On-line simulation in the OPS system," *Proc. 21st National Conf. ACM*, pp. 131–138, 1966.

[8] Phyllis R. Kennedy, "JTS user's guide," System Development Corporation, Santa Monica, Calif., SDC TM-1577/002/00, August, 1965.

[9] S. M. Aranda, "Q-32 time-sharing system user's guide, executive service: symbolic file maintenance (EDIT)," System Development Corporation, Santa Monica, Calif., SDC-TM-2708/202/00. February, 1966.

[10] S. M. Aranda, "Q-32 time-sharing system user's guide, executive service: context editing (EDTXT)," System Development Corporation, Santa Monica, Calif., SDC-TM-2708/204/00, March 1966.

[11] R. R. Linde, "Q-32 time-sharing system user's guide, executive service: debugging (DBUG)," System Development Corporation, Santa Monica, Calif., SDC-TM-2708/390/00, April 1966.