HUMAN PERFORMANCE EVALUATION IN THE USE OF FEDERAL COMPUTER SYSTEMS: RECOMMENDATIONS

Mark A. Underwood*

Navy Personnel Research and Development Center
Code P204 - Bldg. 330
San Diego, CA 92152

There has been increased awareness in recent years of the high cost of non-hardware items in the Federal ADP budget in contrast with decreasing costs for much of the hardware. More attention is being given to software development costs, systems design practices, automatic program testing, and the like. Particular commercial and military systems effectiveness and life cycle costs now take into consideration such factors as part of the planning process. It is suggested that not enough attention has been given to measurement of human performance variables as part of the systems procurement and systems evaluation phases of Federal ADP programs. Recommendations are made for the incorporation of such measures along with conventional hardware/software performance measurement.

Key words: Computer performance; federal systems evaluations; human performance measurements; psychology of computer systems usage.

1. Introduction

Proliferation of low-cost computer systems in the Federal Government may well reduce or hold the line on overall computer hardware costs. This turn of events will draw attention to a steady increase in personnel costs associated with the operation, maintenance, programming, and use of computer systems. Accordingly, there has been some research in the field of the cognitive and human factors involved in programming. Some of the outgrowths of this research have been "software science" (Halstead), "software physics" (Kolence) and the field of structured design. The popularization of structured programming was spurred, seemingly, by a realization that the sophistication of machines being manufactured exceeded human capabilities to utilize them efficiently.

The Federal Government, some have maintained, has been a stimulus for growth in the area of computer performance evaluation (CPE) because of the role that CPE can play in improving utilization of existing computer systems, and the role it can play in Federal acquisition actions. Most of the effort which has been publicly documented has dealt with performance evaluation of hardware configurations, and with software as a system interacting with a collection of hardware resources. Human resources played an indirect role in such measurements: they produced the software, or they were consumers of the hardware-software configuration (generated n transactions per minute, etc.). As a recent review article indicated [1][1], even the study of

*The opinions and assertations contained herein are those of the writer and are not to be construed as official or reflecting the views of the Navy Department.

[1]Figures in brackets illustrate the literature references at the end of this paper.

software science is limited to measures which can be computed automatically from a computer program (e.g., during compilation), which by its very nature limits the human factors which can be measured. This emphasis upon "programmer behavior" as opposed to "user behavior" is a prevalent distinction--software development rather than software usage costs receive most of the attention. However, as the user community broadens in scope and size, and as dependence upon systems software, documentation and maintenance aids grows, it can be expected that a more elastic notion of measuring the effective utility of a computer system (or a network of them) must be adopted. To carry this argument to its extreme, one can imagine a computer system whose hardware and software have been well-designed, well-implemented, are highly complementary in terms of their net measurable hardware utilization, yet which requires enormous cost to use effectively because of deficiencies in the user-machine interface. In short, a more comprehensive notion of a computer system as a utility must be used: to include space, power [2], maintainability, training for users, programmers and maintenance technicians, organizational impact, and cost for documentation of programs, systems software, and electronic and mechanical items.

The emphasis of this discussion is to broaden the notion of computer system performance measurement. This can be accomplished by drawing attention to differences between off-the-shelf software products which directly affect user productivity, and by discussing some obvious enhancements to systems which can increase user productivity. However, the primary intention is to stimulate the use of at least some human performance measures in the competitive procurement of computer hardware and software in the Federal Government. Such measures would be part of the more general use of performance measurement technology in procurement actions. This is not to imply that human performance measures are less important in improving the use of existing systems, but it reflects the belief that change can be most readily and dramatically effected at the initial stages of systems development.

2. Some Human Factors in Computer Systems Use

Even a seemingly superficial cataloging of human factors in the use of computer systems reveals a general lack of consideration given to them in the systems acquisitions process. An attempt will be made to explain the importance of these oft-ignored factors--which do not have glaringly obvious price tags attached to them. Minimal attention will be given to the most-researched factors--i.e., factors relating to programmer performance in the use of computer systems. For such discussions, the reader is referred to [3].

2.1 Error Reporting

The most common experience, perhaps, for computer users of all skill levels, is having the system fail to perform a requested function, or to perform a different, unwanted function instead. A facetious observer might suggest that error messages were designed as though error detection and correction was to be a riddle to be solved by the user. Error reporting, whether system-caused, or user-caused, is an unwanted system perturbation which results in reduced efficiency at the human-machine interface. In the worst cases, the user is thrown back to a shelf of loose leaf notebooks or put into the expert's consultation queue before she can proceed with the work at hand. An example of this is the cryptic, "FAC REJECTED 400100001", from which the user is to infer that a file could not be found in system directories. Error reporting can also be excessive in its detail--users who are familiar with a system must put up with lengthy error diagnostics for errors perhaps caused by typo's. Nor should an attempt be made to "reach a happy medium"; there is no happy medium in the typical heterogeneous user environment. Multiple levels of detail should be available upon request. References to specific pages of written materials should be made at some level of depth if necessary. At the highest level, only the most necessary and self-explanatory information should be given. Various levels of error reporting could be turned off at the beginning of sessions, or turned off and on by the user at will.

The relationship between user actions and system reactions can be obscure. The user will frequently find herself saying, "now what does that message really mean?". And how frustrating to receive the consultant's answer, "Well, you left off a period on your file name; therefore the diagnostic has no meaning." Little wonder that new users remain bewildered as to how finite-state machines can appear to behave so illogically. There are instances of true ambiguity, of course, but the point is that there are far more than there need to be.

2.2 Systems Software Documentation

Most of the sources of problems with error reporting can be traced to systems software documentation practices. The purpose here is not to deal specifically with

systems software design, but rather with its effect upon users. The extremely dynamic nature of systems software (changing several times a year in version if not more often) requires close monitoring of the relationship between new versions of software with new versions of documentation on that software. Because of the intrinsic interdependence of much software, simple replacement of pages here and there is not enough. The format of most systems documentation is also subject to criticism because of a clumsy organization which is based upon the internal software structure instead of the user perception(s) of system functions. Indexes are rarely adequate or complete, and are especially weak in terms of cross-references to related topics. Some ingenuity must be spent in the development of documentation benchmarks; e.g., take a naive user and ask her to perform a particular function "cold", with only the documentation available. Measurement of this time-to-criterion would obtain some revealing information about the design of the documentation. The documentation which would be measured should be based upon a representative job mix of the type of user and programmer behavior which is expected on the system. Normally, a broad spectrum of users would be required for a general-purpose system (general-purpose systems seems to get the most scrutiny).

Some industry standardization, perhaps using CODASYL as a prototype, regarding documentation would be highly desirable. The programmer or user who is required to move from system to system performing various functions in terms of access to data or software development or systems design is hard pressed to keep straight both differences and common capabilities. Transfer to training is most negative, unfortunately. Notably lacking are between-manual references (e.g., between the manuals for a compiler and operating system manuals), and annotated runstreams. The idea of "learning from example" is especially helpful in improving productivity in the use of a system; there is much to be said for the idea of getting the job done using a model, and then figuring out the conventions and the protocol later. Annotated runstreams appear in some training manuals, but for some reason, the managers of documentation staffs seem to be suffering from the delusion that reference manuals are novels or textbooks to be read chapter by chapter in a predetermined order, and examples play a secondary role in such exposition. Like the error reporting functions, documentation must be available at various depths of detail and theory. When one is looking for the fast answer to an acute problem, that is no time to discover, for instance, the theory behind the generalized syntax analyzer for the operating system's command processor.

Along these lines, there should be more effort to put together sections of documentation that are functionally related--not just theoretically or alphabetically related. For instance, when a cryptic message tells a user that she has used up all the disk space she allocated for a file, she wants to know what she can do about it, not all the theory behind how the granules, cylinders, tracks, sectors were estimated in the original allocation. Some of these interrelationships can be discerned from the error messages which the system issues, others from the common sequences of commands seen in annotated runstreams. Another example is the sequence of EDIT-COMPILE-LINK EDIT-EXECUTE-READ DATA-REPEAT. The reader of the EDITOR manual is not told how to compile a file created by the EDITOR. The reader of the compiler manual may not be told how to create a relocatable program. The reader of the LINKEDITOR manual may not be told how to execute a program which has been loaded, or how to associate files with the program that has just been created. A common structure for editor documentation would be a tremendous improvement.

2.3 Data Communications Facilities

The Teletype Model 33 user who is taken into the computer room and sees the systems console operating at 19.2 kbs may be prompted to say, astonished: "Hey, the computer is fast, after all!" For some of the functions involved in computer system use, higher baud rates (especially above 2400 bps) are highly desirable. While there may be some argument as to whether higher baud rates result in less "think time" for users and hence result in inefficient system use, most anyone would agree that, all other things being equal, higher baud rates are desirable.

Yet the matter of data communications design is not given a central role in systems design and implementation of a computer site. Perhaps much of the trend toward decentralization of computer facilities, toward smaller machines, is caused as much by the inability to get hard-wired high speed data communications service to larger, host computers as any other single reason. One might argue that to not have fast baud rates is to be cheated of the speed of the computer system, since the result that might have been computed in microseconds takes many seconds or minutes to be displayed on the screen. Similarly, adept, typewriter-wise users are frustrated by the inability of the computer to keep up with their nimble fingers darting to

the "RETURN" or "TRANSMIT" key. Insufficient thought has been given to whether dedicated, conventional TTY (seemingly the most common based on CRT sales) or poll-and-select protocol is the most desirable, and what the trade-offs might be for a particular mix of users. Another example is the need to be able to utilize the editing features of the terminals available on the market today, and thereby reduce the editing load on the host system; i.e., line-by-line vs. screen-by-screen terminal-host transactions.

User needs and requirements, not terminal manufacturer or host default protocols should determine the type of data communications service to be used for any given case. Planning for diversity in the data communications environment of a large site seems to be the exception rather than the rule. There is an unfortunate temptation to standardize upon a protocol or two and then demand that the community of users conform to this arbitrarily determined standard.

2.4 Software Development Aids

There is scarcely a manufacturer who does not claim to have exceptional program development tools--debuggers, dump analyzers, subscript checkers, etc. These tools, and other software amenities, such as user program libraries (e.g., Programmer's Workbench under UNIX) become a necessity for large-scale software development projects. However, standardization of the functions or documentation of these programs has not occured, and therefore inadequate attention may be given to identifying what the necessary features of these programs may be for competitively selected systems. A careful examination of the system user mix and the past behavior of software development personnel would be the most reliable sources of information regarding such requirements.

In general, however, not a great deal of thought has gone into the development of such amenities, unless they were deemed necessary in the development of the operating system and the compilers (and then subsequently released to the public). For example, in spite of the encouragement of structured programming constructs, the user must keep track of "procedures" and individual routines within a physical "file", due to the rigid relationship between files and the utilities (such as the compiler itself) which must operate upon a single file--not a collection of files containing many procedures. Documentation for programs such as this, expect so far as they are self-documenting (and such a thing does not yet exist in a complete sense) must be developed and maintained totally external to the code itself, or else must comment within/among the code. Typical of the lack of understanding of the relationship between the process of program development and the operation of systems programs, is the term "pretty-print". This term implies that "pretty-printing", which refers to the indentation (and other "display" functions) of sections of code which are embedded within outer control loops, is a nicety, an amenity, rather than a necessity without which inefficient system use is inevitable.

The advent of protected fields, partitioned-screen editing, highlighting, blinking, etc., provides the hardware tools for implementing the capability to document programs while writing them, and subsequently, for documenting the use of these programs for their ultimate consumers, (at different levels of detail). As previously stated, the purpose here is not to discuss research into possible documentation standards, but to establish their relevance to human performance in the use of a computer system.

Other aids which may be considered as essential are structured programming "preprocessors" or compiler enhancements (e.g., Harris Corp.'s and UNIVAC's additions to FORTRAN) to support structured programming, and indirectly, structured design (which is even more critical). Some monitoring of the user's own behavior in system use--using many of the same software/hardware tools that are the stock in trade of the computer performance evaluator--would also provide much-needed feedback to users regarding resource utilization, optimization of code, and the conformance with design objectives.

2.5 Systems Software

The systems software, as commonality of parts become increasingly prevalent through the inroads made by OEM manufacturers, and as systems costs go down somewhat, will become the most important part of any computer system. Therefore, the human factors component of the systems software is the most important single aspect of the computer system. To fully discuss the human factors aspects of systems software would require more time and space than can be allotted here. Some areas indicative of the needs can be pointed out, however:

a. Common utility of systems software among all the compilers

b. Accessibility of systems utilities at the CALL level from all higher-level languages

c. Ability to make best use of hardware available (e.g., multileaved memory, cache memory, character or string manipulation hardware, etc.)

d. Software which tells the user what is going on (transparent to the user) if there is a need to know

e. Software's handling of contingencies, and user's control over how the contingencies are to be handled in different program environments

f. Simplicity of the job control language (as evidenced, e.g., by the ability of a user to "guess" which command performs a given function)

g. Generality of the JCL, i.e., the capability of the software to handle a variety of conditions, such as many file structures with the same processing logic while still remaining within a consistent framework of command structures and syntax

h. Language of the systems commands: how well they relate to their functions (do they actually do what they sound like they should do), how easily can they be remembered, how easily can they be associated with other commands which comprise part of a common logical sequence

i. System behavior when it aborts and recovers

j. "System memory" for events--e.g., remembering what the user has done, recovery of previous versions of files or programs, "undoing" changes to edited files, etc.

k. Capability to quickly create and intelligently examine "printouts" and data files on disk without waiting for listings to be printed, as part of test and evaluation phase of software development

l. Capabilities for applications-oriented software (commonly a need for transactional-oriented capabilities, for example)

m. Performance monitoring capability available to the user to aid in software development or in making intelligent use of systems or applications software (applicable to naive as well as sophisticated users if properly implemented)

n. Manipulation of "pieces" of data and programs rather than just entire physical files

o. Study of how long it takes to type in commands to get work done

p. Capability of creating files compatible with other machines

2.6 Accounting System

In the past, accounting for computer system utilization was regarded largely as a necessary evil. Today, in the "era of limits", accounting is more than a necessary function--it is a matter of survival to be accountable. Accountability and performance evaluation, fortunately, go hand in hand. However, particularly on smaller systems, and particularly when it comes to disk storage accounting, many computer systems are deficient in the accounting system which is provided. The accounting system needs to be well enough integrated into the system itself to: provide on-line, up-to-date accounting for usage within a timeframe of a given shift (less than 8 hours and preferably more often), and to account for usage across all resource dimensions by which systems and user software is utilized to do the computing. The oft-stated maxim that the most frequently used software should get the most attention in terms of optimization and documentation is a fine utterance, but this cannot be identified without the proper accounting software.

More importantly than this, however, is the fact that the accounting system for the computer utility in an organization is the most sensitive interface between the organization and the system. The accounting system can be the greatest source of annoyance to managers to contend with, the most distasteful software for the systems programmer to write or modify, the most dynamic data base on the system, an area sensitive to system crashes, and the most important in providing feedback to the user regarding efficiency of system use.

Ironically, it is this interface between the computer system and the organization which provides the greatest opportunity for performance evaluation in general. It represents the greatest source of control of user behavior on the system, while measuring system performance in terms of the organization's objectives--i.e., project-by-project accounting. Through the application of actual costs and weights in the cost algorithms, based upon true costs of the computer utility in the organization, one would expect a changing algorithm as capital equipment costs were recovered and ongoing communications, personnel, space, supplies, maintenance and power costs comprised an increased proportion of operating costs. However, because of the frequently very indirect relationship between actual

systems support/use costs in the organization and the systems billing algorithm, the accounting system becomes a static entity, unresponsible to the organization and unresponsive to the recommendations derived from data which would be collected by systems performance analysts. Convention seems to dictate which systems parameters are assessed and which are not. The sensitive interrelationship between connect time, disk and tape storage, and channel utilization is disregarded in favor of facile algorithms which do not take into account the sequence of events which determine user behavior, the sequence of motions through which users go to accomplish typical tasks.

The general criticism may be made that the parameters utilized in most accounting systems are based upon the hardware costs/resources, rather than including software and labor costs. This is the reverse of the cost trends in computing.

2.7 Customization Capabilities

Programmers who design software systems whose primary consumers are naive users have special requirements of the systems software. In particular, they must present a hospitable facade to the user. The capability should exist which would permit "respectable" recovery from unexpected contingencies, which shields the naive user from distracting or meaningless system messages and acknowledgement, and which in particular permits the building of sophisticated command files or "runstreams". For some systems, it is desirable for the programmer to request accounting or performance information from the operating system at various stages of performing user-requested functions, in order to inform the user about potential or accumulated costs, or to assess the efficiency of the program's own techniques for processing. A typical application requiring such service is an on-line DBMS environment, with a generalized user language presenting the primary user-system interface. Another need arises from the desire to impose a consistency and a "sensibility" upon command structures and interactive dialog sequences which may not be present in the operating system's design. Once again, the structure of an operating system's interface to the usage consists of a myriad of undifferentiated elements which hang together only when viewed from the systems programmer's perspective upon them. Hence, only the capability for executing user programs to issue commands in the normal operating system format(s) can overcome some shortcomings.

2.8 Maintenance

Maintenance is a topic that also deserves a paper unto itself. The dearth of maintenance and maintainability performance evaluation at previous CPEUG conferences is reflective of the general attitude of the industry toward maintenance. Maintenance is viewed as a necessary evil. Technological developments (LSI) have been depended upon to achieve what common sense and a considerable body of literature on the subject has not achieved [4]. Some of the more neglected aspects of maintenance performance evaluation will be treated here; certainly a more exhaustive treatment is warranted.

a. Maintenance accountability. The process by which maintenance is undertaken remains largely understood except within certain academic and military applications circles. Federal contracts call for accounting for hours and parts involved in maintenance of Federal equipment, with penalties sometimes associated with not meeting minimum "up-time" requirements, but this is obviously a superficial treatment of a difficult subject. It would be a pleasant alternative, from the Government's standpoint, if all maintenance problems could be left to the vendors for solution, but this has not happened, and the Government's lead, with its huge investment in ADPE and enormous recurring costs for software and equipment maintenance, would be a much-needed stimulus.

One solution is to make more attractive the preferences given in procurement actions and contract negotiations to more maintainable equipment. At present, it would be difficult to establish that vendors in general even differ significantly from their maintenance records. Everyone seems to accept that disk and tape units require the most maintenance, followed by unit record and printing equipment, and the adoption of industry standard interfaces in some areas and the common use of components by the large OEM suppliers (Pertec, CDC, Calcomp, EMM, Memorex, Wangco, etc.) may have contributed to a homogeneous maintenance environment. However, even these hypotheses may be questioned, since access to data on software reliability and maintenance practices across vendors is also not readily forthcoming, and some Government initiative in that area as well seems called for.

b. Public knowledge of maintenance needs. The notion of maintenance as a behind-the-scenes necessary evil is a negative factor so far as user's understanding of potential problem areas and causes for failure is concerned. Some knowledge about the causes of failures and areas needing greatest

maintenance could be mutually beneficial for users and maintenance technicians in reducing usage in troublesome areas where possible, and increasing sensitivity to intermittent failures which may be more difficult to track down. Clues to subtle systems problems may be shrugged off by users unaware of the situation.

c. Cost-effectiveness assessment of maintenance practice. The Government (or any other large-scale user for that matter) is faced with the troublesome problem of deciding whether on-call, dedicated, or redundant-component maintenance practices are required for a particular installation. The costs of these various alternatives can vary greatly, given the maintenance history of the system, the current cost of labor, and the cost of the most-likely-to-fail or most-catastrophic-if-failed components. Some attendance to the details of maintenance practices and costs could result in more informed decision-making about the most appropriate plan for providing repairs and replacements.

d. Logging of component histories. As board-level replacement and repair at the factory by unknown remote technicians and third party maintenance becomes more prevalent due to changing technology and design practices, automation of logging of component periodic maintenance and failure histories becomes essential. Manually kept logs are inadequate because automatic and timely component statistics are not available.

e. Automation of diagnosis and treatment procedures. Some of the procedures for diagnosis and treatment of computer system (hardware and software) problems are commonly known by better technicians, but even the best technician can easily overlook a logical possibility for system fault due to the complexity of the problem. Some automation of the diagnosis and treatment as well as intervening test and evaluation processes could be instigated even more than has been begun by some of the manufacturers such as DEC and BTI. For further research on this, see [5].

f. Life cycle costs for software/hardware maintenance. Increased attention in recent years to life cycle costs in ADPE have concentrated upon software maintenance costs more than hardware maintenance costs, and a more balanced perspective is needed. Also, the reduced cost of equipment causes the replacement with-new-machine option to become more attractive earlier in the life cycle. Then the problem is one of continuity between short-lived hardware systems. Five years is not a very long time for a project to exist, and for a considerable library of software to be accumulated, yet the hardware may be obsoleted and not supportable at the end of such a time. Such considerations affect the type of maintenance contracts or in-house capabilities the Government must provide.

g. Source code for mini and microcomputers. A deplorable practice exists in some ADP procurements-- those in which the Government does not receive the source code for systems routines. Certain critical routines are necessary for the Government's avowed multi-vendor system program. E.g., to interface some intelligent terminals with page printers of different manufacture, explicit knowledge and sometimes modification of the firmware controlling the serial printer interface are required. The increased use of firmware in computer systems, coupled with the longevity of much systems software which will be in use in the 1980's implies that greater attention be given to the matter.

h. Levels of detail for technicians. Some of the faults of systems documentation which were previously discussed hold true for the hardware documentation for computer systems. The effect is devastating when the technician is forced to muddle through schematics and theory of operation in order to search for the one small piece of information which she needs for troubleshooting. In order to solve this, some companies (such as Dustin Associates) have developed systems encompassing multiple levels of detail--so that all the same information remains available, but only at the relevant point in time in the diagnosis treatment or training process. Review of maintenance documentation in procurement, with "benchmarking" is required, with adoption of more stringent MILSTD regulations for major commercial procurements perhaps advisable. It is obvious that the vendors cannot be left to their own instincts on this matter, as the author's experience with 11 different manufacturers of terminals can attest. (Yet those writing procurement specifications may lack the technician's background needed to include such requirements.)

i. Accessibility and mechanical design factors. Despite some attention given to human factors in electronics systems in the human factors literature, there are unfortunate conditions which many technicians know exist. Components cannot be tested in place, and card extenders may not exist for them. Subassemblies may be mounted in backwards or upside down so that even cursory examination

of cards cannot be performed without lengthly dismantling. A maze of interconnecting cabling may obscure components which must be troubleshooted. Inadequate cooling or mechanical factors may make working on some devices risky in terms of the damage which might accidently be done to the equipment. When parts fail and must be serviced, such factors should be brought to the attention of the contract monitor and/or supervisor for inclusion into maintenance documentation for the system component. Many of the factors discussed in a joint services study [6] have simply been ignored.

j. Federal Government-wide collection of failure data by component. An enormous untapped data base on MTBF and MTTR for computer components and sub-systems exists in the Government, which, if it were pooled, would be a tremendous asset to individuals responsible for planning new systems and evaluating potential equipment by vendors. This data base currently does not exist, nor is there a mechanism to facilitate or encourage the centralization of such information (except for a few large-scale military systems). This might be true, for instance, in the case of PDP 11's even though the number of PDP 11 systems in the Federal inventory may be in the thousands. Sharing of maintenance expertise within the Government, especially on off-the-shelf commercial equipment is also not facilitated though badly needed, especially in the design of distributed computing networks involving multiple systems of the same manufacture or configuration of components.

Related to this is the problem of nonstandard nomenclatures for IC's and various electronic components, making it extremely difficult for Government technicians to cross-reference vendor-specific part numbers with industry-standard parts which the Government might stock in its own inventory. One spin-off of such a centralization of functions would be a Federal stocking system which would reflect the evidenced failures and the demand for spares. This demand currently is not visible because of the high percentage of contract maintenance, and the practice of obtaining parts and services from the immediate vendor rather than (in some cases) even the first manufacturer of the component who supplied the part(s) to the vendor.

k. Standardized procurement of maintenance aids & routines. Procurements of lesser dollar volume which are not part of larger systems typically do not include procurement of the necessary card extenders, spares, exercisers, and special alignment or test equipment which is associated with the equipment. The availability of such items to the vendor alone makes it impossible for the Government to work on the equipment itself--either to effect improvement, modifications or to permit interfacing with other equipment, and it makes it difficult to handle transitions from one maintenance contractor to another--should the original vendor no longer be awarded the contract for maintenance of the equipment. Commonplace delays in the Federal ordering process can be caused by waiting for maintenance aids, parts or technical data from the manufacturer, even though the items in question might have a nominal cost. Mandatory procurement of such items required for maintenance seems more than just a good idea.

2.9 Training Effectiveness

The matter of the training and educational materials made available to the user is one of considerable concern as turnover of personnel and increased travel and labor costs make training a major Government expense. To some extent, training materials should overlap with actual system documentation, but this appears not to be the case for reasons already discussed in connection with systems software documentation. Management of the computer utility includes certainly, some control over the training of new staff of new users, especially as the user population expands to include a greater number and diversity of people. Training effectiveness and its measurement is a speciality of my agency for the Navy, and therefore I know enough to say that this, too, is a subject deserving of separate study. But several areas of deficiency are blatant enough to justify some superficial criticism.

Until recently, many of the training classes held by the major manufacturers did not even include hands-on or even terminal-oriented exposure to systems. Rather, the emphasis has been upon theory of operation, with concentration upon the language and convention of the manufacturers' product line, rather than using some industry-wide language or concepts to explain system-specific notions. There has been little use of individualized instruction or even much involvement of professional educators or educational psychologists in the design of training programs or curricula for commercial computer systems. Certainly the presence or absence of such factors should make a difference in the acquisition process of evaluating vendor proposals.

Unfortunately, training is viewed as a one-time cost associated with initial system procurement, rather than an ongoing require-

ment which as much as any other single factor may determine system efficiency--poorly trained users presumably make inefficient use of computer and, indirectly, organizational resources. Training must keep abreast of new releases of operating systems, compilers and utilities, and the like. In other words, training, to be effective, must be contemporaneous with the system's state of the art, and must be available on an individualized basis. In the computer industry at least, the opportunity seems to exist, and also the economic incentive, (because of the system investment already made in training materiel support) to develop computer-assisted or computer-managed instructional modules to provide various levels of training assistance to the customer. Such features should become ranking factors in procurement decisions.

3. Incorporation of specifications into RFP's and technical evaluations

In order to implement some of these recommendations, it will be necessary to make some changes in existing procurement practices, especially as they relate to "benchmarking", "life cycle costing", and characterization of the "typical" job mix composition. This is a practicable suggestion which can be implemented, on a small scale at least, at the level of the technical individual responsible for writing specifications, and the contracting officer responsible for ultimately consummating a contract. However, for Federal Supply Schedule 74, 66 and 70 contracts, considerable negotiation will have already taken place; the incorporation of factors such as these insofar as they relate to ongoing support and provision of technical and maintenance information, must be done at that time and at that level, by GSA.

Encouragement of the inclusion of specifications such as those mentioned here can be facilitated by explicit mention of them in the various Federal regulations concerning the justification required for ADPE and data communications acquisition. More critical than this, however, for computer performance analysts that are the prime audience for this discussion, is the systematic inclusion of these variables in the assessment of computer system performance. This encompasses a theoretical as well as a practical domain in which user behavior in response to system hardware and software features is part of a feedback loop of events which ultimately, taken together, determine systems performance. Recommendations based upon isolated analysis of hardware or software alone will not necessarily result in increased systems efficiency if human factors in the use of computer systems are disregarded. This important area can also generate a host of stimulating new needs for systems software, documentation, hardware characteristics, and maintainability of systems which in a vacuum of systems modeling and simulation without user variables might never be considered. And, perhaps more importantly, the newly acquired broad population of computer users may be expected to become impatient with computer systems which were designed to be used most effectively by an imaginary, mid-level programmer with a clear understanding of how the system works; this is especially true since the system only begrudgingly yields clues as to how it can be used with the least amount of effort. Professionals in the computing industry may be forced to face the fact that the user view of computer systems as seemingly stubborn puzzles is a definite liability and an embarassment rather than an insider's source of amusement.

References

[1] Fitzsimmons, A. and Love, T., A Review and Evaluation of Software Science, ACM Computing Surveys, Vol. 10, No. 1, March 1978, pp. 3-18.

[2] Held, G., The Hidden Cost: Power, Data Communications, May 1978, pp. 41-52.

[3] Love, T., An Experimental Investigation of Program Structure on Program Understanding, in Wortman, D.B., ed. Proc. of ACM Conf. on Design for Reliable Software, Raleigh, N.C., March 28-30, 1977.

[4] Van Cott, H.P. & Kinkade, R.G., Eds., Human Engineering Guide To Equipment Design, Washington, D.C.: U.S. Govt. Printing Office, 1972.

[5] Electronics Test, debut issue, July/Aug 1978 (1050 Commonwealth Ave., Boston, MA 02215).

[6] Joint Services Symposium in reference 4.