Lawrence Berkeley National Laboratory
LBL Publications

Title
Cautionary Aphorisms for User-Oriented Computer Management

Permalink
https://escholarship.org/uc/item/8dd1v41q

Author
Stevens, D F

Publication Date
1983-04-01

eScholarship.org Powered by the California Digital Library
University of California

LBL-9538 Rev.
Preprint c.2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

RECEIVED
LAWRENCE
BERKELEY LABORATORY

JUN 8 1983

LIBRARY AND
DOCUMENTS SECTION

Engineering & Technical
Services Division

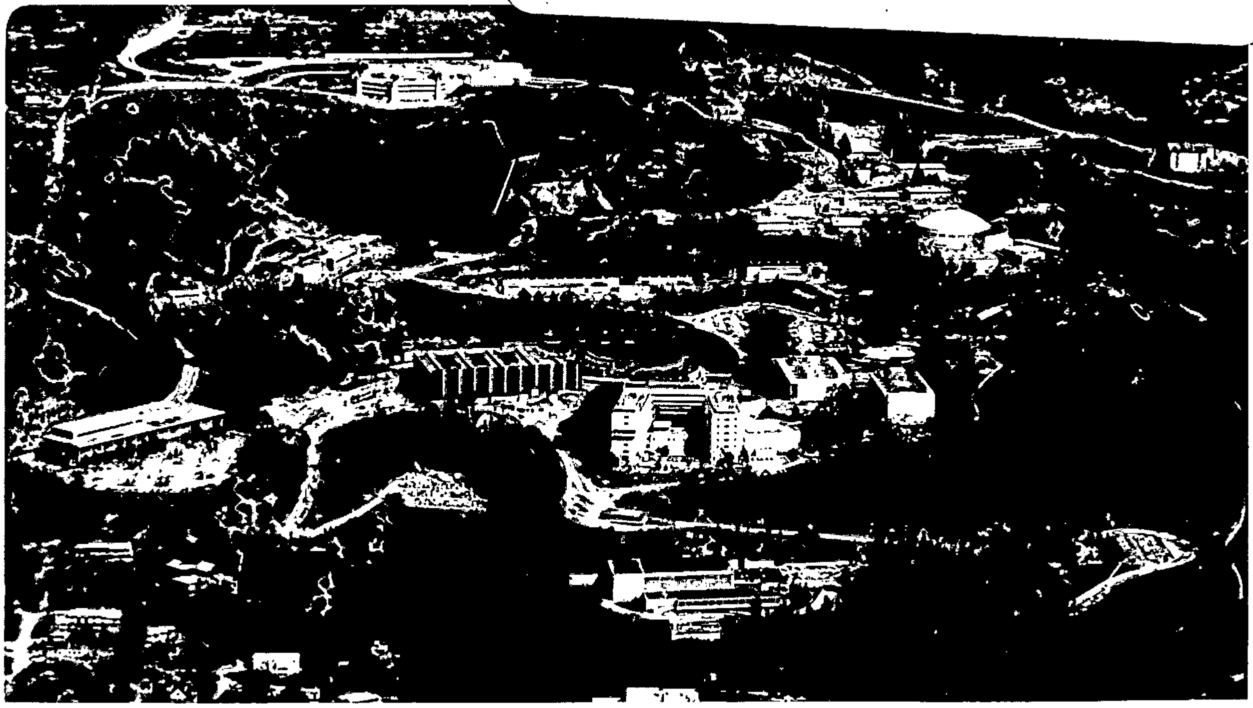
Submitted to Information Resource Management

CAUTIONARY APHORISMS FOR USER-ORIENTED COMPUTER MANAGEMENT

D.F. Stevens

April 1983

TWO-WEEK LOAN COPY
This is a Library Circulating Copy
which may be borrowed for two weeks.
For a personal retention copy, call
Tech. Info. Division, Ext. 6782.



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098.

LBL-9538 Rev. c.2

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

CAUTIONARY APHORISMS FOR USER-ORIENTED COMPUTER MANAGEMENT[1,2]

D. F. Stevens

Lawrence Berkekley Laboratory, University of California

Berkeley, California 94720, USA

Most of the literature of the management of computing is concerned with *technical* management. As computing becomes more deeply imbedded into our daily lives, the human aspects of computer management become more important. User-oriented computer management demands an understanding of human nature, of systems, and of the ways in which the two influence each other. Here, in the form of a commented series of laws and apophthegms, is a compendium of wisdom and experience directed at furthering such an understanding.

Introduction

For he who'd make his fellow, fellow, fellow-creatures wise
Should always gild the philosophic pill.
W. S. Gilbert [1]

The aphorism has a long and honorable history as an effective medium for the dissemination of folk wisdom. It is a dangerous medium in the sense that *any* idea, worthwhile or not, becomes more memorable when presented in a pithy and pungent package. I believe that all of the ideas presented here are worthwhile, and I have tried to find a suitable pithy or pungent expression for each of them. They are grouped into three general categories: non-determinism, user satisfaction, and miscellaneous technicalia. The fact that only one of them is primarily technical reflects the increasing importance of the human side of the human/machine interface.

Please note that this is not a guidebook for success; it is not a set of prescriptions guaranteed to make you a whiz-bang computer manager overnight. It is truly *cautionary*, intended to warn you of some of the problems and pitfalls inherent in the wider integration of the user into the scheme of things. When, in your attempt to become the compleat User-Oriented Computer Manager, you have suffered a particularly distressing setback, a review of these aphorisms may help to remind you that such disappointments are constant, continual, and universal.

[1] This work was sponsored by the U. S. Department of Energy under contract No. DE-AC03-76SF00098.

[2] This is a substantial revision of the paper of the same name presented at the IFIP-80 World Computer Congress.

Much of the content of this compendium is drawn from that mysterious well of wisdom known as "common knowledge". That means I generally don't know the source of the specific formulation used for a given aphorism. (Some of them may even be original.) Sources I do know are acknowledged in the text and the references.

1.0. People are non-deterministic.

A deterministic system is one in which the same set of circumstances always yields the same results. Thus humans, almost by definition, are non-deterministic. It is interesting that when computer people write about *systems* they recognize the dangers of non-determinism, for as Stevens *et al.* have noted [2]:

1.1. Modules that keep track of their own state are usually not predictable, even when error-free.

When they think about the *human* side of the interface, however, they tend to forget it. Humans keep track of their own state and therefore are to be considered as "usually not predictable". People are unpredictable in many ways, but particularly in the ways in which they respond to the trials of system use: Tolerance for the inconveniences encountered when dealing with systems varies widely from person to person, and for any given person it varies just as widely from occasion to occasion. Service that is tolerable under normal circumstances may become unacceptable in times of crisis; a particular idiocy may be acceptable the first eleven times it is encountered but may cause violent reaction at the twelfth instance; a person who is normally placid may become a tiger when his boss, or his girlfriend, is looking over his shoulder.

You cannot be expected to anticipate all of the specific vagaries you will encounter, but you must anticipate that vagaries will occur. If you cannot cope with them when they do, your system may fail, and will certainly fail to please.

But you are not to be downcast if, however much you try to prepare, your preparations turn out to be insufficient, for

1.2. The human capacity for random behavior is always greater than expected.

The failure to allow for this aspect of human-ness has caused the downfall of innumerable systems. As an example, consider the presumably simple matter of names. Most (U.S.) standard forms with space for a person's name require, in some order, "first name", "last name", and "middle initial". The following is a list of problems that have been encountered with this traditional format. (It is not exhaustive.) All of them have caused difficulty for more than one system.

1) there is only one name
2) "first" name occurs last
3) "last" name occurs first
4) "last" name occurs next-to-last
5) compound names
6) first name consists of only an initial
7) two (or more) middle initials
8) no middle initial
9) choice of "middle" changes from time to time
10) prefixes (de, von, ..., and also Dr, Mr, Ms, Mrs, Prof, ...)
11) postfixes (Jr, III, ...)
12) non-English orthography (accents, umlauts, cedillas, ...)
13) insufficient space for one or more of the names

This list illustrates a few of the many sources of human randomness, for some of these "anomalies" are of national origin, some are cultural, some familial or regional, and some individual. A user-oriented computer manager must maintain an awareness of this human potential for randomness, and be sure that the design, implementation, and operation of the systems under her control can accommodate it.

Such an awareness is especially useful when coping with the users' reactions to systems built to their own specifications, for, as noted (in slightly different form) in [3],

1.3. People don't know what they (don't) want until you give them what they ask for.

(This might well be called "The Monkey's Law" in honor of a well-known but rather grisly application described by W. W. Jacobs some years ago. An alternate phraseology of this same principle is found in 2.18.) Much has been written in the trade press about the importance of securing user concurrence on system design before allowing the cement to set. The implication is that with concurrence, duly substantiated by signatures, will come acceptance. We have all seen enough contrary examples to know that that is not the case, but we continue to pursue concurrence anyway, not in order to produce better systems but in order to distribute the blame for the less-than-optimal systems that seem to be inevitable.. We would do better to

1.4. Fix the problem, not the blame.

A large part of the problem is systems that cannot accept the unexpected.

A recognized aspect of randomness is the ability to err; humans carry this aspect one step further:

1.5. To err deliberately is human.

Users are especially fond of exploration and of limit-testing. This activity--called "programming by the experimental method"--can occasionally have serendipitous results: some of the more interesting machine instructions in IBM's 704x and 709x computers were discovered by users who meticulously tried all of the "forbidden" and "undefined" possibilities on predecessor computers. Programming by the experimental method can also lead to spectacular failures. The most famous of these in the 704x/709x case is XEC* * ("execute indirect to self" [4]), the execution of which can best be described by attempting to follow the instructions on a card that says

**Turn this card over and
follow the instructions found in the
place specified on the other side
of this card**

on the front, and

The other side of this card

on the back.

The machine got so involved in turning the card over and over it forgot how to do anything else; the only way out of the bind was to turn off the machine. A system that must self-destruct to undo the consequences of a simple oversight is not very human-tolerant.

Programming by the experimental method is encouraged by two common system defects: poor documentation and unbearable constraints. In the first case, the experimental method is the only way to determine the consequences of an unfamiliar action; in the second, it is employed in the belief that there must be some way to get around a particular limitation. Eliminating these two defects, even if 'twere possible, would not eliminate deliberate errors, however: Human curiosity and cussedness are just too great.

The limiting case of this phenomenon is

1.6. Everything will be tried.

I have recently used an interactive system that quite emphatically stated at one point in the training module "Do NOT press NEXT after you answer Question 24". Two of my colleagues, intelligent men both, but possessed of the spirit of scientific enquiry, pressed NEXT. The system crashed each time. As they discovered,

1.7. No system is foolproof.

Parker [5], Gilb [6], and Murphy (in the exposition of [7]) have all noted that fools attack any system more often, and are more difficult to defend against, than criminals. The latter attack only those parts of the system containing something of value, and in ways designed to shield themselves from detection. Fools, on the other hand, attack all parts of the system with equal intensity, and with an ingenuity that is beyond imagination. There is no defense against fools. You cannot predict where or how or when they will defeat your system, but you can safely predict that they will defeat it. Your only recourse is to do your best to keep defeat from necessarily equalling destruction.

Under no circumstances should you ever be swayed by the argument that "nobody would ever be foolish enough" to do something. Human folly knows no bounds.

2.0. Satisfaction is not just the absence of distress.

"User satisfaction" is one of the catch phrases of the 80's, and part of the catch is that most managers don't really know whether or not their users are satisfied, or how to go about finding out. (Those who are serious about the quest are directed to [8].) The intuitive procedure is to use the complaint level as the major index of dissatisfaction. Thus, by convenient inversion, the manager is tempted to treat the absence of complaints as indicative of satisfaction; but

2.1. Silence signifies apathy rather than acceptance.

While it may be true that happy users do not complain, it is certainly true, as we shall see below, that it is not possible to keep all of your users happy over an extended period. Continued silence must therefore result from some less desirable circumstance.

User comment is driven by two motivating forces: the experience of distress and the hope of redress or improvement. Of the two, the second is more important, for the first alone is, in the long run, insufficient. Continued inattention will silence even the most persistent of naggers.

One of the things a user-oriented manager must do is keep the dialogue between the system minions and the users on a free-flowing and constructive level. It is when things appear to be going well that you most need a friendly and sensitive user who has some expectation of a respectful hearing, for only such an one will bring you the subtle early signs of encroaching saturation or of eroding service. If you have taught her that her warnings go unheeded, she will take great delight in allowing disaster to come upon you unaware.

A manager's best friend is a knowledgeable, critical, demanding, vocal user. If knowledgeable, he knows the significance of the signs he reads; if critical, he will not let you slip into careless habits; if demanding, he will inspire you to better performance, no matter how good you think you are now; if vocal, he will give you the benefit of his informed and constructive criticisms. The elimination of criticism is not a proper goal of user-oriented management; the conversion of diatribe to dialogue is.

If you succeed in fostering a reasonable level of dialogue, you will quickly discover (to your understandable but unavailing distress) that

2.2 Memory is proportional to pain.

The persistence and vividness with which a user remembers the details of any interaction with a system is strongly dependent upon the damage he suffers as a result of that interaction. The limiting case of this phenomenon has been catalogued in [7] as Zimmerman's Law of Complaints:

2.3 A user never remembers a beneficial event.

Understanding and acceptance of this aspect of Murphy's Law are essential to the continuing sanity of the Computer Manager. It is well known, having found wry expression in popular culture by means of the "What have you done for me lately?" sort of joke, but it is easily forgotten. Whenever we are tempted to consider this attitude somewhat unjust, we should remember that a single system error can wipe out not merely the memory of many successful prior experiences, but also their accumulated results. Eastern Airlines has found this principle important enough for integration into their recent advertising: "We're only as good as your last flight." [9] That is an excellent motto for the user-oriented manager in any field to adopt.

Mottoes are not enough, however, for there are inexorable forces at work to prevent us from ever reaching a stable state of user satisfaction, for as you improve the system for some users, you elevate their expectations; the result is more discriminating users. The more discriminating the user, the smaller the disappointment that will be sufficient to provoke misery.

You goal should not be the elimination of misery (for quiet users lead to apathetic management), but a change in the kind of event that brings it on. Thus, if a user today expects his job to complete sometime before next Monday, your goal should be first to elevate that expectation to tomorrow morning, and then to two hours from now, and then to two minutes. You will notice that in the latter case a single minute's tardiness will cause the same sort of disappointment as total loss of the job in the first case. You will have eliminated no misery, but you will certainly have improved your system.

The discerning reader will have noticed the limiting phrase "for some users" a couple of paragraphs back. This is in quiet recognition of a little-known conservation law:

2.4 In any interaction between a well-managed system and a user community, the amount of user misery is conserved.

Thus whatever misery you manage to remove from one user is distributed among the rest. (Please note that the qualification "well-managed" is a necessary element of this law, for it is certainly possible to introduce misery-amplifying changes into any system.) Three specific aspects of this law are worth individual comment:

2.5. There is always a user for whom the system is not designed.

2.6. The full set of user requirements is always inconsistent (i.e., cannot be satisfied by any system).

2.7. No change is transparent.

(2.5 and 2.6 are derived from Ethnotech observations [10].) 2.5 has two aspects: The first is that there are always some kinds of users that you failed to anticipate when you designed the system (the original FORTRAN, for instance, had no provision for character manipulation in the language, but that did not prevent FORTRAN programmers from manipulating characters (and occasionally breaking the system)); the second is that at least one kind of user will be more numerous than you thought possible. In the second aspect it may be more familiar to the systems programmers among you as

2.8. Every table will eventually overflow.

2.6 recognizes that different users have different demands. Rich users, for instance, tend to want instant service, poor ones want cheap service; progressive users want new features, conservative ones want no changes; adventurous users want license, timorous ones want safety. (Lincoln's form of this axiom is most well-known: "You can satisfy all of the users some of the time, ...")

2.7, means, of course, that every time you fix something for one user, you cause something else to break, either for that same user or for a different one. This is the famous "Bag of Jello" problem (poke it in here and it pops out there; poke it in there, and it pops out somewhere else; *et cetera ad infinitum*). We will return to this topic in the next section.

One way to mitigate the misery is to make sure your users know in advance just what changes are coming and how to prepare for them. But put not your trust in printses, for, alas,

2.9. Advance warning is always insufficient.

This is perhaps most well known in its US Navy version, which is of such venerable antiquity that when John Paul Jones uttered his famous rallying cry -- "I have not yet begun to fight" [9] -- a Marine sergeant in the fo'c's'le was heard to mutter "There's always some dumb s.o.b. who don't get the word." No matter how frequently, forcefully, or forehandedly you try to warn the users of an impending change, there will always be someone who doesn't get the word, or who doesn't believe it, or who forgets it, or who *knows* it wasn't intended for him.

I suppose there are a number of reasons for this, but the best two are quite self-explanatory:

2.10. Nobody reads the manual to <u>prevent</u> errors.

2.11. Documents are better reminders than teachers.

Let us return briefly to a consideration of 2.5. It was noted in passing that one aspect of this law is the presence of users who persist in doing *their* work in *their* ways, despite the intentions and limitations of the system. This quite obviously puts some strain on the system, but always remember that [12]

2.12. The value of a system is in the doing of the work, not in the exercising of the tools.

Or, paraphrasing Comer [13]:

2.13. Users write programs to solve problems, not to instruct machines.

The users' attitude towards the problem of adapting the system to their needs is well-expressed by the first of the next two rules, which is sort of a converse to its much-more-well-known successor:

2.14. When a small boy has to drive a nail, anything he can pick up looks like a hammer.

2.15. To a small boy with a hammer, everything looks like a nail.

The second of these complementary statements is known in many variants, and has been rather widely quoted (see, for instance, Kaplan's Law of the Instrument, in [7]); the second is seldom stated explicitly. Between them they address the extreme philosophies employed when a human has a problem and a tool that are ill-matched. The boy with the hammer provides a reminder of the human tendency towards a well- known kind of psychological myopia in which our perception of a problem is distorted by the tools at hand: the fact that we know some statistical analysis leads us to consider user satisfaction a statistical problem; the presence of a hardware monitor encourages us to view a probe point library as the key to system efficiency. A somewhat catchier name for this syndrome, which is frequently encountered in the data base environment, is "product-oriented solution": a "solution" in which the user is required to convert his problem into the particular type of nail for which the system's hammer was designed.

2.15 also says that the systems you give your users determine the kinds of problems they can solve conveniently, and thus have great influence on the kinds of problems they will attempt. If your users need to do more than just drive nails, you'd better give them something more than a hammer.

2.14, on the other hand, reflects the ingenuity your users will display in using systems for their purposes, whether or not those are the purposes the designers had in mind during development. (Those character manipulators in FORTRAN again.) And once having twisted the systems to their purposes, the users will proceed to complain, often bitterly, that these misfit systems don't satisfy their desires *conveniently*.

The whole of the user satisfaction discussion can be summarized metaphorically as:

2.16. Systems run better when they run downhill.

This is a slight modification of one of Gall's observations on Systemantics [14]. It suggests that when your system is helping people to do what they want to do, it has some chance of success. It further suggests that the chance of success will be enhanced if its methods and means (as well as its goals) are sympathetic to the users. Gall's formal characterization of this principal (also in [14]) is rather more pretentious, but nonetheless to the point:

> Systems aligned with human motivational vectors will sometimes work. Systems opposing such vectors work poorly or not at all.

If you wish your users to be satisfied, discover those vectors and follow them. A homely, but undoubtedly familiar, example may serve to set this precept in your memory:

2.17. If *logon* isn't easy they will never *logoff*.

One of the biggest problems between us and our users, of course, is communications. The problem was alluded to in 1.3; it can be expressed more directly as

2.18. "Pay attention to what I mean!"

Users often find it difficult to get us to pay attention to what they say; it is far more difficult to get us to pay attention to what they mean. The problem is that many of us, including users, have trouble expressing ourselves clearly, especially in times of stress. The fault may be in us or in the medium of communication. The User-Oriented Computer Manager must be sensitive to at least two areas where this problem can arise. The first is in the communication between a user and the analyst to whom he has come with a problem. He frequently fails to realize that

2.19. A proposed solution is not a statement of the problem.

(Adapted from [3].) The user means to talk about a problem, but frequently in fact talks about his partial solution and bypasses the original problem, thus obscuring the problem and generally muddying the waters. In his efforts to avoid falling into this trap, the analyst must be equally chary of taking *his own* proposed solution as necessarily being a statement of the real problem. Awareness of our tendencies in this direction, plus a gently probing style, are the analyst's best allies in this situation.

The second major area is in the languages and pseudo-languages with which the user must attempt to communicate with the system. System manufacturers have been of absolutely no help, creating a Babel of incompatible and self-inconsistent monstrosities. Your job is to provide, where possible, a natural overlay that your users will have trouble misusing. Where that is not possible, sympathetic documentation is the only palliative you have.

But (sigh) we have already seen (in 2.10 and 2.11) how well that works.

The last great opportunity to improve your user/system interface is provided by the system test. The hidden flaw here is that the test subjects over which you have the most control tend to be computing professionals, and therefore not representative of your users:

2.20. A system test based upon the reactions of programmers is worthless.

This has been observed by Stemple [15] and, I am sure, many others. Ethnotech [10], for instance, has noted some specific aspects of the problem: Programmers see blanks where people do not; programmers slash their zeros or their O's, whereas people slash neither; programmers count from zero, people from one; programmers print, people write. (One used to be able to say that programmers print *in capital letters*, but the spread of two-case terminals has rendered that distinction somewhat obsolete. Perhaps a more up-to-date statement would be "programmers ignore case".) Programmers are much too flexible and adaptable to be accurate predictors of real human behavior. Programmers are used to, and enjoy, behaving and communicating like machines; given their preference, many of them associate with machines rather than with people. Programmers are addicted to such cabalistic practices as using zero to indicate infinity, doing hexadecimal arithmetic, speaking in acronyms, and spelling words backwards to bracket constructs (as in IF....FI; I am surprised that BEGIN hasn't been replaced by DNE, or END by NIGEB).

It should be clear that programmers are not to be trusted to evaluate the human interface of any system: if you want to know what real people are going to think of some aspect of your system, you're going to have to get some real people to try it out.

3.0. PM stands for "provocative maintenance".

I first heard this principle stated at a users' group meeting in the 60's. The speaker was John Denes of the Brookhaven National Laboratory, discussing a site survey that disclosed a then surprising very high and statistically significant positive correlation between the amount of PM (standing for either *preventive* or *preventative maintenance* (the priesthood couldn't agree upon its proper name)) and the amount of down time experienced. He was apparently the first to notice how much of that down time occurred immediately after PM. (It has been pointed out that perhaps the vendor was merely trying to return to the old ways, inasmuch as an archaic meaning of "prevent" is "to go before"....) I have since heard this sentiment echoed several times by other people, most recently in March of 1983. It seems a suitably cautionary introduction to the "technical" section of the paper.

Most of this technical section will consider various aspects of the performance of a system, but first I will make good on the promise to expound a bit upon the "Bag of Jello" problem. Of the various formulations that exist I most like those of Gall, Weinberg, and Udall (in [14], [16], and [17] respectively):

3.1. Complicated systems cannot be forced to work.

3.2. A system is a collection of parts no one of which can be changed.

3.3. Every reform always carries consequences you don't like.

3.1 is a rather bald statement of the situation, 3.2 suggests why it should obtain, and 3.3, with its reminders of Danton and Robespierre, should serve to warn you of the potential gravity of its consequences. The particular problem, as Gall explains, is that many complex systems *do* work, but they work to the beat of their own drummers, not yours. In your attempts to impose your will upon a recalcitrant portion of the system you will discover 3.2; if you persist, 3.3 will rise up and bite you. Don't forget that any attempt to correct a bug is by definition an attempt to install a change or a reform....

As a prelude to the Fundamental Law of System Performance, consider the following:

3.4. An on-line system is underutilized, overloaded, or both.

This is a law of the excluded middle, there being no possibility of a happy medium; it is a corollary to the Fourth Law of Hebditch[18]. It derives from the fact that a successful on-line system carries with it the seeds of its own destruction. Success begets popularity which begets overcrowding.

That both attributes can be present arises from the existence of multiple viewpoints. Viewed from the austere confines of the comptroller's office almost all on-line systems are underutilized. Viewed from the bleak wasteland of the users' terminals almost all on-line systems are overloaded. The presence of multiple viewpoints makes application of our next principle sligfhtly more difficult, for there may be disagreement over the identity of the "most valuable" resource. Nonetheless, you should try to

3.5. Optimize the use of your most valuable resource.

This can be particularly tricky during periods (such as the present) when the most probable candidate for "most valuable resource" is changing (in our present case, from hardware to people). It is tempting to equate "most valuable" with "costliest", but that is not always the case; there are times when the scarcest resource is the most valuable. A rule of thumb that can help you to determine which of your resources is most valuable has been often quoted by James Baker of the Lawrence Berkeley Laboratory:

3.6. When saturated, work to maximize throughput; otherwise work to minimize turnaround time.

This recognizes the fundamental change in character that saturation imposes upon a system. A saturated system is one that has more work to do than it can accomplish. (A saturated system is not necessarily full in the sense that all resources are fully committed: a single over-committed resource is enough to cause saturation.) The distinguishing characteristic of a saturated system is the presence of lengthening queues. In this situation the strategy of least discontent is to adjust the system to do as much work as possible. (If you are extremely clever -- or lucky -- you might even adjust it well enough to eliminate the saturation.)

By contrast, an unsaturated system can handle all of the work that is submitted. You can then turn your attention from simply doing the work, to doing it in the nicest possible manner. This normally means adjusting the system to complete the individual jobs as quickly as possible.

Note that reducing individual turnaround time is a far different thing from reducing total turnaround. This can best be illustrated by a slightly contrived example. Consider a system capable of running four jobs at once, and assume that four identical one-hour CPU-bound jobs enter the system at the same time. The best one can do for that total set of jobs is four hours. Most simple round-robin schemes would come very close to that minimum, with the four jobs progressing simultaneously through the system like a well matched team of horses, and all finishing at about four hours after submission. By contrast, a very old-fashioned sausage-style scheme (which allows only one job at a time to run, but where a job once started runs to completion) would also require four hours to complete, but three of the jobs would have finished much earlier (one at the end of each hour).

The two schemes have identical total turnaround times; the user-oriented scheme, however, has a 2.5-hour median while the other median is 4 hours.

In working for individual turnaround rather than for total turnaround you are moving away from *center-oriented* measures and twoards *user-oriented* management. This is quite in accord with the Fundamental Law of System Performance:

3.7. Performance is as perceived, not as measured.

Despite Lord Kelvin [19], numbers are no substitute for judgement. That is not to say that one cannot -- or should not -- measure performance, but that measurement won't necessarily pinpoint the problems the users see. Users evaluate a system by comparing its performance with their expectations and their needs, not by checking its performance numbers. A weak system that is available when needed is better than a powerful one that is never available.

Users' expectations can be unrealistic. A part of your job, therefore, is to conduct a continuing course in realism. You must beware of the temptation to define "realism" as "the immediately attainable". A vocal user corps will help you to avoid this temptation, but only if a good relationship has been established.

3.8. Systems adapt to the measures used for their evaluation.

This is in some sense the system analogue of Darwin's ideas on the origin of species: the organism that survives is the one that adapts to the exigencies of daily life. In the world of computing systems those exigencies have tended to have such names as "high CPU utilization", "depth of multiprogramming", etc. System changes that enhance these qualities persist, those that degrade them do not, and the system evolves. True cynics might prefer to see the management version of this principle:

3.9. Systems adapt to the measures used for the evaluation of their managers.

Although the former absolute insistence upon high CPU-utilitzation seems to be easing a bit, there remain vestiges of that attitude. I have long believed that the quickest way to achieve high utilization of any component of a system is to multiply the system manager's base salary each month by the achieved utilization of that component. The more creative among them would doubtless soon find ways of achieving utilizations of 120% under these circumstances.

The following (from [20]) is easily seen to be a corollary to 3.8:

3.10. Unless the goals (of a system) determine the measures, the measures will determine the goals.

Unless the measurement program is consciously applied as a means to an end the evolution of the system will cease to respond to true needs and, in accordance with 3.8, will evolve so as to optimize the measurements. This is a technical seduction that is very difficult to avoid. It results from

3.11. Quantities that are measured tend to become more influential than those that are not.

You also need to beware of the obvious measures, because

3.12 The easier a measure is to obtain, the more likely it is to be misleading.

The measures that are easy to obtain are the ones your manufacturer wants you to take (and remember, your manufacturer is neither objective nor disinterested), the ones that make the system look "good" even while it saturates. Thus, for instance, the vendor of an architecture that has very simple instructions that execute quickly is more likely to push MIPs (millions of instructions per second) as a measure of raw power than is a vendor whose architecture contains complex instructions that do a lot but are relatively slow. They may both do your DO loop in ten microseconds, but the first may require 50 instructions while the second needs only three. The easy measures include averages instead of medians and distributions, and percentages of doubtful bases. This topic is covered in more detail in [21].

The last word:

Fuzzy.

4. People are fuzzy, computers are not.

This statement of the principal difference between humans and computers is due to Zadeh [22]. People are accustomed to thinking in imprecise terms, communicating via incomplete utterances, coping with garbled information. To a great extent they expect to be surrounded by such fuzziness and are dependent upon it. They become uncomfortable when confronted with too much rigidity and exactness; they feel threatened when they are required to deal with traditional computing systems that allow no imprecision of statement, no incompleteness of thought, no variation in form. One of the most important tasks of the User-Oriented Computer Manager is to build suitably diffuse interfaces between his hard-edged machines and his fuzzy users. With such interfaces his users have some chance of feeling comfortable; with such a feeling of comfort comes some chance of a satisfactory relationship.

* * * * * * * * * * * * * * *

L'Envoi: This set of aphorisms is neither complete nor independent; it well may be inconsistent. (That would merely reflect the complexity and fluidity of the task of the User-Oriented Computer Manager.) Much of it is self-evident perhaps even to the point of triviality. But truth is none the less truthful for being trivial, and we often lose sight of the self-evident. Perhaps it will serve as a useful reminder from time to time; I hope so.

REFERENCES

[1] W. S. Gilbert, The Yeomen of the Guard, 1883.

[2] W. P. Stevens, G. J. Meyer, and L. L. Constantine, "Structured design", IBM Systems Journal, Vol. 13 No. 2, 1974, 125.

[3] D. C. Gause and G. M. Weinberg, Are Your Lights ON?, Ethnotech, Inc., 1977, 143.

[4] IBM 7040-7044 Principles of Operation, Form A22-6649-2, IBM Corporation, 1963, 9-27.

[5] Donn Parker, "Computer security differences for accidental and intentionally caused losses, AFIPS Conference Proceedings, vol. 47, 1978 National Computing Conference, 1145-1149.

[6] Tom Gilb, "Laws of unreliability", Datamation, March, 1975, 81-85.

[7] Paul Dickson: The Official Rules, Delacourt Press, 1978.

[8] S. W. Pearson,Measurement of Computer User Satisfaction, unpublished PhD dissertation, Arizona State University, 1977.

[9] From an Eastern Airlines radio advertising jingle, June, 1979.

[10] Ethnotech seminar on Measuring and Increasing User Satisfaction, presented in Washington, DC, December, 1977.

[11] In reply to the British commander, who enquired, when the US flag was shot from the mast after two hours of point-blank firing, "Have you struck your colours?"; in the battle between the Bon Homme Richard and the Serapis; 1779.

[12] D. F. Stevens, "Thoughts on usability", EDP Performance Review, April, 1983.

[13] D. Comer, "Principles of program design", IEEE Transactions on Software Engineering, March, 1981.

[14] John Gall : Systemantics, Quadrangle, New York, 1975, 87-94

[15] David Stemple: Oral comment at VIM-30, May, 1979.

[16] Gerald Weinberg: An Introduction to General Systems Thinking, Wiley, 1975.

[17] Morris Udall: Quoted in "Faded Fiefdoms", Dennis Farney, in the Wall Street Journal for May 3, 1979. The source of the comment is Udall's experience in Congress, first as a junior member hampered by the great power of the committee chairmen, and later as a committee chairman deprived of power by a reform he had helped draft.

[18] David Hebditch: The Ten Laws of Teleprocessing, Datamation, November, 1975.

[19] Lord Kelvin, 1883: "...when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind." Quoted by M. M. Lehman in "Performance Evaluation, Phenomenology, Computer Science, and Installation Management", in Performance of Computer Installations, Ferrari, ed, North Holland, 1978; and by Tom Gilb in Computerware, manuscript in progress, 1977.

[20] David Stevens: "Towards User-Oriented Performance Management", EDP Performance Review, May 1979.

[21] David Stevens: How to Improve your Performance Through Obfuscatory Measurement, NCC-78.

[22] Lofti Zadeh: Quoted in the Preface to Fuzzy Automata and Decision Processes, Gupta, Saridis, and Gaines, eds, North Holland, 1977.

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720