C. A. Baker

KNOWNS AND UNKNOWNS IN IMMEDIATE ACCESS TIME SHARED SYSTEMS

by

Murray Turoff

INSTITUTE FOR DEFENSE ANALYSES

(Speech presented at the American Management Association Conference on The Computer Utility, March 8-10, 1967 in New York City.)

KNOWNS AND UNKNOWNS IN IMMEDIATE ACCESS TIME SHARED SYSTEMS

by

Murray Turoff

Introduction

The white man drew a small circle in the sand and told the red man, "This is what the Indian knows," and drawing a big circle around the small one, "This is what the white man knows." The Indian took the stick and swept an immense ring around both circles: "This is where the white man and red man know nothing."

- Carl Sandburg
  The People, Yes

Ladies and Gentlemen, it really doesn't matter much which of us are the white men or which of us are the red men, for in this instance we are dealing with a situation where the questions of interest lie in the unknown region outside our immediate circles of knowledge.

Because of this, there is very little in the way of cold hard facts that I can offer you. Rather, I am forced to offer you a mixture of intuition and, hopefully, common sense, based upon my own observations. However, it is hoped these opinions can be put in a light which will make them somewhat obvious to your own intuitive sense.

First, let us try to remember that we are not dealing with new ideas but rather a new combination of old ideas. "Immediate access" for humans has been with us since the first hand calculator and is still with us in the use of the "small" computers. What is meant by immediate access is the ability of a user to sit down at a console (perhaps a small computer or perhaps a console) and in some manner control or manipulate the task he wishes to accomplish.

Second, the area of process control (control of a physical process by a computer) has lead to many of the ideas which went into the time sharing systems with which you may be familiar. The unique difference is that it is no longer a chemical or physical process we are attempting to aid by increasing efficiency. Rather it is a mental process as represented by a group of human beings. I can truly perceive of no process which is more unpredictable. At least, in the standard process control applications there always existed a group of experts (e.g., chemists, engineers) who could define the process, indicate the best ways to aid it and provide warnings on what sort of overload or troublesome situations might occur. In attempting to aid mental processes, we would conjecture that, perhaps, the field of psychology would be able to contribute greatly to the construction of meaningful immediate access systems. However, when one looks at some of the current systems of this nature, it becomes quite evident that the evolution of these systems has not been overly influenced by this field. As you may begin to suspect at this point, it is perhaps an interesting philosophical commentary on this subject that before you today is a physicist in the guise of a computer expert and acting like a psychologist.

The reason for this rather illogical development is partly historical and partly human nature. As computers and demand grew bigger and humans were pushed farther and farther away, longer and longer delay times in running programs resulted; for a few minutes of computer time, hours of waiting resulted. Now, humans have the wonderful quality of being able to adapt to almost any situation, even though they may not like it. However, if they really dislike a situation and are given an alternative which is infinitely more pleasing, they will immediately change and be overjoyed at the prospect. So overjoyed at the comparison between what they now have and what they had in the past, that many of them forget to ask if what they have now is really what they want. In other words, in this drastic transition, anything that gives me immediate access is better than nothing, so let me not be too critical of what I get.

My feeling is that in most of the first generation systems very little thought has been given to providing the user with any sort of optimized system, but just a system.

In talking to many users of such systems the common attitude or first remark is that what they have is great. However, a little discussion begins to indicate that there are things (so called, little things) that they don't like but with which they are willing to live for the convenience of sitting at the console. I might also point out that I've met a number of non-users who will not utilize the system because of these "little things."

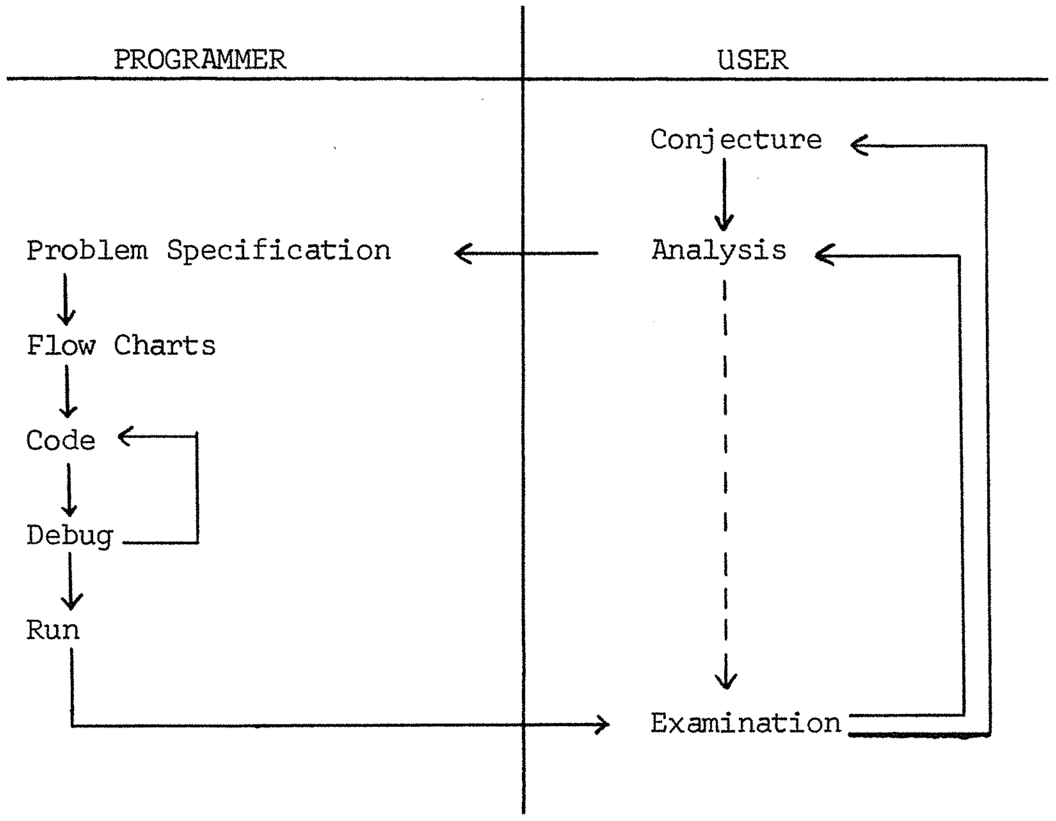
My purpose today is to try to illuminate for you what the requirements would be for a system which approached the greatest possible convenience for the user, one which would conform to the desires of the individual user and not force the user to conform to it.

<u>The User</u>

Who is the user? To me he is not what we think of as the programmer. After all, the poor programmer programs for a living and he will be forced to utilize whatever is provided him. Rather, the user is the person who does not <u>have</u> to use the system. He has the freedom to forget about what he thought of doing, doing it by hand, giving it to the programmer, or sitting at an immediate access system to do it himself. We would like a system that will sway him to take the latter of these alternatives.

Basically, he is a professional man (manager, scientist, engineer, etc.). To him the computer is a sophisticated hand calculator which may or may not aid his major task.

First, it is probably useful to contrast the approach he takes to a problem with the common approach of the programmer when the programmer is at the service of the user.



From this simple little diagram we can infer a number of conclusions:

1) The user would like to avoid concern with what is taking place on the left side (programmer) as far as possible.
2) Whatever must occur on the left should take the shortest possible time.
3) Trying to turn the user into a programmer will discourage a lot of potentially important users.

Therefore, assuming the user wishes to be primarily concerned with his problem and has no interest in computer-unique problems, the major goal of immediate access systems should be to eliminate the left-hand side as much as is economically feasible. This is not to imply that the easement of the programmer's problems is not a worthwhile goal in itself, and I will comment on this later.

One additional problem, besides the delay time, which the user encounters if his approach involves going down the left-hand side is that of communication. The programmer does not have the competence of the user in the problem area and may tend to inject unintentionally misleading logic in the coding stage. This often leads to a certain amount of insecurity on the part of the user when the results are obtained. This, coupled with the delays, severely limits today the number of users who utilize computers and the versatility of the approaches taken when the computer is utilized in the batch environment.

This problem of communication between user and programmer is a good deal more severe than is often realized on the surface. Exact information is rather hard to obtain. However, there recently appeared in Data Processing Magazine (January 1967, "Using the Remote Console") an article that, besides illuminating this point, is so much in direct disagreement with many of the views expressed here, that I strongly recommend it to your attention, lest you feel there is only one side to the coin. The major conclusion of this article, basically, is that since the average user today does not know what a computer is capable of doing or how to use it efficiently, it would be a disaster to give him a remote console unless we train him to be a programmer, e.g., construction of a complete definition of the problem before beginning, flow charts, test cases, input-output, pre-writing of program, etc. The facts presented to support the premise of this conclusion are those obtained in a survey of 25 programmers -- not users. It is enlightening to compare some of the questions asked of the programmers with what might very well be the users' reaction to this, as follows:

	Programmers' Answers	
	Yes	No
When you first discuss a problem with a user, does he usually know what he wants?	35%	65%
Does he usually define the task in writing?	18%	82%
Does he usually define input and output in writing?	12%	88%

USER: IF I KNOW THE APPROACH TO THE PROBLEM THAT WELL, IT PROBABLY IS NOT WORTH DOING!

	Programmers' Answers	
	Yes	No
Does he make any attempt to learn what is involved in programming?	18%	82%
Does he usually know what a computer can do for him?	6%	94%
Does he adequately monitor what you are doing?	18%	82%

USER: IT TOOK MANY YEARS FOR ME TO BECOME COMPETENT IN MY FIELD; DO I HAVE TO SPEND A SIMILAR AMOUNT OF TIME BECOMING COMPETENT IN YOURS?

One additional ground rule I would like to establish before we proceed is that I'm dealing with a user who has a conjectural type problem - one in which he may not be sure if he is asking the question in the proper manner or applying the proper analysis. He will not be sure of his question or analysis until the results have been examined. I wish for the moment to divorce this from the production run type problem where the question and analysis are well defined and what are required are merely repetitive runs of the program as the data fluctuates in time. This latter case is, of course, the bread and butter operation of most business computer installations today. I would only wish to warn you that when I take this subject up, my opinions on immediate access and time sharing will take a 180° turn with respect to usefulness.

Now that a basis has, I hope, been established, I believe we can begin to itemize a set of "user" requirements.

Atmosphere

If you recall, our user has obtained a certain level of competence in his field. Furthermore, not having had years of computer experience, he does not realize that even the best programmer expects to make all sorts of idiot errors when dealing with a new system or language. Therefore, in order to prevent what he considers to be a show of stupidity, he wishes privacy - he and the console alone in a room, or better yet, in his office.

Also, at least for myself, I find it very distracting to try to think through a problem with consoles or key punches chattering away. Our first condition, then, is that the user must have privacy and quiet.

While it is very expensive to put a console in every office, it is not expensive to put a $20 plug in every office and mount the consoles on casters to provide mobility.

Another point is that users should not be charged individually for this service. The financing should be out of overhead for a group of similar users. The time might be monitored which each user spends for the purpose of discovering any unusually heavy usage by a single individual, and then request justification in that case. But once again, if a user feels he doesn't have the facility of an experienced programmer at this game, he might also feel he shouldn't waste his budget on the learning effort. The user should have the same attitude toward this service as he would have in obtaining the use of a secretary from a pool existing for that purpose.

The Terminal

While a time sharing system may be able to support a limited number of visual devices reserved for specialized problems, I feel that for some time to come the average user with a "typical" problem is going to have to contend with the typewriter or teletype. Furthermore, even with some type of temporary visual display, it is usually important for this user to retain some sort of written record of his work for some period of time. Especially in the iterative type thought and analysis procedure we have described, it sometimes becomes impossible to recall from memory the details of what took place only the previous day.

It is rather disturbing, therefore, that while a lot of money and effort have been poured into the area of visual display, only minor consideration has been given to the typewriter. The question that I would like to address is, what would users like to see in the way of modifications to this animal?

There are a few obvious points I would like to make:

1) Users with typing experience find the feel of a typewriter a good deal more comfortable than the teletype. This is not a crucial question but merely indicates the most appropriate choice if there is no "severe" economic penalty involved.
2) The flexibility of having both upper and lower case letters is a good deal more important. All upper case just slows down reading and comprehension and is somewhat tiring if one is not accustomed to it.
3) Two color ribbon on the typewriter is the simplest solution to providing a record of what the user is saying as opposed to what the computer may say. This is particularly helpful in looking back upon previous work.

None of the above modify the typewriter itself. In fact, the only direct change I would like to see is an automatic backward line feed so that those people who wish it will have more flexibility both in plotting and constructing tables.

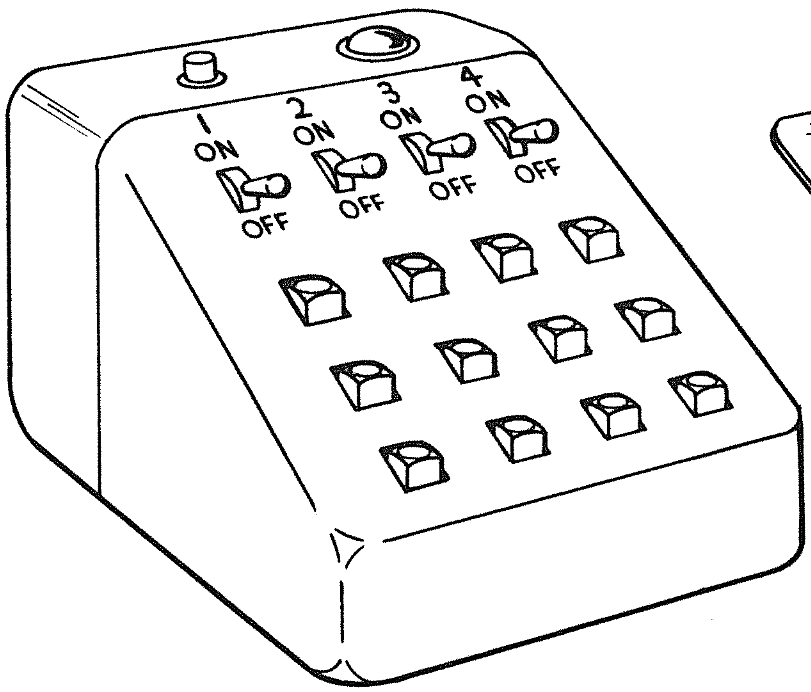
The major change is the addition of a special box containing (see figure):

1) one light and one panic button
2) three to five on-off toggle type switches
3) about ten to twelve blank keys

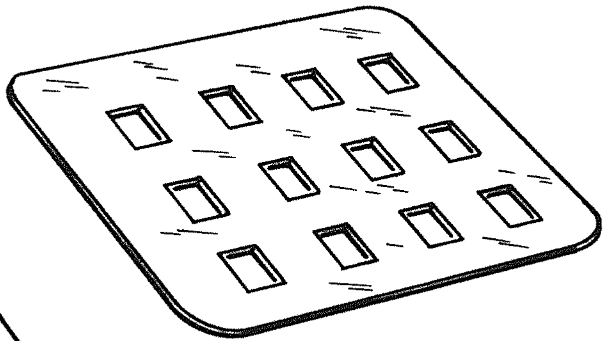
This is the USER BOX; it is really meant to keep him from becoming frustrated. The first thing that disturbs a user is when he sits at the console waiting for something to happen and nothing does. Is he tied up in an endless loop of his own construction or is the system neglecting him? The little light will blink in proportion to the amount of effort (computer main frame time) that the user is receiving. The panic button will give a high priority request to the system to provide a status report on his program (time run and part of program being executed).

The toggle switches are found on every computer console and are usually referred to as "sense switches." The user may program a request in his program to test if a given switch is on or off and act accordingly. This provides the user with a great deal of logical flexibility in controlling the logical flow of his program with an absolute minimum of effort. In other words, he does not have to construct artificial indicators to enter from the typewriter and keep track of; another step in removing the conscious effort of programming. In talking to anyone who has operated a program at the console of a small computer you will find frequent use of this concept.

The ten blank keys are representative of a concept I would like to see plagiarized from the Culler-Fried system. This is a system developed for physicists; it is somewhat tricky in that it involves two full keyboards. However, there is one row of keys on the auxiliary keyboard that represents a set of levels. If the key for the first level is depressed the other board operates like a hand calculator. The second level key turns the board into an algebra machine (a single letter representing a single number), while the next level



USER BOX

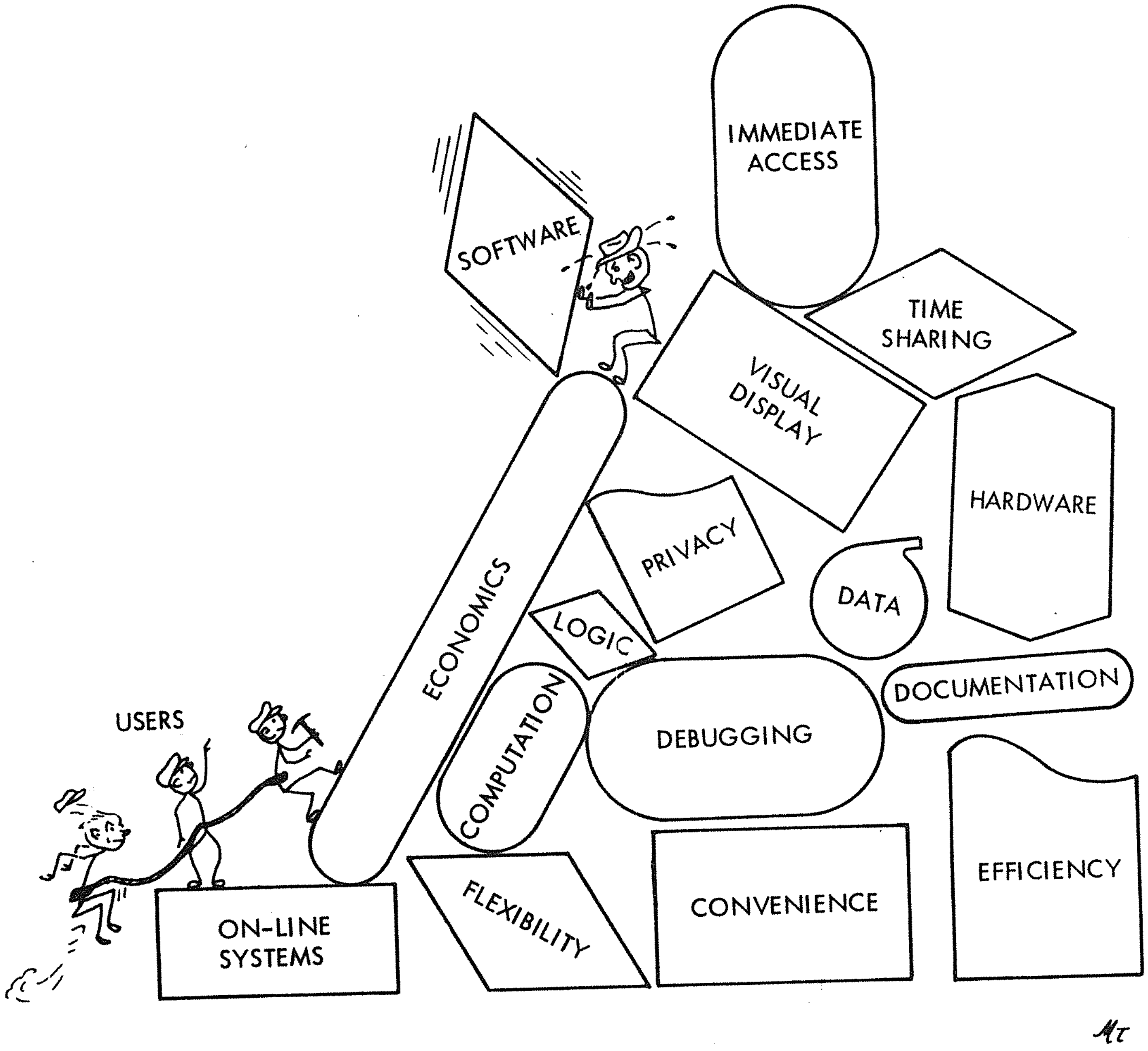


PLASTIC OVERLAY

indicates a vector keyboard (a single letter representing a row of numbers), and so on, up to very complex operations including a level where each key may represent a different program previously constructed by the user. While the complex mathematics with which this system deals are not often employed in commercial operations, there is an analogous situation when dealing with a large file of data and each level might very well represent a hierarchy of data classes or groupings. Also, I feel it is very satisfying for the user to have a group of keys he can call his own and define as he sees fit. We can further provide each user with little plastic overlays on which he may write as he sees fit and place over other keys.

A General Comment

At this point one may detect a general premise behind our approach which should be clarified. The user wants to expend a minimum of effort on anything not directly related to his problem. If he could grunt and the machine would pour forth the answer he wants, this would be a preferred mode of operation. We might note that in any situation involving human beings and a premium on time, the effort of communication tends to be minimized. As examples, take men at war, air traffic control, or taxi drivers. Hopefully, this natural selection process may someday see the death of such items as COBOL, FORTRAN, and some of the more elaborate natural language schemes. The user wants the shortest possible representation of his desire - a specific phrase, a single word, a press of a button. In one of the mac languages, for example, the user may use the option of typing a few letters to indicate a particular command word. The computer will query him if he has abbreviated the word to the point where the computer cannot distinguish it from other possible words.


IMMEDIATE ACCESS
SOFTWARE
TIME SHARING
VISUAL DISPLAY
HARDWARE
PRIVACY
DATA
ECONOMICS
LOGIC
DOCUMENTATION
USERS
COMPUTATION
DEBUGGING
EFFICIENCY
ON-LINE SYSTEMS
FLEXIBILITY
CONVENIENCE

Software

I believe the previous comment leads us to the heart of the problem. After all else is said and done, the biggest stumbling block remaining to be surmounted by the user is the software or language into which he has to translate his problem.

A truly general purpose user language for a time sharing environment does not yet exist. However, there have been some notable successes in user languages oriented to numerical problems. The languages that stand out here are BASIC, TINT, and JOSS. Of these, my own opinion is that JOSS is the most successful in meeting many of the requirements of a user language. Let me now try to go through what I think these requirements are.

1) With not more than a half-hour's instruction or reading, a user should be able to do elementary problems of interest to him at the console. Examples of this are using the system as a hand calculator or requesting and regrouping items of data from a file. Remember, we have already stated our user has demonstrated a level of competence in some endeavor; we do not mean to imply that just anyone should be able to do this. It should also be pointed out, however, that in our own organization and a number of others, some individual users have been able to get their secretaries to do straightforward hand calculating type problems for them on systems such as BASIC.

2) The system must contain informative error messages and educational information which the user can request in order to expand his knowledge of the system while he is at the console. He should be able, for example, to request an explanation on a particular type of

instruction without any interference occurring to the work he has in progress.

3) The grammar should be simple, probably simple declarative sentences or questions. Instruction words should be short as a rule, and the user should have the freedom to abbreviate as much as he wishes.

4) It should always be clear to the user, when he looks back on what happened, what <u>he</u> typed, what <u>his program</u> typed, and what the <u>system</u> typed.

5) The methods of constructing the logic of his program should not be constrained artificially by the software. We would hope to see a logical flexibility similar to that exhibited in the GPSS approach. Users should be able to treat bits and pieces of a problem and tie these together in whatever logical pattern the problem exhibits and not be constrained by the sequential nature of the monitor system in their thinking.

6) The system must eliminate artificial constraints on the user which to him have no logical foundation. Examples of this are the mixed mode restriction in FORTRAN or the artificial distinction between numbers with decimal points from those without. Furthermore, exact arithmetic should be employed (decimal internally rather than floating point).

7) Particularly in commercial applications, it is going to be necessary to have versatile but relatively intuitive ways of manipulating and associating data, both numeric and alphanumeric. Here one would like the sort of capabilities found in a language like SIMSCRIPT. As yet however, there is no general user language, to my knowledge, providing these abilities.

8) Another neglected area in the software, also of importance to commercial application, is the flexibility of different users at different consoles to interact in composing and running the same program. This, I'm sure, would make the modeling or gaming of various business situations a much more realistic or efficient tool than it is in current practices.

9) Last and most important (important because it is the item whose lack causes me the greatest waste of effort every time I sit down at the particular time sharing system my company utilizes), is the concept of interactive editing, correction, and modification. Once a user has a program running, it might stop because of an error or because he doesn't like the results he is getting and he decides to stop it. At this point the user would like to be able to look at data he may have computed in the running but didn't earlier think he should print out, or he may want to change some of the actual program or add to it. Then he would like to be able to start the program running from the point where he left off or any other point of the program he desires without destroying any of the results he has obtained to that point. It might surprise some of you that very few of the on-line systems allow the user this flexibility. BASIC, for example, does not while JOSS is the exception and does. The typical system forces the user to go back and begin his program at the very beginning.

It is quite obvious that if the user has had a program which provided him good results for ten minutes of his time before he reached a part of the program that didn't operate the way he hoped it would, he is going to become frustrated at having to sit through that ten minutes all over again. The basic reason this occurs is that some systems programmer feels that the resulting machine level language program from a recompilation every time a change is made to the user program is 20% or so faster or has some advantages for the time sharing system monitor. If he is right - and I'm not so sure he is - then it means the computer wasn't designed well for this particular job. But what the systems programmer has really forgotten or is unaware of is the nature of the user and his problems. The user is treating a conjectural type problem with an uncertain analysis. Very often there are a lot of preliminary steps, tasks or computations to accomplish which are fairly well defined and only lay the groundwork and organize or create the data for the meat or difficult part of the problem. It is this latter part of the program that will probably be modified many times by the user. The user will just become very upset if every time he makes a change he has to repeat all the initialization section of the program. The systems programmer may be satisfied that the user's program runs 20% faster, but in fact, he may be causing the computer to do ten times the work and the user to take ten times as long to reach a solution. Trying to be as impartial as possible, I consider this interactive feature to be fundamental to a user system.

Basically, we have been trying to describe a software language which contains in it the power of languages like FORTRAN, GPSS, SIMSCRIPT, and perhaps COBOL with respect to the report generating features. However, we don't want it to look any more complex than something like JOSS. We want the grammar to be common through all the diverse applications that this level of capability implies, and more important, we want rather natural subsets of the language to exist with respect to vocabulary and language so that a user can learn a very elementary language to begin to operate on the system and at the same time prepare himself for increasing his scope without undue effort (i.e., learning a new grammar of separate language). Furthermore, we really don't want to force the user to learn any more than he feels he needs to accomplish his task. If the user is only concerned with numerical analyses, he should only have to learn enough to do that, while the user who wants to merely retrieve and examine data stored in a large file should have a subset of language available to him for this purpose which to him appears complete. The user who wishes to accomplish both tasks should not be faced with a changing grammar or language as he moves from one task to another.

There is a movement underfoot to impose standards upon software. I tend to feel this is a particularly inopportune time. We may be entering an era where a process of true natural selection (survival of the fittest) could take place in the software area. I would hate to see the lifetime of relics like FORTRAN or COBOL artificially prolonged merely because of inertia.

Most of the large time sharing systems today are collections of many different languages oriented toward different tasks with the user finding it impossible to communicate problems from one language to another. The user's approach to a problem is largely limited by the particular language he utilizes. Very often complex real world situations modeled on the computer reflect limitations in the software more than they do the real situation. Hopefully, we should have a new generation of software reflecting the supposed increase in computational power inherent in the new generation of computers. For the user - not the programmer - the answer definitely is not PL-1.

Perhaps one of the current problems is that a language like JOSS, for example, appears so simple on the surface, so easy to employ for simple problems, that many people are not aware of the sophistication and power built into it until they have gotten to the point of really applying it to a difficult problem. We should realize, however, that even a person with a good aptitude for programming, entering the field for the first time, needs a year or two of practice before he really acquires reasonable maturity in dealing with complex problems, regardless of the language used.

The greatest danger in the utility concept is that in trying to serve a diverse set of users we reduce the power of a third generation computer to the point where each individual user feels like he is using a first or second generation system with respect to either hardware or software.

Summary on the User

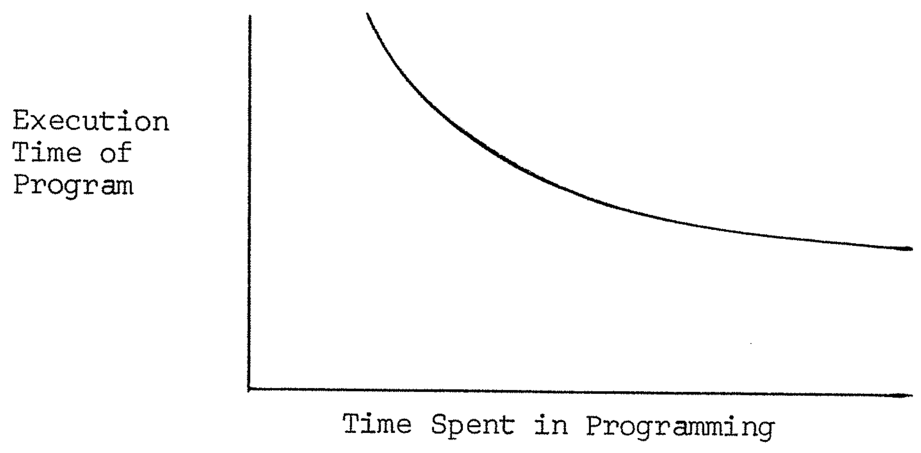
The user could not care less what is at the other end of the wire - a computer or busy little green men. His concern is his problem and his tools are the console and the software language. Any knowledge he is forced to acquire about the other end of the wire is just a waste of his effort as far as he is concerned.

Psychologists can give us very few specifics on the process of creativity. However, there seems to be one opinion universally held. Creativity seems to thrive best in an unrestricted atmosphere. Forced conformity is a damper on the creative process. The measure of a user-oriented time sharing system should, therefore, be how well it conforms to the needs of the user and how little it demands of the user in conforming to it.

I believe these two observations lead one quite naturally to many of the specifics I've attempted to present to you.

The Programmer and Programs

Probably for some time to come the meat of computer usage in the average company will be the production program which will be used a great number of times. History has taught us that a curve similar to the one below exists for any given programmer and any given program of reasonable size for any given computer in any given language.



If the program is only to be run a few times, as are the user-oriented programs we have been talking about, we are really not too concerned about execution time. However, for a production program we should have a great deal of concern.

Now in a batch environment with its forced delays between runs for the programmer, we may unintentionally be forcing the programmer to spend a little more time in thinking about and improving the program. A good programmer who takes pride in his work can usually regulate himself fairly well on how far out on this curve to work. However, I'm not so sure there isn't a significant group of programmers who just want to get the job done as quickly as possible. For the manager who has members of this class in his group, there are rather obviously horrible implications in providing them with an immediate access system. There is also the concern that even a good programmer may tend to become sloppy or "corrupted" by this facility.

It is rather disturbing that none of the time shared systems to day (to my knowledge) have ever examined this in terms of a number of obvious experiments that could be carried out.

It may be wise to require always that a well defined production program be completely programmed at the programmer's desk and the immediate access system only be used for check out or debugging.

Another problem often swept under the rug is the attempt to run either large programs (in terms of memory size) or long programs (in terms of execution time) as a part of a time shared computer.

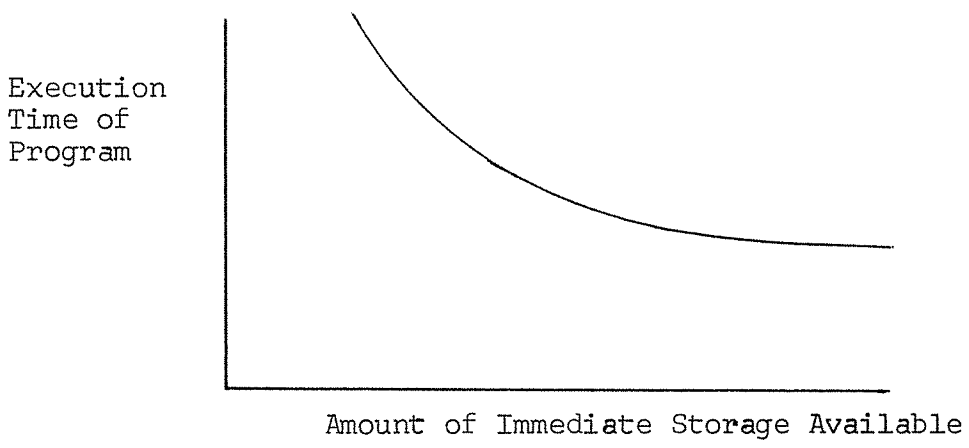
The current estimate on the overhead cost of operating a program in a time sharing system as opposed to a batch system is 10-30%. This is probably true for a program compiled out of a macro language compiler (such as FORTRAN) written for a time sharing environment. However, any compiler originally written for a batch environment will usually produce much larger overheads because the person writing the batch compiler never considered the influence of a time sharing monitor or its interaction with the assembly level programs. (After playing with some of the batch written compilers adapted to a time sharing environment and offered for rental, I feel very strongly this is true but can't really offer any positive proof. In passing, I offer the advice to beware of time shared software not originally written for time sharing systems.)

The same applies to production programs which may be written in assembly level languages by a programmer. If the programmer does not have a working knowledge of how the time sharing monitor works, he can unintentionally write an extremely inefficient program which may work fine in a batch mode but which will be highly inefficient when put in the time sharing system. This implies that whatever was the lowest standard you required from a programmer to write production programs in the batch system, <u>that standard now has to be raised</u> to at least an elementary system knowledge of time sharing operation. This problem of upgrading your programming staff to meet a time sharing environment, I suspect, may be a point to seriously consider for some companies.

Many of the time sharing monitors being implemented today work on what is called a paging scheme where for a large program this means only a portion of the program may be stored in the memory at one time. Now if this large program is getting its current share of computer time and discovers it needs a piece of the program which is out on drum or disk, then the time sharing monitor goes on to give everyone else a share of time while the piece of the large program is brought in from these slower storage units. This means that this one time the large program did not get an equal share of time with the other, perhaps shorter, program. How often this will occur for the large program depends on what might be termed the degree of randomness in its logical structure. There are certain types of programs that can have a very high degree of randomness; good examples are simulations utilizing random events, others are the sorting and merging of data where one likes to keep a significant amount of the data in memory.

For some simulations the claim has been made that a program which takes five minutes when run alone will not just take a 100-minute time when run with an average twenty other users in a time shared environment, but rather 8-1/3 hours; a ratio of 100 to 1 rather than 20 to 1 because of the paging scheme.

We know from experience that in the batch environment there are some types of programs which follow a law of nature represented in the following graph, regardless of time sharing.



I'm afraid there will always be users who will have problems not fit for time sharing and some provisions for them (reverting to first-come, first-served or priorities) should be made.

If one looks carefully, for example, at a place like MIT where the utility concept is fostered greatly and which has a large scale time sharing system, one observes that large and long programs are not too frequently run in this environment and some of the academic departments (e.g., physics) have their own large batch operated computers to handle this type of program. Quoting from Nature, February 11, 1967, "Experience with Project MAC at MIT is said to have shown that roughly a third of the time spent by users of multiple access systems goes on comparatively humdrum activities such as editing and retrieving information." The only time shared system that makes a direct attempt to take on the large scale programs in a time shared environment is the SDC system. Because of the unique character of their hardware, it is extremely difficult to make any cost performance comparisons with commercial installations. Perhaps now that they are converting to a commercial hardware installation, future data will be easier to evaluate.

It took the computer community a while to realize that a small computer feeding a large one was a very efficient setup as compared to one large one all alone. I think we will someday realize that there will be an analogy for time sharing systems where a small or medium computer undertakes to do most of the routine time sharing load such as bookkeeping, compilation, fetching information, and feeds the more difficult tasks or production jobs to a larger computer which operates on a priority as well as a time sharing scheme. On a per dollar spent basis, the smaller computers tend to be more cost effective for these operations anyway. However, there has not been as yet any real effort to look at the netting of computers to share workload and somehow I feel this is the area where the real cost savings and user efficiencies ultimately will be.

Recommendations

My first recommendation, and perhaps the most important, is quite simple:

<u>GO SLOW</u> if possible.

A lot of companies got burned in the attempt to keep up with the Joneses in terms of bigger and faster computers. Considering the publicity time sharing has been getting as the panaceas for all ills, I suspect a little bit of history will repeat itself.

To clarify this, let's follow a logical progression for a company as it moves from a batch environment to a completely time shared one. I might point out that this logical development may, in fact, not be a possible one as it is dependent upon the necessary software being available, which for the smaller company is an issue decided by the manufacturer.

The characteristics of our company are:

1. It has a medium or large computer run in a batch (first priority or occurrence first-served basis) environment.
2. Most of the runs are production type (payroll, inventory, reports, etc.).
3. An average programming staff (a few good, many so-so).

4. The last detailed analysis of the company information flow or structure was at least four to five years ago.

or

5. The computer was always confined to conforming to the structure of information handling that had grown up in the evolution of the company and no real consideration had been given to improve that structure.

The last two suppositions imply that there are some basic differences in handling data (forms, reports, formats, timing, analyses, etc.) which are more or less efficient depending upon whether it is a manual or computer operation, or some compromise of the two. I also have assumed that the "ultimate" of goals for this group is to utilize immediate access to bring "timely" data to individuals in the company for "meaningful" analyses.

There are a number of things this company can do:

1. Set up a "quasi" time shared system with a single small computer used as a terminal interfaced to the larger computer by a disk. By "quasi" here we mean that any program coming into the system will receive about five minutes of main frame computer time in which it is time shared with other programs in the same status or the one program that has been there the longest. After that five minutes are up, if it still has longer to run it goes into the first-come, first-served line and the system reverts to a batch system if no programs are entering it. This eliminates the short program or debugging problem for the programmer and will probably not affect the production run problems too severely. Anyway, there should always be a switch that throws back to a pure batch environment at someone's discretion (another panic button).

2. If the company has a group that is always requesting or would like to request a lot of one-shot numerical analyses (engineers, system analysts), rent an on-line JOSS-like system.

Look for a real non-programmer oriented system (avoid FORTRAN) and reasonable costs (e.g., $250 for 50 hours). This should be billed to the computer operation and individual users should not be charged.

3. Look for a good user-oriented report editing system (especially if the company has a large key punching operation) and rent this too.

The company will now begin to eliminate the production of cards and do a good deal more in the way of versatile error checking as the master tapes are being produced from the on-line terminal.

4. Begin to upgrade the programming staff. Where before you got along with a majority of average programmers, you would now like a majority of junior systems programmers.

5. Start a serious effort to examine the structure of information flow and to analyze its timeliness to the operation of the company.

Now six months or longer have passed and someone from the controller's office bursts into the office of the manager of the computer shop to point out that last month x thousand dollars were spend on time shared services. To placate this upset individual, a notice is sent out limiting time shared usage. This immediately creates a ground swell of unrest which offers strong support for the next and important step.

6. The rented services are eliminated and the x thousand dollars plus a little more are used to replace the small computer with a

medium one and a number of terminals are hooked up to it.

This medium computer must be sufficient to handle the time shared load and provide the necessary software to duplicate the rented services. Also the software should be augmented to allow users to obtain or accumulate information on the disks from the production programs being run through the main computer. The main computer still handles the programmers' debugging of production runs. We now have the ideal situation; we've gone to a time sharing environment without really going through a rewrite of the production programs or computer change. Also we can now provide management with a tool for its attempts to examine the information flowing through the computer system.

Now as time passes the only changes that may take place are the relative sizes of the two computers and the addition of disks. As the flow of information is better understood it will be possible to eliminate more and more of the batch jobs and replace these with a real time gathering and analysis of the information through the use of terminals throughout the company. However, there may be some batch (long run jobs) that can never be eliminated and if these are sufficiently great the batch oriented computer may never be eliminated.

To summarize, time sharing as a tool of management appears to have two important uses:

a) Discovery and analysis by management of what is really going on with information in the company.

b) To implement a timely and meaningful flow of information in the company.

The latter cannot be accomplished without the preceding and the preceding is never completely obtainable. A company and the world it is in changes with time. Perhaps this fact and the realization of time sharing company information systems gives us some hint of what the oft debated role of "middle" management will be someday in the future: To carry out investigations of the company structure and information flow with the goal of providing better tools for "upper" management.

This further implies that a company should carry on a continuing research effort into its own structure much as it carries on research for new products. The recognition of this task as research in the literal meaning of the word is a rather fundamental step in trying to obtain a meaningful company utility whose usefulness will not decay with time.

Concluding Remarks

Let me hypothesize that many of the new computers were well along in design before time sharing was given serious thought. I suspect there have even been some attempts to introduce last minute design changes to accommodate some of the growing demand for immediate access. My conclusion would be that many of todays new generation of computers do not really look like the ideal machines for this sort of utility environment. I would therefore like to speculate what one will see in the fourth generation.

First, single processors will probably have come close to the maximum possible speed and be somewhat smaller and simpler in their instruction set than they are today. A manufacturer will probably market four or five types of processors, all of about the same speed but differing in instruction sets and structure to make them highly efficient in different tasks (e.g., straight numeric computation; gathering and shuffling information; monitoring and controlling communication channels among terminals, memories and processors; compilations and error control). The purchaser would then buy as many of each of these as were needed for his jobs and they would be interconnected to form a multiprocessor. He would probably form a pool with other users whereby they would all be interconnected to distribute peak loads to unused sections of the system. My feeling, therefore, is that time sharing, or the utility concept or even the concept of a real time management information system tied into all the other computer utilization in a company, begins to lead one to a multi-unequal-processor environment. However, with this generation of computers we will still be able to do quite a bit, perhaps not as much or as efficiently as one would like but maybe we will be able to pinpoint in this experience what it is we really need.

In the course of this talk you have probably noted a hint of caution and concern. By this, I in no way mean to imply there are not tremendous benefits that are inherent in the concepts of immediate access and time sharing. However, in the hectic struggle between manufacturers, and even individuals, to remain at the forefront of this rapidly developing field, many loose ends seem to remain dangling to the distress of the average user. Perhaps in the new maturity which seems to be emerging with greater numbers of critical and competent users there will be more emphasis on the little things which, though small individually, become quite bothersome in the multitude of occurrences.

Finally, for those wishing to delve deeper into this morass, a fairly complete bibliography on time sharing may be found in the Proceedings of the IEEE, Volume 54, No. 12, December 1966, pages 1764-65.