IBM Research

Code 455 File Copy

AD-754836

USER BEHAVIOR ON AN INTERACTIVE COMPUTER SYSTEM (1)

Stephen J Boies

August 23, 1972

RC 4169

USER BEHAVIOR ON AN INTERACTIVE COMPUTER SYSTEM

by

Stephen J. Boies

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

The results of an analysis of user behavior on an interactive system are presented. Empirical data on user behavior is presented concerning 1) the duration and frequency of user terminal sessions, 2) the use of language processors, 3) user response time and 4) command usage. The results are discussed in terms of the behavioral literature revelant to the design of interactive systems. Suggestions are made with respect to those areas which should be investigated by behavioral scientists.

RC 4169 (#18013)
August 23, 1972 (Rec'd. 1/2/73)
Life Sciences

Work on this manuscript was done while the author was supported, in part, by a contract from the Office of Naval Research.

Approved for public release and sale; distribution unlimited.

Reproduction in whole or in part is permitted for any purpose of the United States Government.

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication elsewhere and has been issued as a Research Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

Copies may be requested from:
IBM Thomas J. Watson Research Center
Post Office Box 218
Yorktown Heights, New York 10598

Security Classification

DOCUMENT CONTROL DATA – R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
International Business Machines
Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE
User Behavior on an Interactive Computer System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report Interim

5. AUTHOR(S) *(First name, middle initial, last name)*
Stephen J. Boies

6. REPORT DATE
January, 1973

7a. TOTAL NO. OF PAGES
63

7b. NO. OF REFS
17

8a. CONTRACT OR GRANT NO.
N00014-72-C-0419

b. PROJECT NO.
NR-197-020

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
RC 4169

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES
none

12. SPONSORING MILITARY ACTIVITY
Engineering Psychology Programs
Department of the Navy
Office of Naval Research
Arlington, Va. 22217

13 ABSTRACT

The results of an analysis of user behavior on an interactive system are presented. Empirical data on user behavior is presented concerning 1) the duration and frequency of user terminal sessions, 2) the use of language processors, 3) user response time and 4) command usage. The results are discussed in terms of the behavioral literature revelant to the design of interactive systems. Suggestions are made with respect to those areas which should be investigated by behavioral scientists.

DD FORM 1 NOV 65 1473

Security Classification

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT

Security Classification

INTRODUCTION

While it is increasingly recognized by system designers that behavioral data must be considered in the implementation of future systems there is almost a complete lack of behavioral data in a format which can be used by such designers. The current state, however, does not suggest that behavioral data relevant to system design is not available or that the behavioral sciences lack the techniques necessary to develop additional data. Rather, it only indicates that the experimental psychologists and the human factor engineers must organize existing data and develop new data in a format which is most useful to the system designers. This paper is an attempt to show how the methodologies of the behavioral sciences can be used to provide data which is useful to the designers of complex information-processing systems.

There are several techniques developed by the behavioral sciences which will lead to better interactive systems. First, the behavioral sciences are now in a position to develop a quantitative description of how existing systems are being used. This description can lead to an understanding of what features of existing systems should be retained and what should be deleted or changed in

future systems. Second, the behavioral sciences have developed a number of techniques for controlling human behavior in complex information processing tasks. As will be suggested below these techniques might serve to provide solutions to various problems in man-machine systems which are less costly than previously proposed hardware solutions. Third, the behavioral sciences are continually developing an understanding of human behavior in complex tasks. This understanding should be used as the basis for the design of future systems rather than relying on the judgment of the system designer. In addition, behavioral experiments can be designed to address those issues of system design which have not been addressed in the current basic literature.

This paper is a summary of an ongoing project at the T. J. Watson Research Center which has as its goal the development of a basic understanding of the behavioral factors which limit and determine human performance in interactive computer systems. While the project as a whole is using a number of techniques (literature reviews, basic behavioral experiments, development of prototype systems based upon behavioral principles, etc.), this report will concentrate on the results of an observational analysis of user behavior in a complex interactive system, TSS/360.

The paper is divided into three sections. The first provides a brief description of TSS/360 and the methods used in this study. The second section reviews some results concerning user behavior in the systems. Among the issues discussed are; 1) the duration and frequency of user terminal sessions; 2) user response time and its determinants; 3) command usage; and 4) the use of language processors. The final section of the paper suggests how the techniques of the behavioral scientist can be used to gain additional information relevant to system design.

SYSTEM and METHODOLOGY

General Description

The interactive system which served as the basis for this analysis was the TSS/360 system which operates at IBM's T. J. Watson Research Center. Until the recent introduction of the Time Sharing Option, TSS/360 was IBM's major large scale interactive system. It is described in detail in a number of publications and user manuals (IBM 1970, 1971a, 1971b; Thompson, 1971). Briefly, it is a general purpose system which is designed to be used both in program development as well as in interactive applications.

The users interact with TSS/360 through remote terminals. While most terminals are in the user's office or laboratory, a few terminals are in terminal rooms. Most of the terminals are the IBM 2741 Communications Terminal. The keyboard of this terminal is almost exactly like that of the IBM Selectric Typewriter with the only notable exception being that the index key is replaced by an attention key. The maximum typing rate of the terminal is 14.8 characters a second.

The user population of the TSS/360 system at the Research Center is varied. It ranges from the system programmers who are doing system development and maintenance to the users who only know how to sign on to the system and invoke the application program which will lead them through the sequence of steps necessary to perform their tasks. A total of 375 users are permitted to use the system and on an average day between 30 and 45 users are using the system at any one time.

Recording Techniques

The data for this analysis was collected using SIPE (System Internal Performance Evaluator). SIPE is a collection of recording techniques which were developed to

record what aspects of the system were being used and to provide information concerning the failure rate of different system modules (cf. Deniston, 1969). At the Research Center SIPE is being used to record: 1) the line of type that is transmitted; 2) the user identification code; 3) the system and user response times; and 4) the system function which generated the transaction for each line that the system sends to the user and the user sends to the system. If the command system analyzer detects a user-defined procedure during execution, then a record of the primitive commands which were invoked is made. During the user session this information is written out to magnetic tape. Enough information is present on the output tape to completely reconstruct the terminal session for each of the users.

After the user has signed on to the system, the system prompts the user to make a request by unlocking the keyboard on the user's terminal. The user can then type in the command or commands that he wants the system to perform. The system does not start working on the commands until the user depresses a carriage return. At that time the system locks the keyboard so that additional requests cannot be entered and begins working on the task. When the system has completed the task, it again unlocks the keyboard. With

this type of arrangement it is possible to divide the user session into alternating periods of user and system response times. The user response time (URT) is defined as the period of time between when the system prompts the user to enter the next command (by unlocking the keyboard) and when the user depresses the carriage return. The system response time (SRT) is defined as that period of time between when the user depresses the carriage return and when the system unlocks the keyboard for the next request. It should be noted that both the URT and the SRT include typing time. It should also be noted that the SRT is the amount of time which it takes the system to complete the request and not the amount of time which it takes the system to begin working on the request.

The only exceptions to the above cycle are when the user depresses the attention key. If the system is working on a user request when the attention key is depressed the system suspends processing and unlocks the keyboard. If the system is waiting for the user to enter a request when the attention key is depressed, the system ignores anything the user might have typed in up to that point and re-issues the prompt to enter a request.

For purposes of this analysis a user was defined on the

basis of the user identification code in the system. While most individuals have only one identification code and only use their own, there are a few individuals who have more than one code or who use other people's code.

The analysis is based upon the commands which the user issued to TSS/360 or to REDIT, a conversational context editor on the system (Thompson, 1971). Responses to user application programs or responses to prompts for additional information which are occasionally given during the execution of a command were excluded from the analysis.

This study was based upon the terminal sessions which occurred during the working days of the months of September, 1971 and January 1972. There were 21 days in both September and January.

PERFORMANCE ANALYSES

Duration and Frequency of Terminal Sessions

The analysis in this section is based upon the terminal sessions which occurred during the month of September, 1971. The duration of a terminal session was defined as the total time between when the user signed on to the system and when

his terminal was disconnected from the system, regardless of what caused the disconnection.

A total of 182 different users signed on to the system. Figure 1 is a histograph of the number of days each user signed on the the system. As can be seen in the figure, a large number of users signed on to the system only a few days in the month. In fact 36 users, about twenty per cent, signed on to the system on only one day, while over 43 per cent of the users signed on the system on five or less days. A few users, however, used the system on almost a daily basis. Eleven users signed on at least 20 of the 21 days and about 25 per cent of the users signed on 15 days or more. Figure 2 is a histograph of the number of terminal sessions for each user. About 34 percent of the users have five or less terminal sessions during the month while only about 10 per cent of the users had more than 51 terminal sessions during the month. Thus the community of users is made up of a relatively small number of frequent users and a large number of relatively infrequent users.

During the 21 day period there were 3409 terminal sessions or about 162 terminal sessions per day. The total time for all the terminal sessions was about 3712 hours or an average of about 176 hours per day. The upper line in

Figure 3 is a plot of the cumulative per cent of the terminal sessions accounted for by the terminal sessions between 0 and 600 minutes long. As can be seen in the figure there are a large number of very short terminal sessions. Over 20 per cent of the terminal sessions are less than 5 minutes long, over 50 per cent were less than 10 minutes long, and about 75 per cent were less than 75 minutes long. The lower line in Figure 3 is the cumulative per cent of the total connect time accounted for by the terminal sessions between 0 and 600 minutes long. The terminal sessions less than 5 minutes long accounted for less than one per cent of the total connect time. Thus, while there is a large number of relatively short terminal sessions they account for only a small per cent of the total connect time.

The above analysis suggests that a relatively small per cent of the users account for a large per cent of the terminal usage. Figure 4 is the result of, first, rank ordering the users from high to low in terms of the total connect time and then plotting the cumulative per cent of the terminal usage as a function of the cumulative number of users. As can be seen in the figure the 7 per cent of the users who are most active account for 25 per cent of the

total connect time while the 13 per cent of the users who are most active account for over 50 per cent of the connect time. At the other extreme, the fifty per cent of the users who are least active get only 5 per cent of the total connect time. An analysis of the number of terminal sessions, the lower line in figure 4, revealed similar results.

In the above analysis the duration of the terminal session was defined as the period of time between when the user signed on to the system and when he was disconnected. It is, of course, clear that the user might not have been using his terminal all the time he was connected. In fact, it is quite probable that many users leave their terminals for extended periods of time.

An alternative definition of the length of the terminal session which attempts to address the issue of how much the users are actually using the system is to define the length of the user terminal session as the duration of time between when the user signed on to the system and when a significant break in the work occurred. A signficant break in the work includes, of course, when the user's terminal was disconnected from the system. In addition, a signficant break in the work can be defined as when the user fails to

respond to the system for some long period of time. For this analysis ten minutes was selected. If after a signficant break the user started working again, a new terminal session was recorded if he issued more than one command.

Using this definition of user terminal sessions there were 4580 terminal sessions over the 21 day period. A total of 1647 terminal sessions were terminated by a failure to respond for over 10 minutes. As can be expected, the main effect of this definition was to increase the number of terminal sessions which lasted between 25 and 75 minutes long while decreasing the number of terminal sessions which lasted over two hours. The total connect time was reduced from 3713 hours to 3428 hours. This is a reduction of 281 hours or about 16 hours a day. Since this is a more accurate indication of the amount of time the user is actually using the system, it will be used in the analysis below.

Figure 5 is a plot of the total number of hours each of the users were signed on to the system. For base line purposes it should be noted that the system was up about 232 hours or about 11 hours per day. Only one user signed on to the system for more than 100 hours and only 16 users signed

on for more than 60 hours. At the other extreme over half of the users accumulated less than seven hours of connect time during the month. Figure 6 plots the average amount of time which the users used the system on days which he was actually signed on to the system. Thus, this is a plot of the total connect time divided by the number of days the user was signed on to the system. Seventeen per cent of the users were signed on to the system an average of three or more hours on each day that they used the system. Over 50 per cent of the users were signed on for at least an average of 1.25 hours.

The most striking result of this analysis is the high per cent of usage accounted for a relatively small number of users. Looking at the amount of terminal time it was demonstrated that while 13 per cent of the users who were most active accounted for 50 per cent of the terminal time, the 50 per cent of the users who were least active accounted for only 5 per cent of the terminal time. It should be pointed out that this analysis is based upon users who actually used the system during the month. While about 375 users are authorized to use the system only 182 actually signed on. A second aspect of the data is the number of days which the users actually signed on during the month.

Only 25 per cent of the users signed on for 15 or more days. An important question which should be investigated is how the usage of the user who signed on frequently is different from the user who signed on infrequently.

The data collected here bear upon two important issues in system design. First, it is clear that many users are leaving their terminals for relatively long periods of time. During this period of time the only demand that the users place upon the system is holding a port. Since the user often returns only to sign off, it is clear that many users would not be harmed if the system dropped their line after a fixed length period of time had elapsed. Just as most current systems allow the user to specify the amount of CPU time which can elapse before the system deletes the task, perhaps future interactive systems should allow the user to specify the longest keyboard-open time which would be permitted before the system deletes the task. This analysis suggests that such a procedure would reduce the number of ports required to service the same number of users. Second, TSS/360 maintains a user's data sets on-line even when he is not signed on to the system. The fact that many users sign on only occasionally suggests that removing the user's data sets from on-line storage when he is not signed on to the

system is a technique which should be explored as a method of supplementing or replacing existing data migration systems. The use of such a system should permit the same service to a larger number of users with fewer on-line disk packs.

Language Processors

One of the original goals of interactive systems was the facilitation of the production of user programs. TSS/360 has two facilities designed to aid the programmer: the interactive language processors and the program control statements. This part of the analysis will describe how the interactive language processors are being used on TSS/360. Later studies are planned to explore the use (or lack of use) of the program control statements.

There are three fully supported language processors on TSS/360: FORTRAN IV, Assembler, and PL/I. The first two language processors are interactive in two senses. First, the user can write his program in the language processor. In this mode each line of the program is checked for syntax as it is entered. If an error is detected the language processor types an error message and prompts the user to correct the line. Thus the user has immediate feedback

concerning the correctness of each line and a chance to correct it. Second, if the language processor detects an error when it is processing a stored program it will type out the offending line on the user's terminal, indicate the nature of the error, and prompt the user to modify the line. It should be noted that changes are made to the source program as well as being reflected in the output of the language processor. The PL/I language processor is not interactive in either of the above senses.

The purpose of this analysis was to determine how users of TSS/ 360 were using the language processors and to determine if the unique features of such systems were actually being used and were aiding programmer performance.

The first question dealt with how many users were invoking the language processors. In a five day period (1-24-72 to 1-28-72) 39 users invoked one or more of the language processors. During this same time period a total of 114 users signed on to the system. Thus only 34 per cent of the users who signed on to the system were using the language processors. Of the users who invoked a language processors 87 per cent used only one. The PL/I and the FORTRAN language processors were used by an equal number of users (each accounted for 35.90 per cent) and by about twice

as many users as the Assembler language processor (15.39 per cent). The 13 per cent of the users who invoked more than one language processor were almost uniformly distributed among the four possible intersections. As expected, most of these users were system support personnel.

Other questions were asked concerning what was happening when a program was presented to the language processors. Over a five-day period (1-24-72 to 1-28-72) 113 programs were submitted to the FORTRAN language processor. The language processor found an error in only 15.93 per cent of the programs. In the remaining cases either the complete program was processed error free (77.87 per cent) or the user terminated the language processor before an error was detected (6.20 per cent). In the same period 66 programs were presented to the Assembler language processor. In 12.12 per cent of the cases this language processor detected an error, in 25.76 per cent of the cases the user terminated the process before the language processor detected an error, and 62.12 per cent of the cases the complete program was processed without detecting any errors. There were 139 programs presented to the PL/I language processor during this period. Errors were detected in 16.55 per cent, 10.07 per cent were terminated by the user before an error was

detected, and 73.38 per cent were processed completely without an error.

That these results are not unique to the particular circumstances studied is suggested by the results of Moulton and Muller(1969) who found that 66 per cent of student-submitted DITRAN programs compiled correctly. The data presented here are very reliable: evaluation of another five-day period showed essentially the same results. These percentages of syntactically correct programs are considerably higher than the verbal estimates given to us by many computer scientists prior, during and after data collection. Like other important behavioral results, these may seem intuitive after the fact (though of course they were not obvious at all prior to the study).

One might ask what per cent of new programs compiled when first submitted. Although this seems like a sensible question, in fact it is arbitrary if not impossible to define what a "new" program is on interactive system. The critical fact, from the point of view of designing computer systems for people, is that syntactic errors do not seem to be a major bottleneck in computer programming, but occur in only about one in five programs.

It is possible, however, to measure the change which has occurred in the program between successive submissions to the language processor. In the 231 FORTRAN, PL/I and Assembly Language programs that contained no syntactic errors there was an average of 11.87 changes to existing lines and 3.01 lines added between submissions. For the 23 programs that the language processors both completed and found errors in there was an average of 12.12 changes and 31.88 lines added between submissions. Thus there is some support for the assertion that there is a positive relationship between the amount of modification between resubmissions and the probability of detecting a syntactical error.

When the FORTRAN or Assembly language processor in TSS/360 detected an error, a diagnostic message and the offending line were displayed on the terminal and the user was given a prompt to correct the error. In the 26 programs in which this occurred, only once did the user correct the errors in the program. Twice the user corrected some, but not all, of the errors. The most common response was to request the language processor to continue without correcting the error(ten times). Seven times the user terminated the language processor when an error message

occurred. The remaining six times the system crashed or the user session ended while the language processer was waiting for the user to correct the program. These results clearly demonstrate that the users almost never need the interactive error correction features of the language processors and even when they could use them, fail to do so.

Finally, it should be noted that TSS/360 has the facility for checking the syntax of the program as it is typed at the terminal. We found that this facility is almost never used and, when used, the program is usually very short.

There are two types of explanations which can be offered to account for the lack of use of the interactive facilities of the language processors. The first suggests that while the concept of interactive language processors is a good idea, the implementation on TSS/360 is inadequate in one or more respects. The second suggests that interactive language processors are not really a good idea and any implementation would be used about the same.

There are four weakness which are commonly attributed to the TSS/360 implementation. First, many users feel that using the interactive syntax checker to input a program

slows down the system response even when an error is not found in the input line. Most of these users probably feel that since the language processor must check the syntax it will take longer for the system to respond. Actually, however, there is little or no a priori reason to expect that the interactive syntax checker will increase the length of the SRT. When a task is dispatched from a terminal wait it is given a burst of computing resources which is probably sufficient to analyze the syntax of most lines of code. Therefore, if there is a difference in responsiveness it deals only with the difference in the amount of CPU time required to input a line or to input a line and perform the syntactical analysis. However, user behavior is probably more controlled by expectations than reality.

Second, it is generally agreed that the text editing facilities in the language processors are not as powerful or as easy to use as the text-editing facilites found in other sub-systems. This is often used as the basis for suggesting that the users invoke the other editors because it is easier to make the correction. This argument has some face validity to those who do not actually use the system. However, those who have experience with the system know that the actual editing is only a small part of the work that is

required if a different sub-system is used to make the modifications. For example, if the user is going to use another editor to correct an error in a program he must first terminate the language processor, invoke the the text editor, open or load the program, make the necessary changes, file or close the program, terminate the text editor, and re-invoke the language processor. Even if the user had to retype a 100 character line it is hard to see how one can maintain that it is "easier", at least from the standpoint of the whole editing process, to use the other editing facilities.

Third, the language processors on TSS/360 can only be used in the interactive syntax check mode with new programs. Since most programmers spend a great deal of time making relatively small changes to existing programs it is expected that this type of facility would be used little. It has been suggested that if the language processors were changed to work when the user was making changes to old programs they would be used much more. It is, of course, not possible to evaluate this suggestion on the basis of available data. However, it should be pointed out that the implementation of this type of an interactive syntax checker would present several very complex problems, both behavioral

and computational in nature.

Finally, it has been suggested that the interactive features of the language processors on TSS/360 are not used because at one time they did not work reliably. The suggestion here is that the facilities are not used because the users are not sure that they will work. There is considerable informal evidence to suggest that an unreliable aspect of any computer system will take years to live down its bad reputation. However, one would also expect that if the feature were truly helpful to the user he would continue to try to use it. For at least two years prior to this study the interactive aspects of the language processors were as reliable as any other aspect of the system.

There are five arguments which suggest that this type of interactive language processor, regardless of of the strengths and weaknesses of a particular implementation, will not be useful to the programmer. First, the number of programs which have syntactical errors is quite low. Even if all the other objections to the interactive language processors could be met it would be necessary to evaluate their usefulness in terms of their expected frequency of use. Second, the correction of certain types of errors can cause the language processor to re-scan code. Thus while it

may not take any longer to detect an error, the process of interactively correcting a number of errors can greatly increase the amount of computer resources required to process a program. Third, many times the program is entered by someone other than the author. Under this condition the only type of error feedback which would be useful would be the typing errors which might be made. Fourth, embedded within the notion of interactive error correction is the notion that the errors will be "local" in nature. It is possible to define two different classes of syntactical errors. First there are those statements which are invalid (i.e., "X=4.0*Z-T)" is invalid because of unbalanced parentheses). Second, there are those statements which are valid if considered by themselves but which are in conflict with some other aspect of the program(i.e., the statement "100 X(I,J,K)=A" is valid only if statement number 100 is not defined elsewhere and if X is defined as a three dimensional array, etc.). In the first type of error one can see where a line by line syntax analysis with the potential for immediate correction may be of some use. However, the proper correction of the second type of error depends upon understanding the program as a whole. When these types of errors are presented line by line it seems to emphasize the local nature of the error rather than pointing

toward the more general orientation which is necessary to properly correct the program. Fifth, when a programmer is entering code at a terminal he is usually thinking about the "correctness" of the code in terms of whether it is the best way to perform the task that he has to do. It may be rather distracting and annoying to be told that a rather trival syntactical error has just been made. Being told that an error exists and given the opportunity to correct it may very well serve as a disruption to the on-going process of writing the program.

The results of this analysis strongly suggest that the syntactical errors, at least those detected by a language processor on TSS/360, are not a major bottleneck in program development. In addition, the results clearly indicate that the users do not utilize the interactive error correction features of the language processors even when syntactic errors are detected. There are, of course, many possible explanations for the lack of use of this facility. The important finding is, however, that syntactical errors do not represent a major source of delay in the development of programs.

These results strongly suggest that investments in advance techniques for detecting and eliminating syntactical

errors in programs will not lead to payoffs in terms of faster program development. The results do suggest, however, that techniques which permit the user to rapidly make relatively small changes to his program on the basis of information which he gets from attempting to run the program would lead to the reduction in the amount of time which is required to program a given application.

Command Usage

This analysis is based upon the commands issued to TSS/360 and REDIT, the locally implemented context editor. It should be noted that this analysis is based upon the commands which the user issues from the terminal. User or system-defined procedures, therefore, are not broken down into primitive TSS/360 commands. In addition, the responses which the user gives to user programs are eliminated from this analysis. Callaway, Considine and Thompson (1971) have recently described user applications running on TSS/360 at the Research Center which make extensive use of the unique and powerful features of TSS/360 programming environment.

Included in the analysis are the user responses which occurred during the week of January 24, 1972 to January 29, 1972. Tables 1 and 2 present the 20 most-frequently used

TSS/360 and REDIT commands, respectively.

One of the most striking features of the data is the very high per cent of commands which were text-editing commands: over 63 per cent are commands issued to REDIT. In addition, eleven of the most heavily used TSS/360 commands are text-editing commands. Altogether over 75 per cent of the commands issued during this time period were text-editing commands. This points to the extreme importance of having an excellent text-editor on any general purpose interactive system.

The very high use of the text-editing commands and the relative low use of TSS/360 commands, especially programmer-oriented commands such as the Program Control Statements, raises some question as to how TSS/360 is being used. In an attempt to answer this question the command usage for each of the users was examined in an attempt to classify each user as to the type of work he was performing. On the basis of a preliminary analysis four different types of users were defined: programming (defined by the use of the language processors and DDEF commands), netting (defined by the user of commands to ship jobs to the Research Center's 360/91 over a high-speed link), manuscript preparation (defined by the use of RUNOFF and the lower-case

mode in the text editors) and miscellaneous (defined by the absence of the other criteria). Most of the users in the last category issued only a few commands and these commands were of a general nature. Several users, for example, only signed on, read their mail and the news and then signed off. In almost all cases the user could easily be assigned to one of the four categories. The only major exception is that several users who used the netting feature also made relatively frequent use of the RUNOFF facility.

Table III presents the results of a separate analysis of the command usage for each of the four groups. The first aspect of this data is that while 39 per cent of the users were classified as being in the miscellaneous category, they account for only 6.81 per cent of the commands issued. This confirms the results which were based upon the amount of terminal time used and indicated that a relatively large per cent of the users accounted for only a very small per cent of the total system load. The second aspect of the data is the command usage of those users who invoked a language processor. These users issued slightly more TSS/360 commands than REDIT commands and accounted for about 41 per cent of all commands issued. The users involved in manuscript preparation and netting combine to account for

52.32 per cent of the commands issued. Over 80 per cent of the commands issued by these users were text-editing commands. An analysis of the commands which were not text-editing commands indicated a high usage of commands dealing with public storage (i.e., searching for a data set, transferring data sets between migrated and public storage, printing and erasing data sets, etc.). Thus, while these users make little or no use of such TSS/360 features as virtual memory, program control statements, and interactive language processors, they make good use of the excellent context-editing facility and those features unique to TSS/360 which deal with the allocating and use of on-line storage. It is of particular interest to note that very few users who ship to the model 91 use the language processors on TSS/360. This can be interpreted as providing converging support for the generality of the results discussed in the section on language processors. This is based upon the assumption that the user would use the language processors on TSS/360 (FORTRAN and PL/I) as relatively fast turn-around syntax checkers prior to shipping the job to the batch OS machine if they expected a syntax error.

An analysis of all commands issued revealed that there were many cases in which the user would issue a REDIT

command while in the TSS state or a TSS command while in the REDIT state. While a small percentage of these may be the result of errors in the analysis program, most are the results of the user issuing the wrong command for the state that he was in. (Excluded from consideration here are the cases where the user intentionally issued a TSS/360 command while in the REDIT state using the underscore facility). Because of the apparent failure on the part of the user to remember what command state he is in, considerable caution should be used in assigning the same command name for different or even similar functions in different command substates. For example, if the command to leave the REDIT state had the same name as the command to terminating the user session then a number of users would find their terminals disconnected when they thought they were only leaving the edit substate.

This result, of course, does not establish that the same name should not be used for similar or different functions in different substates. Rather it only suggests that if this had been done on TSS/360 it would have lead to the users making a number of rather serious errors. However, if techniques could be implemented where the user did not lose track of the substate that he was in using the

same command name in different subsystems of the command language could lead to faster acquisition of the command language.

There are several interesting aspects of the command usage in REDIT. The first is the high frequency of the FILE command, for which there are probably two reasons. First, many users invoke REDIT to make only a few changes to the program. After the changes have been made the user issues the FILE command to make a permanent copy of the data set on public storage. REDIT currently loads the entire data set into virtual memory when the user is working on a data set. Thus if the user only wants to change one line in a 300 line program, considerable unnecessary burden is placed upon the system in having first to load all the user's program into virtual memory and then to write all the program out on public storage.

In view of the relative high frequency of minor modifications to the program, it would seem advantageous to adopt other procedures for at least some of the loads. Perhaps the user should be given the option of using either the record I/O or virtual memory mode of operation. Using virtual memory to store a program is clearly advantageous if a large number of changes are to be made to an existing data

set. If only a few changes are to be made to an existing data set, however, it would seem that using record-I/O would result in better overall responsiveness. The second reason for the high rate of FILE commands is the fear of system failures. Since REDIT maintains the data set in virtual memory, the modifications which were made since the last FILE are lost if the system crashes. Thus users commonly file several times while making extensive modifications to the data set out of fear of losing work because of a system failure. This is a good example of how the system dynamics can modify the behavior of users in ways which result in a high demand being placed upon the system's resources.

The second aspect of REDIT usage is the techniques used to move between parts of the data set. REDIT has two commands to perform context searches between lines, LOCATE and FIND. Together these commands account for only 6.79 per cent of the commands issued to REDIT. There are six commands used to move the current line pointer on the basis of lines or line numbers associated with the data set (POINT, NEXT, TOP, BOTTOM, UP and NP). These commands account for 26.89 per cent of the commands issued to REDIT. The POINT command alone accounts for eight per cent. This finding suggests that users often have a relatively new

listing of the data set which they are editing. This view is supported by the high use of the TSS/360 PRINT command.

It has been proposed that text-editing systems be designed to function where the user does not have a hard copy of the data set he is working on. The assumption commonly is that if one has a fast local output device, usually a scope, one can reduce or eliminate the need for hard copy. These results indicate that in a typewriter-based system such as TSS/360 the user is clearly relying heavily on hard copy. The reliance is apparently so heavy that it is reasonable to question if a fast local-output device will change the usage pattern. A second and related point is that unless the user has information concerning the structure of the data set (i.e., the line numbers, etc.), context will have to be relied on to perform searches, locate information, etc. An analysis of most text processing applications suggests that the data set, whether it be a manuscript or program, will have too much redundancy to make context searches useful. Searching through almost any data set for a character string, unless the string is very long, will result in a number of false positives. This is especially true if the searches are based upon strings, as they are in almost all existing text editors, rather than

on words or symbols. For example, a search for the variable "I" in a program will stop at occurrences of "WRITE", "IF", "ISCORE", etc., if they occur prior to an occurrance of "I". Likewise, a search in a manuscipt for all the occurrances of the string "THE" will almost certainly locate a number of strings such as "THEN", "THEY", "THERE", etc. Thus even in a system which has a fast local-output device, it seems that context searches will not serve as an adequate replacement for the POINTS, NEXT, etc. which are used in current systems.

There are a number of reasons which combine to make it important to develop an understanding of the behavioral criteria which should be used in designing command languages for interactive systems. First, the command language is the users' major interface with the system. A system can have all the functions desirable but still be useless to the user if he does not know how to invoke the commands. Second, the command languages on many system seem to be a real bottleneck to getting work done. This is illustrated by the large number of users who know only a very few commands, who use only the simplest form of the commands which they use, and who often use a lengthy sequence of commands to accomplish what one command could do. Finally, the format

in which the command langauge is implemented makes little or no difference to the system designer. Hence, behavioral criteria can be used as the basis for selecting the format of the command language without adding to the cost or complexity of the system.

One of the first behavioral questions which should be addressed deals with how many commands should be in a command langauge. Given that the number of functions or things which the user can do with the system are held relatively constant, the system designer could implement a command language where there were a large number of commands but each command had only a few parameters: or the designer could implement a command language where there were a small number of commands but each command had a large number of parameters. From the standpoint of the design and implementation of a system it would make little difference which of these alternatives were used. However, it is reasonable to suggest that the two different implementations might have very different effects upon user performance. Behavioral studies should be undertaken to explore how these two different types of command formats influnece user-performance.

A second issue deals with user-defined commands.

TSS/360 has two different facilities for implementing user defined commands, the PROCDEF and the BUILTIN facilities. The first is oriented toward the user and represents, usually, either a sequence of commonly-used commands grouped into one command or a command which has some of the parameters pre-set to special values. The second facility is oriented toward the system programmer and is based upon assembly language code. It is usually used when the system programmer is implementing a new function which is designed for the the users of a particular system. It seems necessary to evaluate the usefulness of both of these types of facilities for extending the command langauge. It seems particularly important to evaluate the usefulness of the first type of facility in systems which are oriented toward the non-EDP professional. It may very well be that in such systems only the BUILTIN type facility will be needed or used.

A third issue deals with the organization of the command parameters. This will be most important in command languages which have only a few commands with a large number of parameters associated with each command. However, virtually any command language will have parameters associated with at least some of the commands. In current

command langauges there are three basic techniques used to specify command parameters: the positional, keyword, and mixed formats. Basic behavioral work should be undertaken to explore the advantages and disadvantages of the three formats from the standpoint of user performance.

The above series of questions deal with command langauges which are based upon the current state of the art. It is also important for the behavioral scientist to begin to explore alternative types of command languages. There is general agreement that all the problems of a command language will be solved when a command language is implemented in the natural language of the user. There is not, however, general agreement concerning what a natural command language will look like.

There are two different notions with regard to a natural command language. The first general idea is that the command language should have a syntax which is the same as the syntax of the natural language of the user. The system would be able to understand the command as long as it was in a syntactically valid form and the words were in the lexicon. Thus the user could say either "print the data set named paper.sipe.rev on the white line printer with the default line spacing" or "on the white line printer print

paper.sipe.rev using the default line spacing" or any other syntactically valid combination of words.

The second type of analysis suggests that what is important is to make the communication natural rather than having the communication in a natural language. This analysis suggests that the primary fact which characterizes communications between members of a work group is not that most transactions are syntactically valid. If anything, most transactions do not conform to "english teacher syntax". Rather the communication between members of a work group is sucessful because it is based upon a common understanding. There are at least three aspects to this common understanding. First, the members have an understanding of the terms and words which are used. Second, the members have an understanding of the general and specific goals of the group. Third, the members of the group know what is reasonable to do in order to accomplish the goals of the group. This implies that if the computer is to be made an effective member of a work group it must have the common knowledge which is shared by the other members of the group.

This line of reasoning suggests that the computer must, among other things, be capable of correctly inferring what to do on as few clues as possible. For example if the user

issues the command "print paper.sipe.rev,source.sipest" the command system analyzer should be able to figure out that the first file should be printed on the white printer and the second file should be printed on the standard printer.

It is clear that these two different notions concerning what constitues a "natural language command" system have quite different implications for system design. A system which supports a natural language syntax will not necessarily be able to understand what the user is trying to say. Likewise, a system which is designed so that it can make reasonable inferences about what it should be doing may not have to allow input in a wide range of syntactical forms. The need for behavioral work to evaluate the usefulness of these two different approaches to command languages is clear.

User Response Time

Since the introduction of interactive systems designers have looked for information which would provide accurate characterization of the URTs in a system. This information is of interest to the system designer for estimating the amount of load that each user will place upon the system. In addition, since the URT can be seen as measure of user

performance, it is important to understand how various system parameters, such as the system response time SRT, are related to URT. This part of the paper is an attempt to provide a characterization of the URT in TSS/360. It is based upon the user sessions which occurred in the month of September, 1971.

The mean of the user response TIME ~~(URT)~~ was 59.89 seconds, the median was between 9 and 10 seconds, and the mode was four seconds. As indicated by the differences between the mean, median and mode, the URT's form a very skewed distribution. Table IV presents the data regarding the number of user response times which are less than each one-minute interval between 1 and 14 minutes and the number of user responses above 14 minutes. The first line in the table indicates that over 90 per cent of the URTs are less than one minute long and the mean of these responses is 12.61 seconds. As can be seen in the last two lines of the table only about one per cent of the URTs are over 14 minutes. The inclusion of these times in the analysis, however, more than doubles the average URT.

Should have occurred earlier

An analysis of the long (above ten minutes) URTs indicates three things. First, many of these response times are very long (two or three hours). Second, often a long

URT will follow a long SRT. Third, over seventeen per cent of the very long URTs (over 600 minutes) are commands to log off. Many other times the user issues only one or two additional commands and then logs off.

These data indicate that during the time the user is actively interacting with the system he is maintaining a very high rate of responding. This is most clearly indicated by the fact that over 50 per cent of the URTs are less than 10 seconds long. It is also clear that users tend to leave their terminals for relatively long periods of time. The overall mean of the user responses is based therefore upon the combination of two very different distributions and probably has very little value as a characterization of the "average URT".

The relatively large number of fast URTs can be used to support the inference that users go to the terminal with their work planned out relatively well. In addition, it suggests that users can use some of the time while a system is responding to command N to prepare for command N+1.

The realization that there are so many very fast URTs leads to a better understanding of the effects which changes in the SRT will have upon user performance. For example,

suppose that the decision is to have either a 10 or a 20 second SRT. If one uses the mean URT as the base then the average command cycle (URT and SRT) is changed from about 70 to 80 seconds, a relatively small percentage increase. An alternative way to view the increase is that the command cycle will be changed from between 10 and 20 seconds to between 20 and 30 seconds in over fifty per cent of the time. This results in a much larger percentage increase in the command cycle time and gives a much more realistic estimate of the effects of a long SRT on user performance.

There are a number of factors which are related to the length of the URT. First, the length of the URT is related to the type of activity which the user is engaged in. The average (the mean of those responses less than 600 seconds) URT for TSS/360 commands was 32.24 seconds while the average URT for REDIT commands was 19.28 seconds. Second, previous work (Boies and Gould, 1971) has indicated that the URT is related to the complexity of the command. Third, there is evidence from this study that URT is related to the length of the SRT. There is a marked tendency for the length of the URT to command N+1 to increase as the length of the SRT for command N increases. The correlation coefficient between the SRT to command N and the URT to command N+1 is

.837 (Pearson's r). As the SRT increases from 1 to 10 seconds the URT increases, almost monotonically, from about 15 to 24 seconds.

It is not yet clear how these various factors are interrelated. For example, it is unclear how much of the difference between REDIT and TSS commands can be accounted for by the fact that TSS commands tend to be more complex. Likewise, the fact that there is a strong correlation between system and URTs does not establish that a long system response time causes a long URT, although there are a number of lines of converging evidence to suggest that it is reasonable to assert that there is a causal relationship between the two variables. Additional work, both the continuation of the SIPE analysis and basic behavioral experiments, is being planned to develop a better understanding the relationship between these various factors.

DISCUSSION

In the introduction it was suggested that there were three general techniques which were available to the behavioral scientist to use to provide information relevant to the design of computer systems. The purpose of this section is to review those techniques in light of this study.

The first method which was proposed was the observational analysis of existing interactive systems. It was suggested that by knowing what features of current interactive systems were being used a better understanding of the facilities which should be incorporated into future systems could be reached. The behavioral sciences have in the past developed techniques to characterize the performance of individuals in various tasks. This analysis extends the same techniques to complex information processing tasks.

Analysis of the use of language processors provides support for the notion that this type of approach will lead to better systems. The results presented in this paper indicate that the users of TSS/360 seldom need the interactive error-correction features of the language

processor. In addition, in those cases when the user could use the correction features, he seldom does. Thus, it seems reasonable to assert that in designing future interactive systems one could expect that a larger payoff, in terms of programmer productivity, would result from investments in areas other than interactive language processors.

This analysis of use of the language processors, together with the other results presented in this paper, point to the importance of implementing SIPE-type data collection systems. If intelligent decisions are to be made concerning the effectiveness of various interactive systems and their sub-systems adequate behavioral data must be available. The techniques originated in SIPE are suitable for this purpose and should be extended to other systems.

The second procedure which the behavioral scientist has available is the application of behavioral control techniques to the behavioral problems in interactive computer system. The emphasis here is attempting to use what is already known about human performance in complex information processing tasks to improve human performance in computing systems. With this procedure the behavioral scientist searches the basic literature for a solution to a problem and then demonstrates that the solution which worked

in the basic studies also works in the setting of an interactive computing system.

In the present analysis a good demonstration of this technique is the analysis of the relationship found between SRT and URT. Perhaps the best known behavioral "fact" about interactive computer systems is that users do not like a long SRT. The results presented here indicate that a long SRT is related to a long URT. The problem then is to attempt to find a behavioral control technique which can be used to reduce the undesirable effects of a long SRT. It should be noted that various hardware solutions have been offered to this problem. However, most solutions of this type are very expensive. The problem for the behavioral scientist is to attempt to reduce the SRT-URT relationship without adding to the cost or complexity of the system.

In the basic psychological literature a number of studies have explored the relationship of temporal uncertainty (TU) on human performance in speeded information processing tasks. TU is the amount of uncertainty which the subject has with regards to when he will be presented information which he needs for the next response. It can be controlled by having either a long and/or variable inter-trial interval (ITI). The ITI is equivalent to the

SRT. A number of basic studies (Klemmer, 1957; Karlin, 1959;, Nickerson and Burnham, 1969) have demonstrated that as the TU in a task increases the user performance in the task decreases (i.e., it takes him longer and/or he makes more errors). Basic studies (Klemmer, 1957; Posner and Boies, 1971) have also demonstrated that the effects of a long and/or variable ITI on performance can be reduced or eliminated by the introduction of a warning signal. In most tasks the warning signal is a brief auditory or visual stimulus which is presented shortly (usually 500 msec.) before the onset of the stimulus which is to be processed.

It seems reasonable to assert that the introduction of a warning signal just prior to the completion of the SRT would reduce the correlation between the SRT and the URT. It is, of course, difficult for the computer to determine when it is "almost ready". To avoid this it would be possible to have the computer respond "almost ready" when it was in fact ready and to respond "ready" shortly thereafter. While this would increase slightly the SRT (about 500 msec.), it would probably reduce the total command cycle (the URT and the SRT) because of the high postive correlation between the SRT and URT.

It should also be noted that most studies which have

employed high TU in a task report that the subjects dislike the task and get bored and tired very quickly. However, when the subjects participate in a study which has a speeded task with a warning signal, they generally show considerable interest in the task and substain a very high rate of performance over relatively long periods of time. This suggests that the introduction of a warning signal into an interactive system would not only improve user performance but would also improve user acceptance of the system.

It should be pointed out that the introduction of a warning signal into an interactive system could be done for little or no additional cost. For example, one could use as the warning signal several printable characters separated by idle characters. The "cost" of this solution compares very favorably to the cost of providing enough hardware in the system and limiting the number of active users to insure that the SRT never increases beyond some limit.

These suggestions concerning the implications of the results of the basic studies of TU for the design of computer systems will have to be verified by experimental techniques before it is reasonable to incorporate these procedures in the design of computer systems. The behavioral science group at IBM Research is presently

undertaking an effort to accomplish this experimental test.

The third technique which the behavioral scientist has available is the development of a laboratory approach to problems which are limiting human performance in computing systems. This technique would be used when there is a clearly defined behavioral problem in a computing system but existing behavioral control techniques do not provide an adequate solution to the problem. The basic literature, however, will most likely point to the techniques which can be used to develop the experiments necessary to provide the answers.

An example of the need for this technique can be seen in the command usage on the system. The command language of TSS/360 is very rich. Counting the REDIT commands there are well over 300 commands which the user can issue to perform his task. However, the results indicate that only a very small number of commands account for a very large per cent of the total command usage. In addition, one does not have to look very far to discover sequences where the user emitted several commands where one command would have performed the task. This results, of course, in additional burdens being placed upon both the system and the users. It is not surprising that most users do not develop very

sophisticated approaches to their computing problems. If one is limited to about ten verbs (commands) one does not produce very interesting or efficient text of any kind.

Recent work in the field of experimental psychology has been aimed at developing an understanding of how information is stored in long term memory and the factors which influence the success and speed of the retreival processes. It seems reasonable that an effort should be undertaken to explore how the recent results of Meyer(1970) and Atkinson and his associates(Juola and Atkinson, 1971) in this area can be used to improve the number of commands which the user has available.

Boehm, B. W., Seven, M. J., and Watson R. A., Interactive problem solving-An experimental study of "lockout" effects, Proceedings Spring Joint Computer Conference, 1971, 205-210.

Boies, S. J. and Gould, J. D. User performance in an interactive computer system. paper presented at Fifth Annual Princeton Conference On Information Sciences and Systems, 1971

Callaway, P. H., Considine, J. P. and Thompson, C. H., Virtual Memory systems on a scientific environment, IBM Research Report, RC 3654, 1971.

Carbonell, J. R., Elkind, J. I., and Nickerson, R. S., On the Psychological Importance of Time in A Time Sharing System, Human Factors, 1968, 10, 135-142.

Deniston, W. R., SIPE: A TSS/360 Software measurement technique AFIPS Conference Proceedings, Spring Joint Conference, Vol. 34, 1969.

IBM System/360 Timesharing System: Command System User's Guide, Order No. GC28-2001-6, 1971a.

IBM System/360 Time Sharing System: Concepts and Facilities Order No. GC28-2003, 1971b.

IBM System/360 Time Sharing System: Terminal User's Guide, Order No. GC28-2017, 1970

Juola, J. G. and Atkinson, R. C., Memory Scanning for words versus categories, Journal of Verbal Learning and Verbal Behavior, 1971, 10, 522-527.

Karlin, L., Reaction time as a function of time-period duration and variability, Journal of Experimental Psychology, 1959, 58, 85-19.

Klemmer, E. T., Simple reaction time as a function of time uncertainty, Journal of Experimental Psychology, 1957, 54, 195-200.

Meyer, D.E., On the representation and retrieval of stored semantic information. Cognitive Psychology, 1970, 1, 242-300.

Miller, R. B., Response times in man-computer conversational transaction, Proceedings of Fall Joint Computer Conferences, Thompson Book Company, Washington, D.C., 1968, 267-277.

Moulton, P.G. and Muller, M. E., DITRAN-A compiler emphasizing diagnostics, Communications of the ACM, 1967, 10, 45-52.

Nickerson, R. S. and Burnham, D. W., Response times with

non-aging time periods, Journal of Experimental Psychology, 1969, 79, 452-457.

Posner, M. I., and Boies, S. J., Components of Attention, Psychological Review, 1971, 78, 391-408.

Thompson, C. H., "User's Guide to the Research Context Editor", IBM Research Report, Ra-28, 1971.

Footnotes

1. R. N. Ascher performed much of the programming required for this report, J. D. Gould provided the initial guidance for the work, L. A. Miller made a number of useful comments on this paper, and Ms. Rebecca Stage assisted in the manuscript preparation. Work on this manuscript was done while the author was supported, in part, by a contract from the Office of Naval Research.

Table 1. Command frequency and per cent of overall usage for 20 most used TSS/360 commands.

COMMAND	FREQUENCY	PER CENT OF TSS COMMANDS	PER CENT OF ALL COMMANDS
USER ATT	2171	15.25	5.53
REDIT	983	6.90	2.51
LOGOFF	732	5.14	1.87
ERASE	562	3.95	1.43
DDEF	408	2.86	1.04
LIST	379	2.66	.97
PRINT	344	2.42	.88
NETOS	330	2.32	.84
	270	1.90	.69
RMDS	255	1.79	.65
INSERT	249	1.75	.64
DSS?	230	1.62	.59
DISPLAY	216	1.52	.55
NUMB	180	1.26	.46
END	179	1.26	.46
NETOS?	158	1.10	.40
PC?	155	1.09	.40
EDIT	154	1.08	.39
SET	151	1.06	.38
EXCISE	125	.88	.32

Table 2. Command frequency and per cent of overall usage for 20 most used REDIT commands.

COMMAND	FREQUENCY	PER CENT OF REDIT COMMANDS	PER CENT OF ALL COMMANDS
CHANGE	4303	17.22	10.97
POINT	1997	7.99	5.09
PRINT	1959	7.84	4.99
NEXT	1656	6.63	4.22
DELETE	1577	6.31	4.02
INPUT	1511	6.05	3.85
FILE	1397	5.59	3.56
LOCATE	1249	5.00	3.18
NP	1142	4.57	2.91
QUIT	1020	4.08	2.60
UP	843	3.73	2.15
TOP	799	3.20	2.04
LOAD	785	3.14	2.00
FIND	447	1.79	1.14
USER ATT	338	1.35	.86
BOTTOM	331	1.32	.84
OVERLAY	277	1.11	.71
DUPLICAT	226	.90	.58
BP	221	.88	.56
DELETEL	202	.81	.52

Table 3. Per cent and frequency of TSS and REDIT command usage for four different types of users.

TYPE OF USER	NUMBER OF USERS		COMMAND USAGE					
			TSS		REDIT		TOTAL	
	Number	Per Cent	Number	Per Cent	Number	Per Cent	Number	Per Cent
Programming	30	26	8532	53	7566	47	16098	40.87
Manuscript Preparation	16	14	1285	18	5697	82	6982	17.22
Netting	24	21	2697	20	10950	80	13647	34.60
Miscellaneous	44	39	2103	78	581	22	2684	6.81

Table 4. Number, cumulative per cent, and mean user response time for each one minute interval between 1 and 14 minutes and above 14 minutes.

FOR RESPONSES LESS THAN (IN MINUTES)	CUMULATIVE FREQUENCY	CUMULATIVE PER CENT	AVERAGE RESPONSE TIME
1	94127	90.786	12.618
2	98372	94.880	15.638
3	99793	96.251	17.487
4	100580	97.010	18.967
5	101068	97.481	20.163
6	101412	97.812	21.207
7	101674	98.065	22.149
8	101905	98.288	23.119
9	102070	98.447	23.903
10	102192	98.565	24.556
11	102291	98.660	25.142
12	102387	98.753	25.762
13	102476	98.839	26.390
14	102554	98.914	26.985
14 OR MORE	103680	100.000	59.165

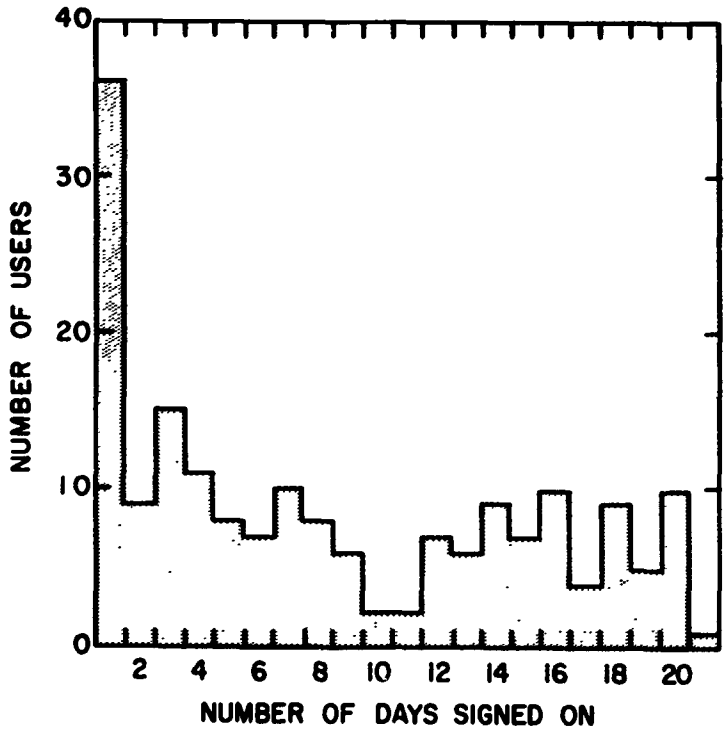

Figure 1. Histograph of the number of days each user signed on to the system during the 21 days of the analysis.

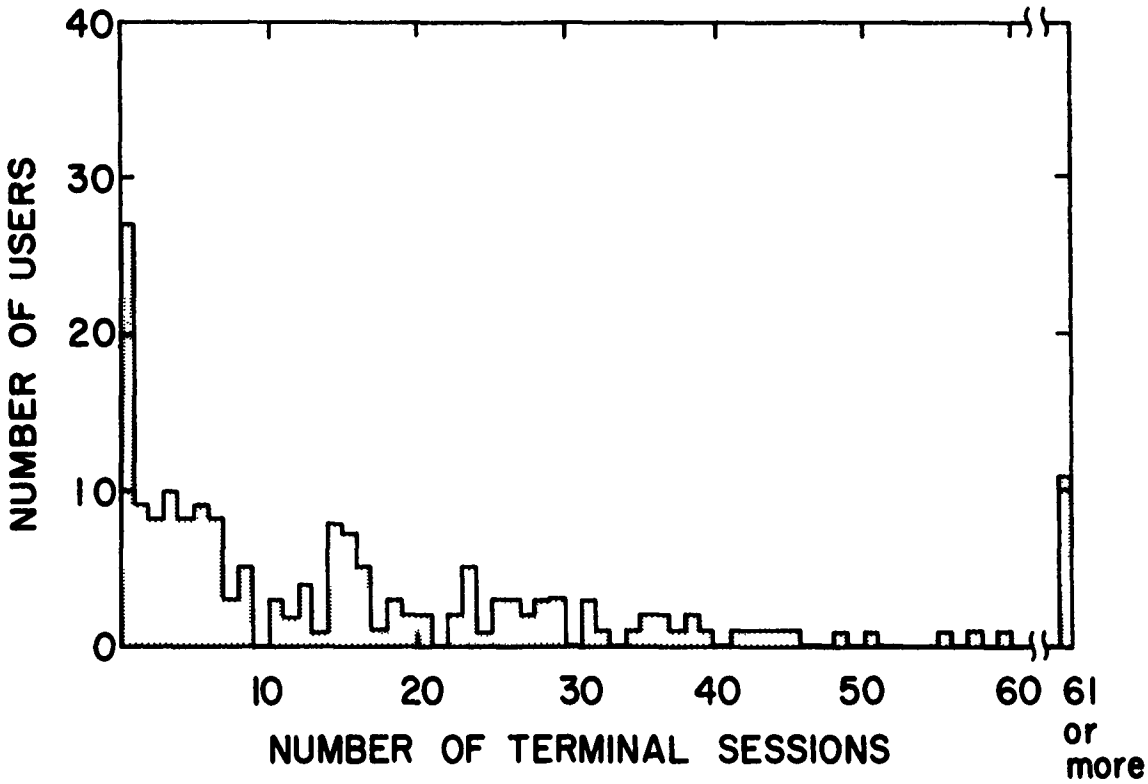

Figure 2. Histograph of the number of terminal sessions for each user during the 21 days of the analysis.

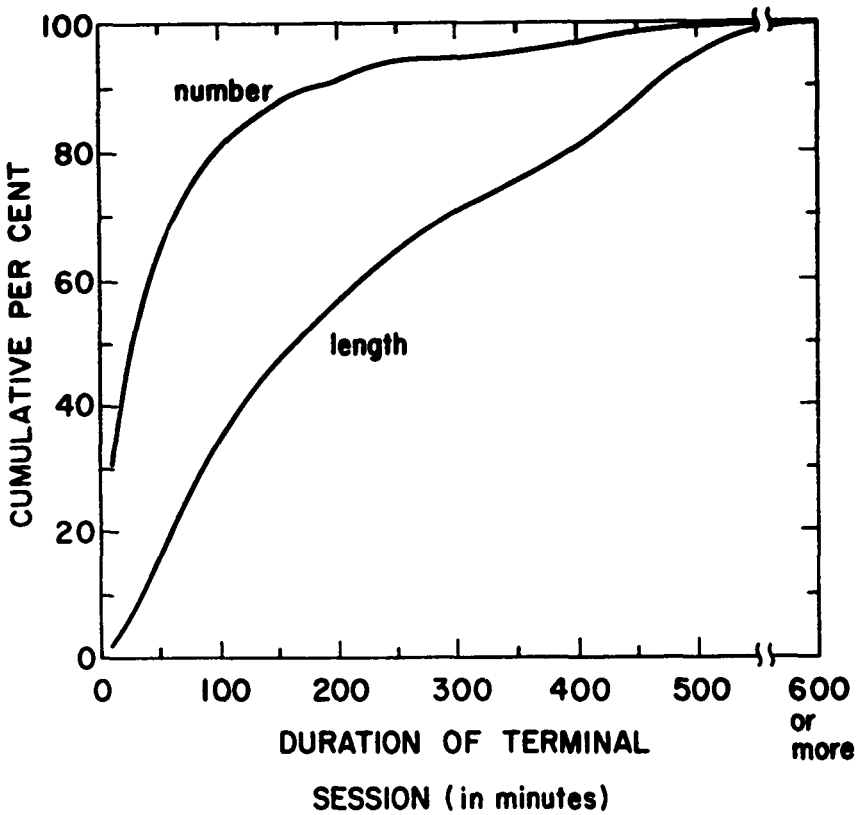

Figure 3. Cumulative per cent of the number and duration of the terminal sessions as a function of the duration of the terminal session.

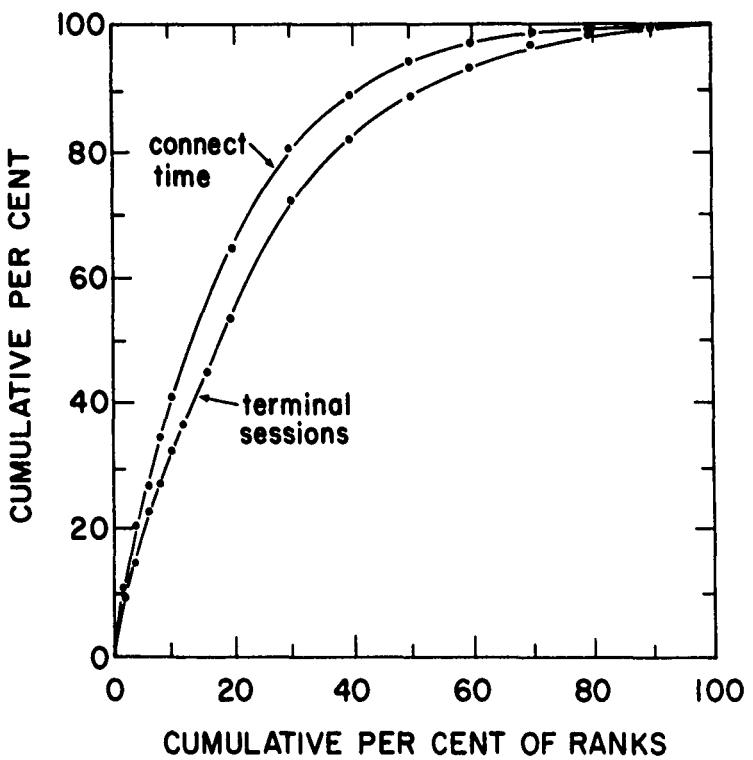

Figure 4. Cumulative per cent of the terminal resources as a function of the users ranked ordered on the amount of terminal activity.

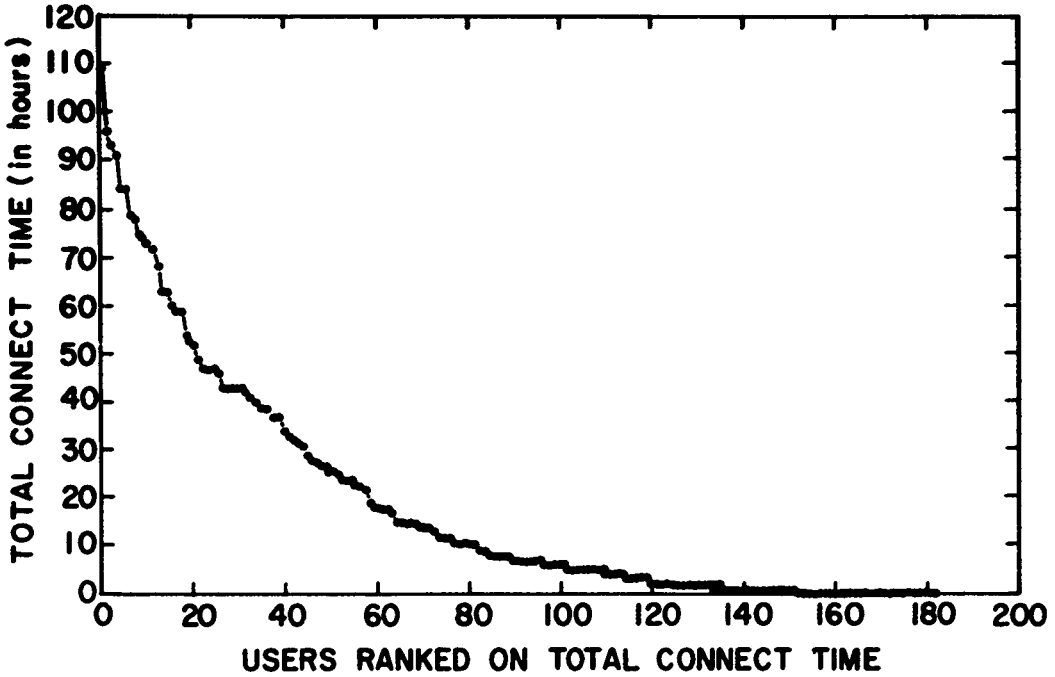

Figure 5. The total number of hours each user signed on to the system during the 21 days of the analysis. The users are ordered on this dimension.

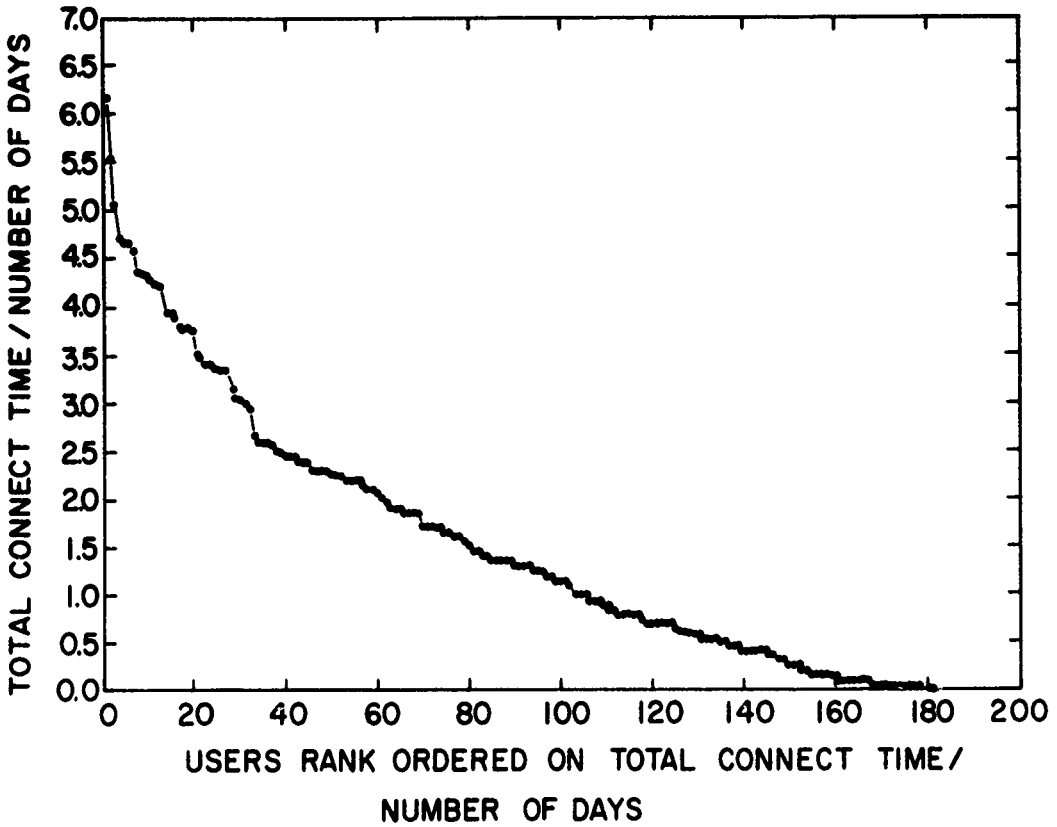

Figure 6. The average number of hours each user was connected to the system on those days he did sign on to the system. The users are ordered on this dimension.

TECHNICAL REPORTS DISTRIBUTION LIST
CODE 455

Director, Engineering Psychology (5 cys)
Programs, Code 455
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

Defense Documentation Center (12 cys)
Cameron Station
Alexandria, Virginia 22314

Director, ONR Branch Office
Attn: Dr. C. Harsh
495 Summer Street
Boston, Massachusetts 02210

Director, ONR Branch Office
Attn: Dr. M. Bertin
536 S. Clark Street
Chicago, Illinois 60605

Director, ONR Branch Office
Attn: Dr. E. Gloye
1030 East Green Street
Pasadena, California 91106

Director, ONR Branch Office
Attn: Mr. R. Lawson
1030 East Green Street
Pasadena, California 91106

Director, Naval Research Laboratory (6 cys)
Technical Information Division
Code 2027
Washington, D. C. 20390

Director, Naval Research Laboratory (6 cys)
Attn: Library Code 2029 (ONRL)
Washington, D. C. 20390

Mr. John Hill
Naval Research Laboratory
Code 5634
Washington, D. C. 20390

Office of Naval Research
Mathematical Sciences Division, Code 430
Department of the Navy
Arlington, Virginia 22217

Office of the Chief of Naval
Operations, Op-095
Department of the Navy
Washington, D. C. 20350

Dr. John J. Collins
Office of the Chief of Naval
Operations, Op-987F
Department of the Navy
Washington, D. C. 20350

CDR H. J. Connery
Office of the Chief of Naval
Operations, Op-987M4
Department of the Navy
Washington, D. C. 20350

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code AX
Washington, D. C. 20380

Dr. Herber G. Moore
Hqs., Naval Material Command
Code 03R4
Department of the Navy
Washington, D. C. 20360

Chief of Naval Material
Prog. Admin. Personnel & Training
NAVMAT 03424
Department of the Navy
Washington, D. C. 20360

Chief of Naval Material
Ocean Engineering & Support Tech. Div.
NAVMAT 034
Department of the Navy
Washington, D. C. 20360

Commander, Naval Air Systems Command
Attn: Mr. Hal Booher, AIR 415G
Washington, D. C. 20360

Commander, Naval Electronics
Systems Command
Command and Display Systems Branch
Code 0544
Washington, D. C. 20360

Mr. Joseph B. Blankenheim
Naval Electronics Systems Command
Code 0474
Washington, D. C. 20360

Commander, Naval Facilities Engineering
Command, Plans & Programs Division
Code 031
Washington, D. C. 20390

Commander, Naval Air Systems Command
NAVAIR 340F
Washington, D. C. 20360

Mr. James Jenkins
Naval Ships Systems Command
Code PMS 302-43
Washington, D. C. 20360

Naval Ships Systems Command
Code 03H
Washington, D. C. 20360

Commander, Naval Supply Systems Command
Logistic Systems Research and
Design Division
Research and Development Branch
Washington, D. C. 20390

Mr. A. Sjoholm
Bureau of Personnel
Personnel Research Div., PERS A-3
Washington, D. C. 20370

CDR Robert Wherry
Human Factors Engineering Systems Ofc.
Naval Air Development Center
Johnsville
Warminster, Pennsylvania 18974

Human Factors Engineering Branch
Code 5342 U. S. Naval Missile Center
Point Mugu, California 93041

Mr. Ronald A. Erickson
Head, Human Factors Branch, Code 4011
Naval Weapons Center
China Lake, California 93555

Human Engineering Branch, Code A624
Naval Ship Research & Development Center
Annapolis Division
Annapolis, Maryland 21402

Dr. Robert French
Naval Undersea Center
San Diego, California 92132

Mr. Richard Coburn
Head, Human Factors Division
Naval Electronics Laboratory Center
San Diego, California 92152

Dr. Gerald Miller
Human Factors Branch
Naval Electronics Laboratory Center
San Diego, California 92152

Dean of Research Administration
Naval Postgraduate School
Monterey, California 93940

Mr. E. Ramras (3 cys)
Technical Director
Personnel Research & Development Lab
Washington Navy Yard
Washington, D. C. 20390

Commanding Officer (3 cys)
Naval Personnel and Training
Research Laboratory
Attn: Technical Director
San Diego, California 92152

Dr. J. J. Regan
Human Factors Department, Code 55
Naval Training Equipment Center
Orlando, Florida 32813

Dr. George Moeller
Head, Human Factors Engineering Branch
Submarine Medical Research Laboratory
Naval Submarine Base
Groton, Connecticut 06340

CDR Thomas Gallagher
Chief, Aerospace Psychology Division
Naval Aerospace Medical Institute
Pensacola, Florida 32512

U.S. Air Force Office of
Scientific Research
Life Sciences Directorate, NL
1400 Wilson Blvd.
Arlington Virginia 22209

Dr. J.M. Christensen
Chief, Human Engineering Division
Aerospace Medical Research Lab
Wright-Patterson AFB, Ohio 45433

Dr. Walter F. Grether
Behavioral Science Laboratory
Aerospace Medical Research Laboratory
Wright-Patterson AFB, Ohio 45433

Dr. J.E. Uhlaner
Director, U.S. Army Behavior &
Systems Research Laboratory
1300 Wilson Blvd.
Arlington, Virginia 22209

Chief of Research and Development
Human Factors Branch
Behavioral Science Division
Department of the Army
Washington, D. C. 20310
Attn: Mr. J. Barber

Army Motivation & Training Lab
Room 239 Commonwealth Bldg.
1300 Wilson Blvd.
Arlington, Virginia 22209

Technical Director
U.s. Army Human Engineering Labs
Aberdeen Proving Ground
Aberdeen, Maryland 21005

Lt Col Austin W. Kibler
Director Behavioral Sciences
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, Virginia 22209

Dr. Stanley Deutsch
Chief, Man-Systems Integration
OART, Hqs., NASA
600 Independence Ave.
Washington, D. C. 20546

Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, Virginia 22202

Mr. Luigi Petrullo
2431 N. Edgewood Street
Arlington, Vitginia 22207

Dr. Ward Edwards
University of Michigan
Dept. of Engineering Psychology
Ann Arbor, Michigan 48105

Human Performance Center
University of Michigan
Ann Arbor, Michigan 48105

Dr. Edwin A. Fleishman
American Institutes for Research
8555 Sixteenth Street
Silver Spring, Maryland 20910

American Institutes for Research
Library
135 N. Bellefield Avenue
Pittsburgh, Pa. 15213

Psychological Abstracts
American Psychological Association
404 East Lancaster Street
Wayne, Pennsylvania 19087

Dr. A. I. Siegel
Applied Psychological Services
404 East Lancaster Street
Wayne, Pennsylvania 19087

Dr. L. J. Fogel
Decision Science, Inc.
4508 Mission Bay Drive
San Diego, California 92112

Dr. Joseph Wulfeck
Dunlap and Associates, Inc.
1454 Cloverfield Blvd.
Santa Monica, California 90404

Dr. Robert R. Mackie
Human Factors Research, Inc.
Santa Barbara Research Park
6780 Cortona Drive
Goleta, California 93017

Dr. Amos Freedy
Perceptroncs, Inc.
17100 Ventura Blvd.
Encino, California 91316

Dr. C. H. Baker
Director, Human Factors Wing
Defense Research Establishment Toronto
P. O. Box 2000
Downsville, Toronto, Ontario
CANADA

Dr. D. E. Broadbent
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, CB2 2EF
ENGLAND

Dr. Cameron Peterson
Decisions and Designs, Inc.
Suite 600, 7900 Westpark Drive
McLean, VA 22101