REPRINTED FROM IBM SYSTEMS JOURNAL, VOL13, NO 1, 1974; © 1974, 1999

Discussed are observations on the usage of an interactive computing system in a research environment. Empirical data on user behavior are discussed that concern the duration and frequency of terminal sessions, the use of language processors, user response time, and command usage.

User behavior on an interactive computer system

by S. J. Boies

The methodologies of the behavioral sciences can provide data useful to systems engineers and designers of complex information processing systems. For example, the quantitative description of the usage of existing systems can provide an understanding of those features that should be retained, deleted, or modified as new systems evolve.

This paper summarizes one aspect of an ongoing project at the IBM T. J. Watson Research Center. The project has as its goal a basic understanding of the behavioral factors that limit and determine human performance in interactive computer systems. Reported here is an observational analysis of user interaction with a complex interactive system, the IBM System/360 Time Sharing System (TSS/360).[1-3] The paper is divided into two sections. The first provides a brief description of TSS/360 and the method used in this study. The balance of the paper reviews our results relative to user behavior. Among the issues discussed are the following: the duration and frequency of terminal sessions; the use of language processors; command usage; and user response time and its determinants.

System and methodology

system The user-system interactions described in this paper refer to TSS/360 and terminals scattered throughout the Research Center. Until the introduction of the Time Sharing Option (TSO), TSS/360 was the major IBM large-scale interactive system. Briefly, TSS/360 is a general-purpose system designed for use both in program development and in interactive applications. Users interact with TSS/360 through remote terminals, most of which are IBM 2741 Communications Terminals in offices and laboratory spaces. The remainder are in common terminal rooms. The terminal keyboard resembles that of the IBM Selectric® Typewriter, with the exception that the index key is replaced by an attention key. The maximum typing rate of the terminal is 14.8 characters a second.

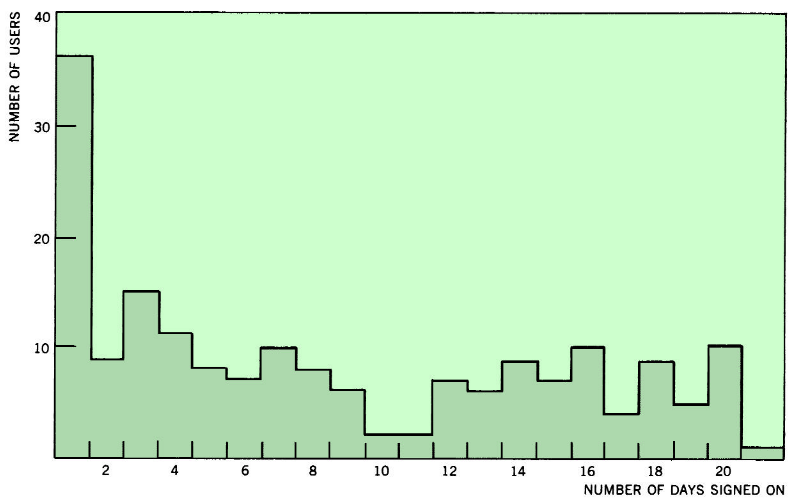
Training levels of the population using the TSS/360 system reported here range from system programmers, who are doing system development and maintenance, to users who simply sign on and call application programs that lead them through sequences of steps necessary to perform their tasks. A total of 375 persons were authorized to use the system via an identification code. On an average day, between 30 and 45 persons use the system at any given time.

recording technique

The data for this analysis were collected by the System Internal Performance Evaluator (SIPE)[4], which is a collection of programs for recording the use of aspects of the system and for and providing information concerning the failure rate of system modules. SIPE records the following items: (1) line of type that is transmitted; (2) user identification code; (3) system and user response times; and (4) the system function that generates the transaction for each line that the system sends to the user and that the user sends to the system. When the command system analyzer detects a user-defined procedure during execution, a record is made of the primitive commands that were called. This information is written out to magnetic tape during the user session. Hence, there is sufficient information present on the output tape to reconstruct each user's terminal session completely.

After the user has signed on to the system, the system prompts the user to enter his 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 executing 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* is defined as the time between the system's prompting the user to enter the next command (by unlocking the keyboard) and the user's depressing the carriage return. The *system response time* is defined as the time between the user's depressing the carriage return and

Figure 1 Interactive system usage during a twenty-one day period



the system's unlocking the keyboard for the next request. It should be noted that both the user response time and the system response time include typing time. Also, the system response time is the elapsed time required for the system to complete its action on the request and not the time it takes the system to begin working on the request.

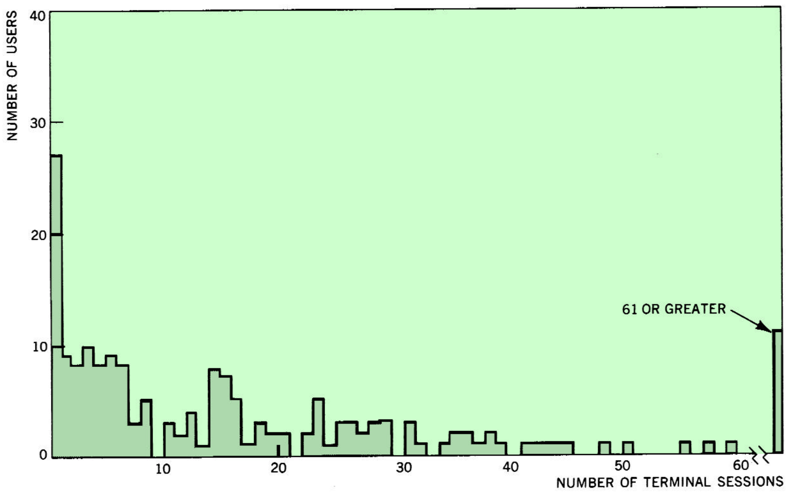
If the system is working on a user request when the attention key is depressed, the system suspends processing and unlocks the keyboard. If the system is waiting for the user to enter a request when the attention key is depressed, the system ignores anything the user might have typed in up to that point and reissues the signal for the user to enter a request.

In our methodology, a user is defined on the basis of an identification number in the system. Although most individuals have only one identification number each and only use their own, there are a few individuals each of whom has more than one number or who use numbers that belong to others.

Our analysis of the methodology is based on the commands that users issue to TSS/360 or to REDIT,[5] a conversational context editor. Responses to user application programs, or responses to signals from the system for additional information that are occasionally given during the execution of a command, are excluded.

This study is based on terminal sessions during the twenty-one working days of each of the months of September, 1971 and January 1972.

Figure 2 Number of terminal sessions of users during a twenty-one day period



Terminal usage patterns

definition of terminal session

The analysis in this section is based on terminal sessions that occurred during the month of September, 1971. The *duration of a terminal session* is defined as the total time between the user's signing on the system and disconnecting the terminal from the system, regardless of the cause of the disconnection.

system user frequency

The community of 182 users consists of a few frequent users and many infrequent users. Figure 1 is a histogram of the number of days on which users signed on the system. The usage pattern indicates that 36 users, or about 20 percent of the population, signed on the system on one day, and that more than 43 percent of the users signed on the system on five or fewer days. A few users, however, signed on the system almost daily. Eleven people signed on the system at least 20 days of the month, and about 25 percent of the user population signed on 15 days or more.

terminal session frequency

Figure 2 is a histogram of the number of terminal sessions per user. About 34 percent of the users had five or fewer terminal sessions each during the month. Only about 10 percent of the users had more than 51 terminal sessions each per month.

During the 21-day period, there were 3409 terminal sessions, or about 162 terminal sessions was day. The total time for all the terminal sessions was about 3712 hours, or an average of about 176 hours per day. The solid line in Figure 3 is a plot of the cumulative percentage of terminal sessions accounted for in the

Figure 3 Cumulative percentage of number and connection time of terminal sessions

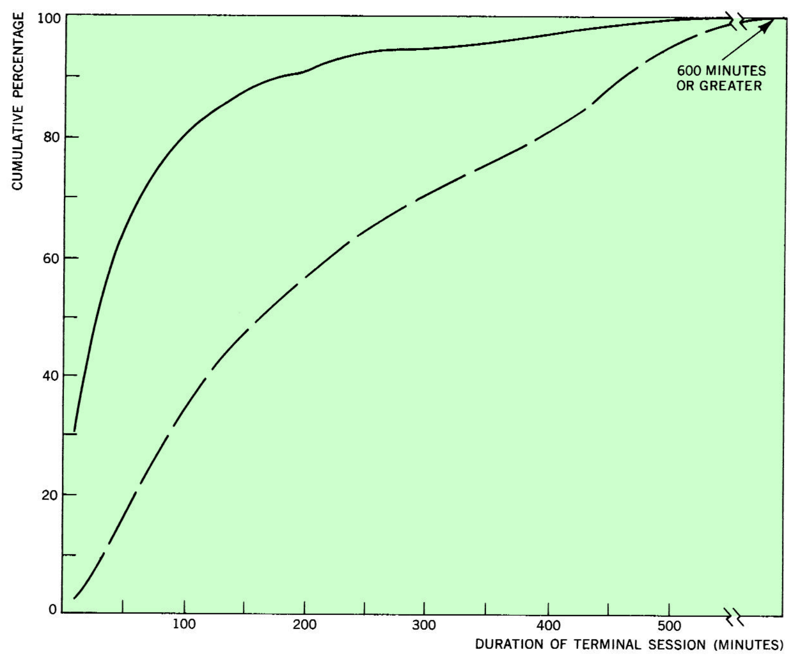
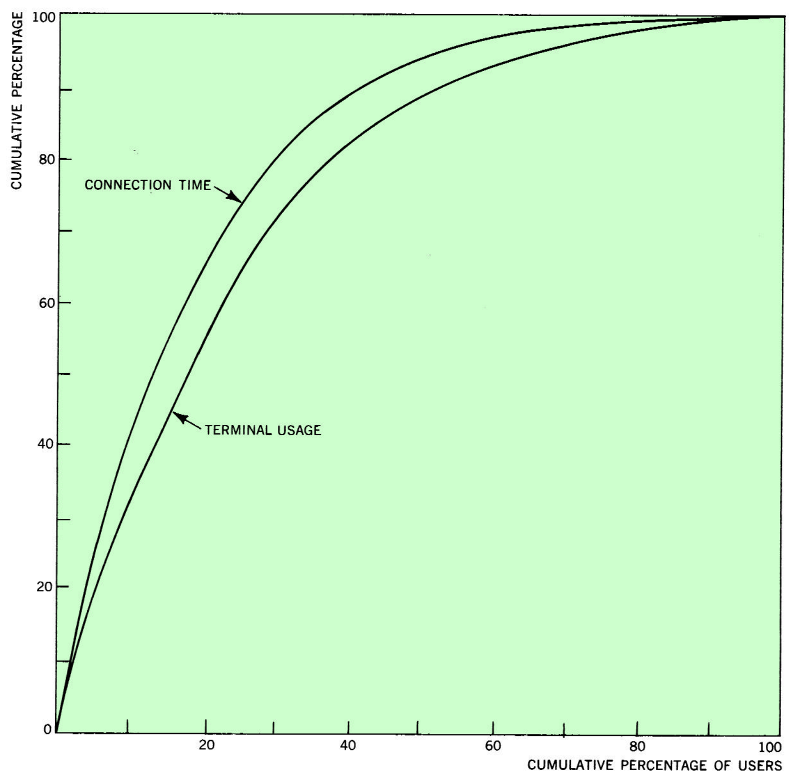


Figure 4 Users ranked by connection time and terminal usage



range of sessions from 0 to 600 minutes duration. This plot shows that there is a large number of brief terminal sessions, which may be summarized as follows:

- More than 20 percent less than 5 minutes
- More than 50 percent less than 10 minutes
- About 75 percent less than 75 minutes

sessions and terminal usage

A large number of relatively short terminal sessions account for only a small percentage of the total connection time. The dashed line in Figure 3 is the cumulative percentage of the total connection time accounted for by the terminal sessions ranging between 0 and 600 minutes duration. Although 20 percent of the terminal sessions are less than five minutes in duration these short terminal sessions account for less than one percent of the total connect time.

users and terminal usage

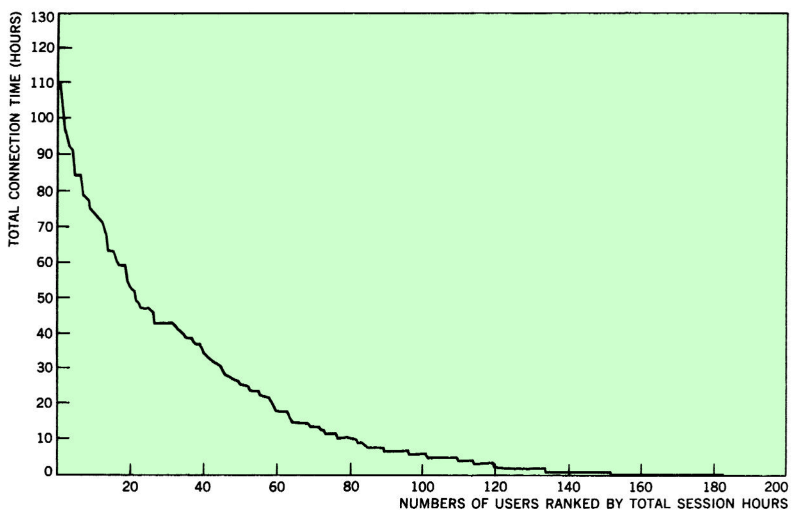
A relatively small percentage of the users account for a large percentage of the total terminal usage. Figure 4 shows the cumulative percentages of total connection time and terminal usage, both as a function of the cumulative percentage of users. The figure indicates that the most active 7 percent of users account for 25 percent of the total connect time, and that the most active 13 percent of the users account for more than 50 percent of the connection time. At the other extreme, the least active 50 percent of the users require only 5 percent of the total connection time. The cumulative percentage of terminal sessions as a function of cumulative percentage of connection time reveals a similar pattern.

modified definition of terminal session

Under the previously given definition of terminal session duration, one can imagine that the user might not have been at the terminal all the time he was connected. We now, therefore, consider the terminal usage pattern under an alternative definition of the length of the terminal session. Here we attempt to embody system usage in the definition as follows: a terminal session is the duration of time from sign-on to a significant break in the work, which also includes disconnection. For purposes of this analysis, we have defined a 10-minute interruption as the threshold of a significant break in the session. Recommencing work after such a break is counted as a new terminal session, provided more than one command is issued following the break.

Under the new definition of terminal session, 4580 terminal sessions were tabulated over the 21-day period. A total of 1647 terminal sessions were terminated by a failure to respond for over 10 minutes. The main effect of this definition has been to increase the number of terminal sessions that last between 25 and 75 minutes and to decrease the number of terminal sessions

Figure 5 Number of users ranked by total connection time



that last more than two hours. The total connection time was reduced from 3713 hours to 3428 hours, a reduction of 281 hours or about 16 hours per day.

session hours accumulated by number of users

Since the 10-minute-lapse criterion gives a more accurate indication of the amount of time a user is actually using the system, it is used in the following analysis. Figure 5 is a plot of the total number of session hours accumulated by numbers of users who signed on the system. For base-line purposes, the system was up about 232 hours or about 11 hours per day. Only one person used the system for more than 100 hours, and only 16 people used the system for more than 60 hours each. At the other extreme, more than half of the users accumulated less than 7 hours each of connection time during the month.

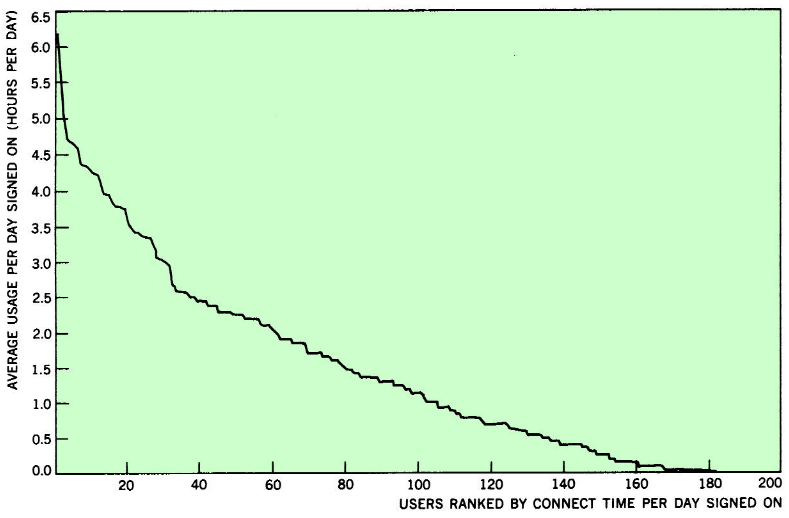
session hour per user per day

Figure 6 is a plot of the average amount of time users used the system on days they signed on the system. In other words, this is a plot of the total connection time per user divided by the number of days users signed on the system. Seventeen percent of the users signed on the system an average of three or more hours on each day they used the system. More than 50 percent of the users signed on for at least an average of 1.25 hours.

high usage by few users

Again, the most striking pattern revealed here is the high percentage of use accumulated by a relatively small number of users. The amount of terminal time shows that 13 percent of the users accounted for 50 percent of the terminal time, whereas the

Figure 6 Users ranked by connection time per day signed on



least active 50 percent of the users accumulated only 5 percent of the terminal time. This observation is based on data for users who actually used the system during the month. Although about 375 users were authorized to use the system, only 182 actually signed on. A second aspect of the data is the number of days that users actually signed on during the month. Only 25 percent of the users were active on 15 or more days. A question for investigation is how the usage of users who sign on frequently differs from that of users who sign on infrequently.

observations

The data indicated two key factors regarding the user population studied. Many users leave their terminals for relatively long periods of time, during which their only demand is that of holding a keyboard-open port into the system. Since a user may return only to sign off, he would not have been harmed if the system had dropped his line after an established period of disuse. Just as current systems allow the user to specify the amount of CPU time that can elapse before the system deletes a task, perhaps interactive systems should allow the user to specify the longest permissible keyboard-open time before a task is deleted. The data suggest that such a capability could reduce the number of ports required to service a given number of users.

The second observation is that TSS/360 maintains data sets online even though the users are not signed on the system. The fact that many users sign on only occasionally suggests that removing users' data sets from on-line storage when users are not signed on the system is a technique that could be explored as a method of supplementing existing data archival systems. Such a technique seems likely to allow equivalent service to more users with fewer on-line disk packs.

Language processors

One goal of interactive systems is that of simplifying the production of application programs. TSS/360 has two facilities designed to aid the programmer: (1) interactive language processors, and (2) program control statements. This section discusses our findings of the way interactive language processors on TSS/360 were used and whether the special features of such systems do, in fact, aid programmer performance.

TSS/360 supports three language processors: FORTRAN IV, Assembler Language, and PL/1. Both the FORTRAN IV and Assembler Language processors are interactive in two senses. The user enters a program in the language of the language processor, and the processor performs a line-by-line check of the program for syntax errors. 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. Also, if the language processor detects an error when it is processing a stored program, the processor types out the offending line on the terminal to indicate the nature of the error, and prompts the user to modify the line. In this way, changes are made to the source program as well as being reflected in the output of the language processor. The PL/1 language processor is not interactive in either of the above senses.

distribution of language processor users

The first question in the language processor study deals with numbers of language processor users. During a five-day study period, one or more of the language processors was invoked by each of 39 users, out of a total of 114 who signed on the system. That is, 34 percent of the users signing on the system used language processors. Of the users of language processors, 87 percent used only one of the two interactive processors. The PL/1 and the FORTRAN IV language processors were equally used, each accounting for 35.90 percent of the total. Thus PL/1 and FORTRAN IV account for about twice the usage as the Assembler Language processor (15.39 percent). The 13 percent of the users who invoked more than one language processor were almost uniformly distributed among the four possible combinations. As expected, most of these users were system support personnel.

FORTRAN IV error detection

Studied also were error detection and reporting experience when programs were presented to the three language processors. Over a five-day study period, 113 programs were submitted to the FORTRAN IV language processor. The language processor detected and reported errors in 15.93 percent of the programs. In the remaining cases, either the complete program was processed error free (77.87 percent), or the user terminated the language processor before an error was detected (6.20 percent).

Assembler Language error detection

During the same period, 66 programs were presented to the Assembler Language processor. In 12.12 percent of the cases, the Assembler Language processor detected an error; in 25.76 percent of the cases, the user terminated the process before the language processor detected an error; and in 62.12 percent of the cases, the complete program was processed without detecting any errors.

PL/1 error detection

Similarly, 139 programs were presented to the PL/1 language processor during this period. Errors were detected in 16.55 percent of the programs; 10.07 percent of the programs were terminated by their users before an error was detected; and 73.38 percent of the programs were processed completely without an error.

These data were found to be repeatable over another five-day period, wherein essentially the same results were obtained. The percentages of syntactically correct programs that we have obtained are consistently higher than the verbal estimates given to us by many computer scientists prior, during, and after data collection. Like results of other behavioral studies, the error-detection findings may seem intuitive after the fact, though, of course, they were not obvious at all prior to the study. That these results are not unique to the particular circumstances studied is suggested by the results of Moulton and Muller[6], who found that 66 percent of student-submitted DITRAN programs were able to be compiled correctly.

modification, resubmission, and syntactic errors

One might also inquire into the percentage of new programs that are compiled correctly as first submitted. Although this seems like a sensible question, it is, in fact, an arbitrary if not an impossible thing to define a "new" program on an interactive system. The critical fact, from the point of view of human factors in systems engineering, is that syntactic errors are not a major bottleneck for computer programmers, since syntactic errors occur in only about 20 percent of the programs.

It is possible, however, to measure changes that occur in programs between successive submissions to a language processor. In the 231 FORTRAN IV, PL/1, and Assembler Language programs that contained no syntactic errors, an average of 11.87 changes occurred to existing lines and an average of 3.01 lines were 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 syntactic error.

user error correction observations

The system being studied was so designed that when the FORTRAN IV or the Assembler Language processor in TSS/360 detected an error, a diagnostic message and the offending line were displayed on the terminal, and the user was prompted to correct the error. Of the 26 programs in which this event occurred, in only one case did the user correct all of the errors in his program; twice he corrected some but not all the errors. The most common response was to request the language processor to continue without correcting the error (10 times). In seven cases, the user terminated the language processor when an error message appeared. In the remaining six cases, the system stopped or the session ended while the language processor was waiting for the user to correct the program.

These results demonstrate that users seldom need the interactive error correction features of language processors, because, even when they can use them, they fail to do so. Although TSS/360 has the facility for checking program syntax as it is typed at the terminal, we found that this facility is seldom used, and then it is used only when the program is short.

Results of this analysis suggest that syntactical errors detected by a language processor or TSS/360 are not a major bottleneck in program development. Results indicate that programmers do not use 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. Our observations suggest that techniques that permit the user to make small changes to his program on the basis of information signaled during the program run might reduce the time required to program a given application.

Command usage

We now discuss our observations on user commands issued from terminals to TSS/360 and REDIT. The latter is an editing system that has been implemented at the IBM Research Center. User or system-defined procedures, therefore, are not broken down into primitive TSS/360 commands. User responses to an application program are eliminated from this analysis. Callaway, Considine and Thompson[7] have recently described application programs running under TSS/360 that make extensive use of the TSS/360 programming and environment.

text editing commands

This study includes user responses that occurred during the January 24, 1972 to January 29, 1972 observation period. Tabulations of the 20 most frequently used TSS/360 and REDIT commands reveal that more than 75 percent of the commands issued during the study period were text-editing commands. More than 63 percent of the total were text editing commands issued to REDIT, and 11 of the most heavily used TSS/360 commands were text editing commands. This high usage suggests the importance of having a smoothly working text editor on any general-purpose interactive system.

TSS/360 command usage

The high usage of the text editing commands and the relatively low usage of TSS/360 commands—especially programmer-oriented commands, such as the program control statements—raise the question of how TSS/360 is being used. To answer this question, the command usage for each of the users was examined so as to classify each user as to the type of work he was performing. On the basis of a preliminary analysis, the following four different types of usage were defined: (1) programming (as indicated by the use of the language processors and data definition commands); (2) netting (indicated by commands to ship jobs to the System/360 Model 91); (3) manuscript preparation (indicated by the use of RUNOFF and the lower case mode in the text editors); and (4) miscellaneous. Most of the users in the miscellaneous category issued only a few commands, and these commands were of a general nature. Several such users simply signed on, processed no data, and then signed off. In almost all cases, the user could easily be assigned to one of the four categories. The only major exception was that several users of the netting feature also made relatively frequent use of the RUNOFF facility.

A tabulation of the results of a separate analysis of the command usage for each of the four groups indicates that the 39 percent of the users classified in the miscellaneous category account for only 6.81 percent of the commands issued. This confirms results based on the amount of terminal time used, and indicates that a relatively large percentage of the users accounts for only a very small percentage of the total system load. A second finding is that in the command usage of those users who invoked a language processor, slightly more TSS/360 commands than REDIT commands were issued. The TSS/360 commands accounted for about 41 percent of all commands issued.

manuscript and netting

Users involved in manuscript preparation and computer netting combined to account for 52.32 percent of the commands issued.

Over 80 percent of the commands issued by these users were text editing commands. An analysis of the commands that are not text editing commands indicates a high usage of commands dealing with public storage (e.g., searching for data sets, transferring data sets between archival and public storage and printing and erasing data sets). Although manuscript and netting users made little or no use of such TSS/360 features as virtual storage, program control statements, and interactive language processors, they made good use of the excellent context editing facility and of those features of TSS/360 that deal with the allocation and use of on-line storage. Few netting users who shipped information to the System/360 Model 91 used the language processors on TSS/360. This can be interpreted as providing support for the generality of the results discussed in the earlier section on language processors. Our interpretation is based on the assumption that one uses the language processors on TSS/360 FORTRAN IV, and PL/I as relatively fast-turnaround syntax checkers prior to shipping jobs to the batch OS/360 machine if he expects a syntax error.

An analysis of all commands issued reveals that there were many cases in which a user would issue a REDIT command while operating in the TSS/360 state, or issue a TSS/360 command while operating in the REDIT state. Although a small percentage of these anomalies may be the result of errors in the analysis program, most of them are the result of a user issuing the wrong command for his operational state. (Excluded from consideration here are cases in which users intentionally issued TSS/360 commands while in the REDIT state via the underscore facility). Because of the apparent failure of users to remember their command states, caution should be used in assigning the same command name for different or even similar functions in different command substates.

REDIT

There are several interesting aspects of the command usage in REDIT. The high frequency of the FILE command may result from many users invoking REDIT to make only a few changes to programs. After the changes have been made, a 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 storage when the user is working on the data set. Thus if the user only wants to change one line in a 300 line program, an unnecessary burden is placed upon the system by first having to load the entire program into virtual storage, and then to write the program out on public storage.

Another aspect of REDIT usage is its technique for moving between parts of a data set. REDIT has two commands—LOCATE and FIND—to perform context searches between lines. In our study, these commands account for 6.79 percent of the commands issued to REDIT. 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) account for 26.89 percent of the commands issued to REDIT. The POINT command alone accounts for eight percent. This finding suggests that users often have a relatively new listing of the data set that they are editing, which is supported by the high use of the TSS/360 PRINT command.

observations and behavioral studies of command languages

Based on our studies up to this time, we believe that it is important to develop an understanding of the behavioral criteria that may be useful in designing command languages for interactive systems. The command language is the users' major interface with the system. Yet a system can have all the desirable functions and still be lacking in some respect if the user does not know how to use the commands. Our results seem to indicate that a large number of users know and use only a few commands, or use only the simplest form of the commands. The result is that they often use lengthy sequences of commands to accomplish what a single command could do. Since almost any command language format can be implemented, behavioral criteria can be used as the basis for selecting formats that best suit users' needs and habits. Because virtually any command language has parameters associated with at least some of the commands, basic behavioral work should be undertaken to explore the advantages and disadvantages of positional, keyword, and mixed formats from the standpoint of user performance.

In this paper, we have discussed command languages that are in the current state of the art. We believe that it is also important for the behavioral scientist to begin to explore alternative types of command languages. For example, there is a body of opinion that the problems of a command language will be solved by languages implemented in the natural language of the user. However, there is no general agreement concerning what a natural command language should look like. One often cited proposal is that the command language should have a syntax that is the same as the syntax of the natural language of the user. The system should then be able to understand a command as long as it is a syntactically valid form composed of words in the lexicon.

Another opinion suggests that it is more important to have natural communication than to have communication in a natural language. This is based on the observation that the primary characteristic of communications among members of a work group is not that they are syntactically valid, but rather that they are based upon a common understanding. This implies that if the computer is to be made an effective member of a work group it must have the common knowledge that is shared by members of the group.

User response time

Accurate characterization of user response times (URT) in a system is of interest to system designers for estimating the amount of load that individual users place on a system. Since the URT can be seen as measure of user performance, we must understand how various other system parameters—such as the system response time (SRT)—are related to URT. This part of the paper, which summarizes our findings relative to URT in TSS/360, is based on user sessions that occurred in September, 1971.

The mean user response time was found to be 59.89 seconds, the median was between 9 and 10 seconds, and the mode was 4 seconds. As indicated by the differences among the mean, median, and mode, the URTs form a very skewed distribution. Our tabulations indicate that over 90 percent of the URTs are within one minute, and the mean of these responses is 12.61 seconds. Only about 1 percent of the URTs are over 14 minutes. The inclusion of these times in the analysis, however, more than doubles the average URT.

An analysis of URTs in the range above ten minutes indicates that many of these response times are two or three hours. Often a long URT follows a long SRT. Over 17 percent of the URTs of 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 interacting with the system he is maintaining a very high rate of response. This is most clearly indicated by the fact that over 50 percent 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. We conclude that the overall mean of the user responses is based on the combination of two very different distributions and probably has very little value as a characterization of the average URT.

Using the finding that there are many very fast URTs leads to a better understanding of the effects that changes in the SRT might have on user performance. For example, assume a 10 or a 20 second SRT. If one uses the mean URT as the base, then the average command cycle (URT plus SRT) changes from about 70 to 80 seconds, a relatively small percentage increase. An alternative way to view the increase is that the command cycle changes from between 10 and 20 seconds to between 20 and 30 seconds in more than 50 percent of the cases. 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.

A number of factors are related to the URT length. The length of the URT is related to the type of activity in which the user is engaged. This is indicated by the following two observations: (1) the average (mean of those responses less than 600 seconds) URT for TSS/360 commands was 32.24 seconds; and (2) the average URT for REDIT commands was 19.28 seconds. Previous work[8] has indicated that the URT is related to the complexity of the command, and evidence from this study tends toward the same conclusion. There is also a marked tendency for the length of the URT to increase as the length of the SRT increases. The correlation coefficient between an SRT to a given command and the URT to the next command is 0.837. 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 factors are interrelated. For example, it is unclear how much of the difference between REDIT and TSS/360 commands can be accounted for by the fact that TSS commands tend to be more complex. Similarly, the fact that there is a strong correlation between system response and user response times does not establish that a long system response time causes a long URT. Work is continuing on the SIPE analysis as well as on basic behavioral studies to develop a better understanding of these factors.

Concluding remarks

We have observed programmers at work using existing interactive systems in an IBM research environment. Our observation of the use of language processors indicates that users of TSS/360 seldom need the interactive error-correction features of the language processor. Also, when a user might use such features, he seldom does. By implication, these observations also point to the importance of implementing SIPE-type data collection systems. We have presented our observations of the relationship between system response time (SRT) and user response time (URT). The results indicate that a long SRT is related to a long URT. This suggests that ways be sought to reduce the undesirable effects of a long SRT and to reduce the SRT-URT interaction without adding to the cost or complexity of the system.

The command language of TSS/360 is rich. Counting the REDIT commands, there are well over 300 commands available to the user. Our studies indicate, however, that only a small number of commands account for a large percentage of the total command usage. Also observed is the use of sequences of commands where one command would have performed the task. This habit places unnecessary burdens on both the system and the user.

These observations have developed out of an ongoing project. Many other areas therefore remain to be studied. It seems reasonable, for example, that an effort should be undertaken to explore how recent results of Meyer[9] on the representation and retrieval of semantic information and Juola and Atkinson[10] on human memory scanning can be used to increase the use of commands that the user has available.

ACKNOWLEDGMENT

The author wishes to acknowledge R. N. Ascher, who performed much of the programming for this study; J. D. Gould, who provided the initial guidance; L. A. Miller for his useful critiques; and Rebecca Stage, who 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.

CITED REFERENCES

1. *IBM System/360 Time Sharing System: Terminal User's Guide.* Form No. GC28-2017, IBM Corporation, Data Processing Division, White Plains, New York 10604 (1970).
2. *IBM System/360 Time Sharing System: Command System User's Guide*, Form No. GC28-2001-6, IBM Corporation, Data Processing Division, White Plains, New York 10604 (1971).
3. *IBM System/360 Time Sharing System: Concepts and Facilities*, Form No. GC28-2003, IBM Corporation, Data Processing Division, White Plains, New York 10604 (1971).
4. W. R. Deniston, SIPE: "A TSS/360 software measurement technique," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **34** (1969).
5. C. H. Thompson, *User's Guide to the Research Context Editor*, IBM Research Report Ra-28 (1971).
6. P. G. Moulton and M. E. Muller, "DITRAN-A compiler emphasizing diagnostics," *Communications of the ACM* **10**, 45–52 (1967).
7. P. H. Callaway, J. P. Considine, and C. H. Thompson, "Uses of virtual storage systems in a scientific environment," *IBM Systems Journal* **11**, No. 3, 200–218 (1972).
8. S. J. Boies and J. D. Gould, "User performance in an interactive computer system," *Fifth Annual Princeton Conference on Information Sciences and Systems* (1971).
9. D. E. Meyer, "On the representation and retrieval of stored semantic information," *Cognitive Psychology* **1**, 242–300 (1970).
10. J. G. Juola and R. C. Atkinson, "Memory scanning for words versus categories," *Journal of Verbal Learning and Verbal Behavior* **10**, 522–527 (1971).

Stephen J. Boies

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

Cognitive psychology (Ph.D., University of Oregon, 1971). Joined IBM in 1970 as a Research Staff Member. He investigated behavioral factors that limit and determine human performance in interactive computer systems. He has also investigated human capacities for representing and processing information. He is currently exploring ways that a digital computer can be used to process and store speech.