

AD-A110 460

INTEGRATED SYSTEMS SUPPORT INC FALLS CHURCH VA F/8 9/2  
INTERACTIVE PROGRAMMING: SUMMARY OF AN EVALUATION AND SOME MANA--ETC(U)  
MAR 75 J M REASER, J C CARROW DAAK02-72-D-0529

UNCLASSIFIED

USACSC-AT-74-83

NL

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

1-1

END

DATE

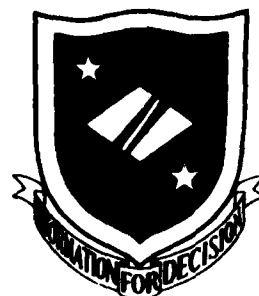
FORMED

3 42

DTIC

**LEVEL III**

**20**



**USACSC-AT-74-03**

**INTERACTIVE PROGRAMMING:  
SUMMARY OF AN EVALUATION  
AND**

**SOME MANAGEMENT CONSIDERATIONS**

Prepared for

**UNITED STATES ARMY**

**COMPUTER SYSTEMS**

**COMMAND**

**DTIC  
ELECTE**

**FEB 3 1982**

**S D**

**DISTRIBUTION STATEMENT A**

Approved for public release  
Distribution Unlimited

**PREPARED BY**

**INTEGRATED SYSTEMS SUPPORT, INC.**

**FALLS CHURCH, VIRGINIA 22041**

**USAMERDC CONTRACT NO. DAAK 02-72-D-0529**

**DA PROJECT NR. SX065003MY10-04A**

**D**

**409048**

**JAP**

**02 02 02 056**

**FORT BELVOIR, VIRGINIA**

**AD A110460**

**ITD FILE COPY**

AD

TECHNICAL DOCUMENTARY REPORT

U.S. ARMY COMPUTER SYSTEMS COMMAND

USACSC-AT-74-03

INTERACTIVE PROGRAMMING: SUMMARY OF AN EVALUATION  
AND SOME MANAGEMENT CONSIDERATIONS

Authors: Joel M. Reaser and John C. Carrow

March 1975

Prepared for  
U.S. ARMY COMPUTER SYSTEMS COMMAND  
FORT BELVOIR, VIRGINIA 22060

Prepared by  
INTEGRATED SYSTEMS SUPPORT, INC.  
Falls Church, Virginia 22041  
USAMERDC Contract No. DAAK 02-72-D-0529

DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited.

DA Project Nr. SX865803MY10-04A

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



DTIC  
ELECTE  
S FEB 3 1982 D  
D

#### ABSTRACT

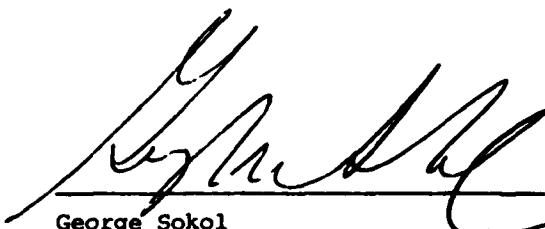
This report summarizes the results of an evaluation of interactive programming versus batch programming within an actual software production environment at the U.S. Army Computer Systems Command. The purpose of the study was to determine productivity and cost effectiveness differences between the two modes of operation. The results of the study indicate that, on a line-of-code basis, the interactive system offers an increase in productivity and a decrease in overall cost.

Based on the data gathered formally and informally from the programmers, topics are discussed which should be considered in management planning for interactive programming.

FOREWORD

This report was prepared in support of the U.S. Army Computer Systems Command Research and Development Task IVA, Interactive Programming. The report was prepared by Integrated Systems Support, Incorporated and the Human Resources Research Organization (HumRRO) under subcontract to ISSI under Contract No. DAAK02-72-D-0529.

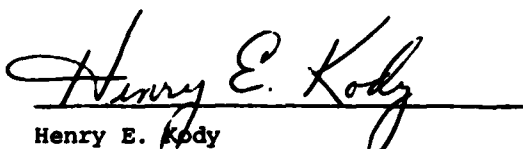
This Technical Report has been reviewed and is approved.



George Sokol  
Deputy For Engineering  
U.S. Army Computer Systems Command



Giora Pelled  
Director, Advanced Technology  
U.S. Army Computer Systems Command



Henry E. Kody  
Chief, Technology Transfer Division  
U.S. Army Computer Systems Command



John C. Carrow  
Captain, U.S. Army  
Project Officer

TABLE OF CONTENTS

<u>CHAPTER</u>		<u>PARAGRAPH</u>	<u>PAGE</u>
1	INTRODUCTION		1-1
	The Problem	1	1-1
	Background	2	1-3
2	PLANNING AND PREPARATION		2-1
	Installing the System	1	2-1
	Programmer Preparation	2	2-1
	Other Preparations	3	2-2
	The Evaluation Plan	4	2-2
3	RESULTS OF THE EVALUATION		3-1
	Productivity Comparisons	1	3-1
	Cost Effectiveness Comparisons	2	3-2
	Other Findings and Lessons Learned	3	3-3
	The Study in Retrospect	4	3-7
4	IMPLICATIONS		4-1
	Planning Future Terminal Requirements	1	4-1
	Resource Management	2	4-2
	Future System Utilization	3	4-3
5	SUMMARY		5-1
	REFERENCES		R-1

LIST OF FIGURES

<u>NUMBER</u>	<u>TITLE</u>	<u>PAGE</u>
1	Summary of Sackman's Findings	1-2
2	Breakout of Programmers' Time	3-1
3	Programmer Time Invested Per Line of Code Produced	3-2
4	Cost Per Line of Code Produced	3-3

## CHAPTER 1

### INTRODUCTION

#### 1. The Problem

The advent of time-sharing systems has introduced a controversy into the ranks of data processing professionals and created a dilemma for managers. As more and more on-line interactive systems reach the market, the controversy over the cost effectiveness of on-line programming intensifies, fueled by widely divergent opinions. Portions of the data processing industry push their time-sharing products with claims of 200-400% increases of programming productivity while pragmatic users trying to verify these claims are faced with the enigma of determining programmer productivity. Meanwhile the profit and production oriented business manager is faced with industry performance claims and the inherent desire of his data processors for the intrinsic attractions of an interactive system that will call for an immediate increase of cash outlay. These factors, coupled with natural fiscal conservatism and the skeptics' warnings, create a dilemma for the manager. Spend the money and take a chance on highly increased productivity? Try an interactive system for awhile and measure the results before total commitment? The second course of action seems to reduce the risk until one examines just how one goes about measuring programmer productivity.

The United States Army Computer Systems Command, in facing this dilemma, recognized the obvious difficulties and expense in performing valid performance measurements. Examining previous findings in this arena, one encounters the work of Harold Sackman<sup>1</sup> of System Development Corporation, who had conducted a number of independent studies and summarized them. The results of these studies are shown at Figure 1. All of these studies dealt with very small programs (a few man-days of effort at most), all of which were academic exercises lacking the "real world" programming environment.

The comparative results of these studies indicate that overall there is a tendency to be slightly more productive at a somewhat higher cost when one uses a time-sharing system. These statistics provide little confidence to the manager who must make the decision to invest in an interactive programming capability, especially since the step toward developing an extensive time-sharing system within a single installation is such an expensive proposition in terms of machine upgrade, terminal purchase, leased lines, control units, extra memory, etc.

Further, it was realized that since these studies were laboratory controlled experiments, the translation of the results in an actual programming environment would be a matter of conjecture having little basis for confidence for management decision making. It was with this information, or lack of information, that in August 1973 a pilot interactive



ON-LINE VERSUS OFF-LINE PROGRAM PRODUCTION:  
COMPOSITE RESULTS OF STUDIES

Study	Man-hours	Computer Time	Costs	User Preference
Air Force Academy (1968 Pilot Study)	Time-Sharing <sup>a</sup> 1.2:1	Batch 1.7:1	Approx. same	Approx. same
Air Force Academy (1969 Pilot Study)	Batch 1.2:1	Batch 1.8:1	Batch	Approx. same
Air Force Academy (Main Study, 1969)	Batch 1.1:1	Batch 1.9:1	Batch	Time-Sharing
Adams and Cohen (1969)	Approx. same (1:1)	Batch 3.8:1	Batch 6.7:1	Batch
Erikson (1966)	Time-Sharing 1.9:1	Time-Sharing 3.4:1	Time-Sharing	Time-Sharing
Frye and Pack (1969)	Not Reported	Batch 3.8:1	Batch 3.3:1	Time-Sharing
Gold (1967)	Time-Sharing 1.2:1	Batch 5.7:1	Approx. same	Time-Sharing
Grant and Sackman (1967)	Time-Sharing 1.6:1	Batch 1.4:1	Approx. same	Time-Sharing
Schatzoff, Tsao, and Wiig (1967)	Batch 2.1:1	Time-Sharing 1.1:1	Batch 1.5:1	Not Reported
Smith (1967)	Instant <sup>b</sup> 1.2:1	Batch 1.5:1	Approx. same	Instant
Median for all studies	Time-Sharing 1.2:1	Batch 1.8:1	Batch	Time-Sharing Preferred

<sup>a</sup>The mode showing a reported *advantage* appears in each box together with its favorable ratio; e.g., this entry shows fewer man-hours for time-sharing at a 1.2:1 ratio.

<sup>b</sup>"Instant" batch is treated in this table as a simulated version of time-sharing.

Figure 1. Summary of Sackman's Findings

programming project was initiated within the U.S. Army Computer Systems Command. The basic goal of this study was to develop an efficient, interactive programming capability and to determine the merits and cost effectiveness of interactive programming within an actual programming environment. A detailed plan was defined concerning installation of the hardware and software, training of the programmers, development of the data collection instruments, and methodology to be used to evaluate the value of TSO\* for the Command. The purpose of this report is to document the experience of the Command in installing and using TSO and to summarize the results of the formal evaluation of the system. (A complete technical report<sup>2</sup> of the findings is also available.)

## 2. Background

The growth of the number and variety of on-line, interactive computer systems has been phenomenal since the early nineteen-sixties and especially since its availability commercially around 1965. By 1967 some 40 such systems were installed. Today these systems and their applications number in the thousands. In addition to airline reservation services, laboratory equipment control, management information systems, computer-assisted instruction, and other information manipulation systems, interactive systems also have been used increasingly for program development, i.e., the coding, debugging, compilation and testing of computer programs.

Although designers and salesmen of interactive systems have had a great deal to say about the improved man-machine functioning of such systems over batch processing, each case must be examined individually. Too many factors are involved to make generalizations regarding the cost and productivity advantages of one mode of operation over the other. For example, early comparisons played up the trade-off between additional equipment costs and such factors as increased problem-solving capabilities and lowered turnaround time. However, recent technological and equipment production advances have made supporting hardware far less expensive when compared to software development costs, to the point that some data are beginning to show actual dollar savings for interactive systems in a broad range of settings and applications. The purpose of the project described below was to determine, in real-world production environment, what the cost and production benefits of one interactive system would be.

---

\*IBM's Time-Sharing Option (TSO) was the real-time software package selected for pilot implementation.

## CHAPTER 2

### PLANNING AND PREPARATION

#### 1. Installing the System

As with most installations, TSO was not the only systems improvement being installed by the Command. The significant changes to the system more or less directly related to the installation of TSO were the following:

First the systems software was upgraded to OS/MVT (from OS/MFT) with HASP. In addition, TCAM (IBM's Telecommunication Access Method) was installed along with the TSO software and a number of program products. The products included the COBOL and Assembler Promoters, the COBOL V4 Compiler, the COBOL Interactive Debug Package, and the TSO Data Utilities. Most of this effort in installing the software prior to hardware modification worked out well, with most problems being ironed out before delivery of the additional hardware.

Hardware improvements related to TSO were also made. A half megabyte of core was added, bringing the system core capacity to 2 megabytes. In January 1974, the interactive hardware was installed on the system. This included ten IBM 3270 CRT display terminals, four IBM 3286 remote printers, and two 3330 disk drives plus associated connecting and interfacing equipment. The complete system was up and running by the end of the month.

#### 2. Programmer Preparation

To make full use of the interactive capability, programmers were prepared by way of a formal 3-day training session. This class covered TSO commands, terminal operations, and general systems familiarization. Some 70 applications programmers were trained. Not all of these people participated in the formal evaluation study of TSO but almost all made at least some use of the system. A users guide was prepared containing detailed explanations and examples of all system commands and step-by-step instructions for using the system. Programmers (and their supervisors) were familiarized with the data collection forms to be used as part of the evaluation study.

In addition to the formal training, programmers had four weeks before and after training for familiarizing themselves with the operation of the interactive system. As it turned out, the programmers could have used more time after training to become more proficient with the system, but as with most projects, everything did not fall in place exactly according to plan.

### 3. Other Preparations

In addition to the system enhancements and programmer preparation mentioned, three other system improvements were made which had significant impact on the system and the evaluation. First, the system's interface between TSO and the source program library was written and installed. This interface permitted retrieval and replacement of all OS programs.\* Second, additional TSO user commands were obtained from the Defense Communications Agency and installed on the Computer Systems Command System. Third, the capability for initiating background jobs via TSO was installed. This Foreground Initiated Background (FIB) capability permitted a programmer to enter jobs into the HASP job queue through a terminal. As it turned out this feature was used very heavily, and in most cases the standard procedure was to modify a program and then submit it via FIB rather than to compile the job interactively. The reasons for this are discussed more extensively later.

### 4. The Evaluation Plan

To ensure a professional, objective and unbiased evaluation of the new interactive system, the services of Integrated Systems Support, Incorporated, and the Human Resources Research Organization were acquired by contract.

A study was designed to measure the cost in personnel time, computer time, and other indicators of the comparative effectiveness and responsiveness of the batch and interactive systems. The primary reason for the acquisition of TSO was for use by programmers developing and maintaining the software systems which are the responsibility of the Command. It was ultimately on this one application of TSO, production programming, that the decision to go or not go with an interactive system was to be made. To provide the best possible true estimates of potential cost benefits, the strategy of the study was to collect data on a large number of regular production jobs\*\* of all sizes, shapes and descriptions.

---

\*The Command also maintains a number of DOS application systems. Some of the programs for these systems are maintained in OS with the DOS code imbedded as comments in the OS code. Only those DOS programs set up in this way could be manipulated using TSO.

\*\*The alternative strategy (i.e., selecting a small number, say 3 or 4, predefined simulated production tasks) had been used before by other researchers. Results of these studies retain a certain sterility because of the controlled environments in which they were conducted. A significant goal of the study was to carry out the data gathering in a real-world situation so that the resulting data and its implications could be related directly to the concerns of managers and programmers in the Command.

The study itself spanned the first six months of 1974, during which detailed plans of the study were finalized and programmers were trained to use TSO. Data gathering then began for all programming tasks completed as a normal part of the work effort. By the end of the project, data had been gathered on 51 programming tasks, 24 of which were completed using the batch mode and 27 using TSO.

CHAPTER 3

RESULTS OF THE EVALUATION

The results of the data are best summarized in response to three questions:

1. Which mode allows programmers to be more productive?
2. Which mode costs less?
3. What else was learned from the study?

1. Productivity Comparisons

As would be predicted by the proponents of interactive systems, TSO was the more productive mode of operation. Several things illustrate this conclusion. First, total time was broken out into productive and non-productive time. (The latter category includes time spent walking to and from the computer center, to and from keypunch facilities, etc.) Figure 2 illustrates that for the time chargeable to completion of a specific job, non-productive time in batch was substantial (almost 16%). For TSO, time in the non-productive category was negligible.

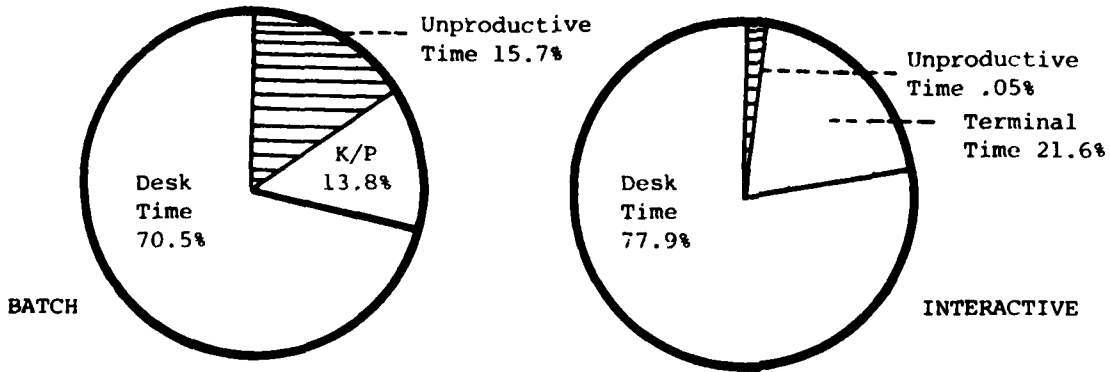


Figure 2. Breakout of Programmers' Time  
(Difference in unproductive time significant at the .99 level of confidence.)

A second measure taken was hours spent per line of code produced. As shown in Figure 3, the number of man-hours per line for batch was twice that for TSO.

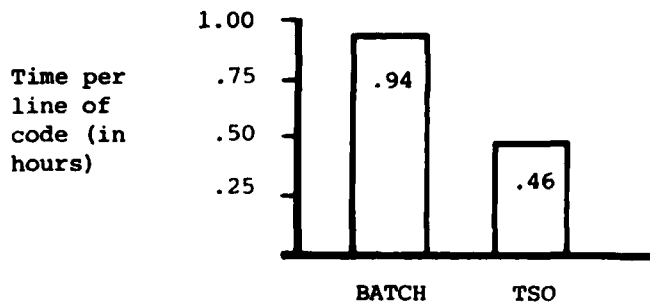


Figure 3. Programmer Time Invested Per Line of Code Produced  
(Difference significant at the .90 level of confidence).

For all jobs, batch required almost an hour per line of code; TSO required .46 hours. A time advantage also was found for doing specific kinds of tasks. For example, JCL procedure libraries were updated and modified using TSO in only two thirds of the time required using batch.

A summary answer to questions about programmer productivity is that in using the interactive TSO system:

- (1) Non-productive time was reduced,
- (2) On a by-line-of-code basis, programmers were twice as productive as when using batch, and
- (3) Particular programming functions are especially assisted by using the features of TSO, e.g., modification of JCL procedure libraries.

## 2. Cost Effectiveness Comparisons

A second question posed above was: Which mode costs less? On a by-line-of-code basis, TSO was actually cheaper than batch by about a 3 to 5 ratio, i.e., TSO was 63% of the cost of batch. Thus, in spite of the greater equipment costs (additional hardware and software costs amounted to \$22.03 per hour more than batch) increases in productivity of applications programmers more than compensated for the additional equipment investment. The differences in cost are illustrated in Figure 4.

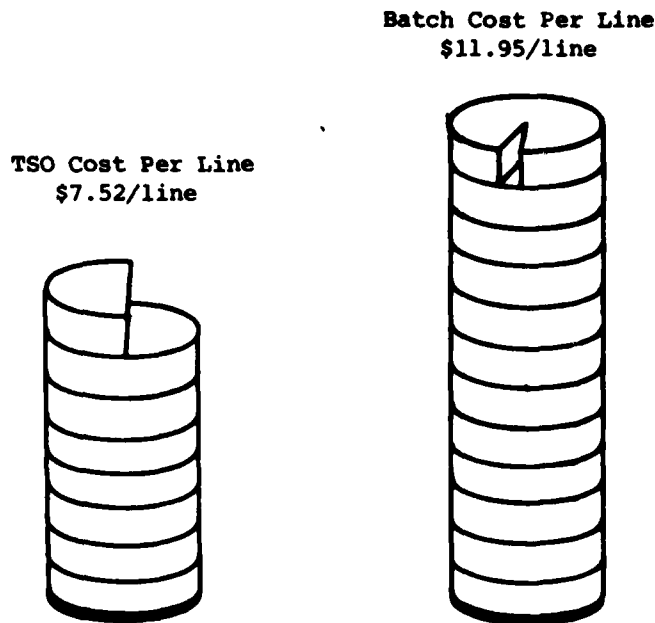


Figure 4. Cost Per Line of Code Produced  
(Difference significant at the .90 level of confidence)

Suffice it to say that the results of the study demonstrated the cost-effectiveness and utility of using TSO as a tool for program development and maintenance by the programming staff of the Computer Systems Command.

### 3. Other Findings and Lessons Learned

Some of the most valuable information obtained from a project like this may not be the formally collected data, but the incidental information learned from the process of acquiring, installing and using a tool such as interactive programming.

One lesson learned has to do with the specific kinds of programming tasks done most easily by TSO as opposed to batch. For example, the evaluation data showed that modifying procedure libraries is more easily accomplished using the interactive system. This data was collected from only a few programmers. Other programmers stressed using the terminal for test data manipulation. Some felt everything should be done interactively. Some used TSO and batch depending on the job, its urgency, etc. Others simply preferred batch.



To some extent there is an analogy between a comparison of batch and TSO and a comparison of a calculator and an abacus. There are still occasional contests between the two and depending upon the skill of the operator, the operation of the piece of equipment, and the specific task to be performed, the abacus, an "antiquated" instrument, could outperform the calculator.

The lesson to be learned, as implied in this analogy, is that the interactive system and batch system are tools at the disposal of the programmer. Depending on such factors as skill, work habits of the programmer, work load, efficiency, and the task to be performed, one or the other of the systems could be appropriate.

What, then, is the most effective way to get things done in most cases?

- (a) Entering code for a small program or routine is more effectively and efficiently done by the programmer using a terminal than punching cards by a time factor of about 2 to 1.
- (b) The cutoff point between "large" and "small" programs depends upon the dexterity of the programmer using a terminal.
- (c) Longer programs should be entered by submission to a centralized keypunch facility or by training a clerical person to enter programs via the terminal.
- (d) The benefits of interactive debugging are quickly offset if response time is poor or if there is a high programmer to terminal ratio.
- (e) The benefits of interactive programming are dependent on the features included in the supporting software. For example, the feature which made procedure library modification and program conversion easy was the change-all command.
- (f) Interactive programming necessitates an efficient on-line source code library system. Without it the system is only a programmer's toy.
- (g) If proper accounting procedures are to be maintained, entry into the system must be achievable within seconds. This is one problem that impacted the study. As it was, one programmer in each group logged onto the system in the morning, and all other programmers used the terminal regardless of whether they were working on the same project, different

projects, or going through a terminal orientation. This situation existed because of the time it takes to get into the system.

- (h) If a programming house trains its own programmers, training should incorporate use of TSO from the beginning. Once a programmer establishes his own work patterns, it is hard to reorient to use a new tool such as TSO efficiently.
- (i) Testing of a multi-program applications system presents a real challenge to the system, particularly if extensive indirect access devices are used for data file storage. Usually, system testing is purely a batch function.
- (j) There is a natural tendency for managers and project teams to become proprietary. That is, once a terminal is in place, the rights to use of that terminal become highly guarded. Although this situation may not exist where there is a low programmer-to-terminal ratio, some kind of flexible priority system is required.

Programmers usually are adaptable problem-solvers. This was demonstrated in several ways. It became apparent, for example, during the early months of using the interactive system, that interactive debugging in some cases was taking as long as for a FIB job to be in and out of the system. Programmers did not stick with the interactive system. They usually used TSO to modify their source data sets and used the FIB procedure for compilation, because this procedure freed up the terminal and they had time to work on other things.

Other examples of programmer adaptability include the fact that they had to get used to the 7-to-1 programmer-to-terminal ratio. Although there were instances of inter-project contention for time on the terminal, there were more instances of accommodation and cooperative arranging of schedules to mutual satisfaction. An estimated two to four programmers could use the terminal for retrieval, modification and background submission for a program test in the same time it would take a single programmer to retrieve, modify and interactively compile in foreground. This is becoming less true as the interactive system continues to be tuned and the system resources are shifted to support more interactive work.

Programmers also adapted to relatively slow response time. Especially during high demand hours, the system would occasionally take several minutes to execute a command. Nonetheless, the programmers used the system productively as the formal evaluation data showed.

The real lesson learned here was that programmers can deal effectively with interactive systems which may be good but may not always work optimally.

Another thing learned was that the quality of the structure of the code was probably improved by using TSO. Programmers evidenced a tendency to insert comments, properly indent hierarchical coding structures, and perform other clean-up tasks made easy by terminal. They said that they would not perform such changes with a keypunch. The question is: Will this result in measurably lower costs for maintenance of those programs? Presumably, if programs are more readable and contain internal documentation, they can be more readily modified.

Some programmers pointed out that the real savings, using the terminal, were realized in tasks ancillary to development of a program. TSO made it simple to dump files, to create or check test data, or to modify test data to check a specific program condition. It made it easier to insert utility programs into the batch job stream, which in batch mode would have required scratching some JCL on a sheet, finding a free keypunch, punching the set-up, walking to the computer center and repeating that trip to get the output back. Finally, the terminal is the best way to keep track of jobs in the queue or being processed. It reduces the lost time a programmer spends trying to find out how soon he can go back to work.

One disappointment was the apparent lack of use of the COBOL Interactive Debug Facility. Perhaps this was caused by the requirement to use the V4 COBOL Compiler with the Debug package. In order for the USACSC Headquarters to maintain compatibility with the field, the V2 Compiler must be used after the V4. Another probable reason was the high contention among programmers to have time on the terminals. Peer pressure did not encourage a programmer's tying up a terminal for an interactive compilation.

A final comment from the programmers broached the subject of supervision/management. Their comments centered on the fact that when a terminal wasn't available, it wasn't always easy to shift into batch mode or to find some other way of being productive. Some inferences can be drawn from this commentary. For example, be sure that enough terminals are available and consider modifying management techniques to ensure efficient use of the terminal as a resource.

There is another set of lessons learned which deal more with the evaluation study rather than with interactive programming per se.

As much time should be taken in preparing all levels of the organization for the study as is used in actually collecting data. Programmers must be trained in the use of the system but it is just as important to train them to use data collection forms. Supervisors

must understand their responsibilities in assigning tasks and ensuring proper documentation of the effort for each task. Upper management must be cognizant of the impact of the data collection effort on completion of the normal work load and make appropriate allowances, especially in the area of overall man-hours. The systems support staff must be ready to provide for capture of needed machine utilization statistics and the comptroller must be involved in defining costing algorithms for establishing comparable batch and interactive programming costs.

As part of the internal public relations effort which must occur as a part of the overall effort, the thrust of the study must be the evaluation of the method of operation, not the people involved. This must be emphasized repeatedly. The study and use of the data by management should not be used for personnel or project performance assessment.

#### 4. The Study in Retrospect

Determining programmer productivity is not an easy task. It is possible, however, to come up with good information even in a production environment. Planning for the data collection must be meticulous and involve the participating programming groups. The plan must be flexible, and at the same time strictly enforced, to produce valid results. The actual data collection period for this project was 3 months. This time period was adequate; however, a longer period would have allowed collection of more data and more detailed analyses would have been possible. The trade-off is that as time increases, controls change and additional factors influence the variability of the results. We opted for less time and more uniform results.

The coordination of the data collection effort is facilitated by a highly conversational mode of operation between project staff and the programmers and their supervisors. Weekly meetings to discuss problems proved to be an absolute necessity in the Command's study.

More valuable information is obtained from experienced programmers than from programmer trainees. Although the pressures and changing situation of the real-world work situation make data collection a more difficult task, it is only the experienced programmer working at his regular job who can provide a thorough assessment of the systems with which he must work.

One of the highlights of this project was the smooth installation and growth of the Interactive System. It is a credit to the project office, the system programming staff and the IBM contractor brought in to assist in the project. Great assistance in solving problems and locating software enhancing packages was provided by participation in SHARE, INC.

This project presented a paradox to personnel training and terminal usage. Over 70 personnel received training. Twenty of the 70 were participants in the evaluation. The other 50 provided normal terminal contention and presented no real problem. When the 40 interns were trained, however, terminal contention peaked to a point that when intern programmers were using the terminals, productivity of those using the terminals was significantly impaired.

In a project of this nature there is great anticipation by upper management and, as a result, project management, to show results within a short time frame. In an attempt to stay on schedule, some familiarization time was sacrificed to expedite the beginning of data collection. The result was some wasted effort due to participants' lack of understanding of both the proper way to fill out data collection forms, and of the use of the interactive system.

And finally, it was reassuring to see some management personnel change from a state of apprehension toward IAP replaced by total acceptance of IAP and some apprehension concerning any possibility that it might be removed.

## CHAPTER 4

### IMPLICATIONS

At this point we address the question of how TSO will affect management decisions and operations in the Command.

#### 1. Planning Future Terminal Requirements

One of the first tasks is the orderly planning for additional equipment. Requirements must be defined for additional terminals, physical placement of the terminals, and additional disk space for work areas and temporary files. The information from the study provides the following guidance in such planning:

In planning for number of terminals needed, estimate the number of terminals on about a five programmer-per-terminal ratio. This is only a guideline and depends on the programmers, the tasks, the response time and many other variables. In some cases, two programmers per terminal may be too many; in others, one terminal may be sufficient for ten people. To be more specific: First, if a programmer is using a terminal for JCL procedure changes, he can use it efficiently for as long as he has work to do; theoretically, each programmer could have his own terminal. Since fatigue becomes a factor after several hours, a 2:1 ratio (programmers to terminals) may be sufficient. The same rule can apply to program conversions and to any other task when the programmer must plow through lengthy data sets, making numerous changes before handing the data set back to the computer. Second, if programmers use the terminals for following batch jobs through the queue or for simply modifying programs and initiating background processing of their programs, a ratio of 5:1 is sufficient.

Another factor determining the number of terminals is the cyclical nature of the work load. Some of the software systems supported by the Command undergo quarterly modifications and maintenance cycles. Because of this, there are times when everyone in a programming group is using the terminals for completing their tasks, correcting program "bugs," or modifying a routine, etc. There are other periods when few in the group need time on the terminal, e.g., during system test or slack periods prior to receipt of a change package. The point is, the number of terminals required should be estimated by requirements at the peak period--not the slack period, and not "on the average."

Another factor might be called the "fire fight factor" ( $f^3$ ). That is, if tasks which are to be performed demand very quick turnaround, terminals should be available to facilitate the process. The fact still remains that, other factors being equal, TSO is the fastest way to perform any task. A project with a high  $f^3$  situation may need a 1:1 ratio.

Placement of terminals is a planning consideration, especially from the man/machine system point-of-view. Although physical limitations of the equipment obviously must be satisfied (e.g., in some cases terminals be no more than 2,000 feet--or some other relatively short distance--from the interfacing hardware). Terminals should be in a relatively secluded work area to minimize interruptions; the work area should provide table space for reference and other materials; in no case should the terminal be placed in the middle of an open area which has a lot of traffic; for typewriter terminals (e.g., 2741s), the terminal should be in a separate small room, if possible, because of their noise level.

Because work loads fluctuate, some consideration should be given to providing a minimum number of terminals to a programming group at the higher (8:1) ratio so that additional terminals might be installed temporarily when the work load requires (with any overall ratio of, say, 3:1).

A final consideration in planning and placement of terminals is establishing the types of terminals required. Most of what has been said so far assumes that the principal device will be the 3277 CRT. There are also 2741 typewriter terminals and a variety of remote printers. The trade-offs between CRTs and hardcopy terminals are well established and need no review here. The Command study made use of CRTs exclusively, and for most of the work, the speed of the CRT far outweighs the advantage of a hardcopy record of the typewriter terminals. There probably is reason for having a few hardcopy terminals but the CRT should constitute the bulk of the terminal hardware.

## 2. Resource Management

In addition to aspects of future hardware planning, there are a number of other points under the general rubric of resource management which will need management attention.

First, a decision might be made on how best to perform programming steps given the TSO environment. As an outcome of the evaluation study, the generalized steps of programming were reviewed and recommendations were stated as to how to best perform each step. However, the question to be raised here is not how to perform each step best but whether management should require that each step be performed in a recommended way. The answer depends on the management of a particular installation. On the one hand, only the programmers know whether a terminal is available, what the priority is of the task, how long it will take him/her (not how long it should take Mr/Ms Average Programmer) to do what's needed. The programmer needs the prerogative to perform the steps best, given the tools he has at hand. The environment changes too quickly for detailed rules (on whether to punch a card or key in a statement to perform a particular step for a particular program) to be of value.

Second, managers' time should not be taken up enforcing counterproductive rules. On the other hand, programmers should be given guidance based on what has been learned on how to most efficiently use the system.

A second point under this general topic is the possible need for definition of new jobs. The specific job of concern here is the job of terminal operator (TO). A TO is needed in the cases where a great deal of code or other data needs to be keyed into the terminal. The common example would be in keying in a newly written program. The most efficient and cost effective means for doing this is to have a specially trained clerical person take the coding sheet and input the program into the system. This frees the programmer (who probably isn't much of a typist anyway) to perform some other task to which his salary is more suited. Once the source data set is created, the programmer can quickly check and visually verify the code and submit it for compilation.

The general point to be made is that some redefinition of jobs may be called for with the installation of TSO and a formal job analysis may help in defining how best to structure the jobs.

The third point was alluded to earlier and has to do with management of the peripheral hardware. It would not be cost effective to have the maximum number of terminals required in each programming group at all times. It would be more appropriate to have a minimum number, but have in addition, the capability for temporarily installing additional terminals as the workload required. The trade-off is that lines will have to be installed for the temporary hookups and someone will have to manage and control the pool of temporary terminals. The evaluation study did not perform a cost analysis comparing the savings of a temporary terminal pool approach, but the potential savings would have to be compared with the personnel and logistics problems innate to such an approach. No solutions are offered here, only the presentation of the problem.

### 3. Future System Utilization

Once the hardware/software system is enhanced to support interactive operations, careful consideration should be given to additional interactive applications: management information systems providing flexible real-time information retrieval and hardcopy formatting; computer assisted instruction packages providing self-paced, highly motivating training in programming and any other topic of concern to personnel in the Command; text processing packages to reduce the clerical work load of changing documentation reports, and the like, which usually undergo several iterations before final acceptance. One specific application which might be considered is a flow process control system for use as a management tool for controlling change package processing.



## CHAPTER 5

### SUMMARY

The purpose of this report was to present the key findings of the evaluation of interactive programming and to discuss the implications and lessons learned for management planning. Special emphasis was given the actual task of the programmer and the impact which TSO will have on the programmer carrying out his day-to-day activities.

The evaluation showed that increases in productivity more than made up for the increased expense of the additional hardware and software needed to support the interactive system. With the increasing management interest in the system, the concerns addressed in this report are being dealt with. The prospects for increasingly effective and efficient use of interactive programming are excellent.

REFERENCES

1. Sackman, Harold. Man-Computer Problem Solving. Princeton, N.J., Auerbach Publishers, Inc., 1970.
2. Reaser, J., Priesman, I. and Gill, J. A Production Environment Evaluation of Interactive Programming. Technical Report USACSC-AT-74-03, December 1974, Integrated Systems Support, Inc., USAMERDC Contract Nonr DAAK02-72-D-0529, U.S. Army Computer Systems Command.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A 110 4160	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Interactive Programming: Summary of an Evaluation and Some Management Considerations		5. TYPE OF REPORT & PERIOD COVERED Final one-time issue
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(S) Reaser, Joel M. and Carrow, John C.		8. CONTRACT OR GRANT NUMBER(S) DAAK02-72-D-0529
9. PERFORMING ORGANIZATION NAME AND ADDRESS INTEGRATED SYSTEMS SUPPORT, INC. 5827 Columbia Pike Falls Church, Virginia 22041		10. PROGRAM ELEMENT PROJECT TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Computer Systems Command ATTN: CSCS - TSO Fort Belvoir, Virginia 22060		12. REPORT DATE March 1975
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Not Applicable		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Not Applicable		
18. SUPPLEMENTARY NOTES None		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Interactive vs. batch processing Evaluation Programmer productivity Time sharing programming		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report summarizes the results of an evaluation of interactive programming versus batch programming within an actual software production environment at the U.S. Army Computer Systems Command. The purpose of the study was to determine productivity and cost effectiveness differences between the two modes of operation. The results of the study indicate that, on a line-of-code basis, the interactive system offers an increase in productivity and a decrease in overall cost.		

20. Abstract (Continued)

Based on the data gathered formally and informally from the programmers, topics are discussed which should be considered in management planning for interactive programming.

END

DATE  
FILMED

3-82

DTIC